Foundations of Computer Vision CSCI-631

Final Project Report

Anushka Yadav (ay4034)
Vedika Painjane (vp2312)
Eshaan Deshpande (ed6939)

Abstract

The goal of this project is to create a method that uses object segmentation and machine learning to determine whether or not a given cotton image has been subjected to irrigation. Our system takes as input a RGB cotton image, and its task is to determine if the crop was Rainfed, Fully irrigated, Percent deficient, or Time delay. It is assumed that the cotton plant is somewhere in the middle of the picture and that the edges have nothing to do with it.
Initially, object segmentation is utilized to first isolate the cotton plant from its immediate surroundings. Then, the photos and labels are used to train a Convolution Neural Network classifier. The supplied dataset of labeled cotton images is used for both training and testing the classifier. Thus far, promising outcomes have been observed, with a 94% accuracy rate on the test dataset.

Introduction

The problem addressed in this project is the classification of the irrigation treatment of cotton plants in an image. The objective of this challenge is to determine which of the four irrigation treatments; Rainfed, Fully irrigated, Percent deficit, or Time delay, a cotton plant received based on an image of the plant that was provided as input. It is presumed that the cotton plant may be found in the middle of the image, and that the outskirts of the picture do not show any information that is pertinent to the plant.

Techniques such as object segmentation and image classification are utilized throughout the project in order to accomplish this objective. The input data is made up of RGB pictures of cotton plants and the labels that go along with them, all of which are stored in their own respective files. The Deep Learning Toolbox in MATLAB was used to train a custom neural network consisting of three convolutional layers and two pooling layers. The model was trained through the use of stochastic gradient descent with a learning rate of 0.001 over the course of 10 epochs, and the batch size taken was 32. The final model attained an accuracy level of approximately 94% on the dataset used for validation.

For the purpose of validating the model, a script known as predict.m was developed. This script preprocesses the photos and makes predictions on each image contained within the "images_for_prediction" folder. The script begins by loading the model and the images, then it preprocesses the photos by performing morphological operations on them, it then feeds the segmented image into the machine learning model, and it concludes by making a prediction on

each image. On the console, the results of the predictions will be displayed. In addition, a visualization of the model's predictions was generated by using the make_montage.m script. This script takes six photos from each irrigation treatment class and generates a subplot of the model's predictions.

Overall, the project demonstrates the capability of applying machine learning approaches to accurately categorize the irrigation treatment of cotton plants. This capability has the potential to have substantial applications in precision agriculture and crop management.

Methodology

The segment.m file contains the MATLAB code for segmenting the images. The photos of the cotton were saved in the ".TIF" format. In order to get started, the last channel of each and every image was removed so that it could be utilized in the pre-processing steps prior to the segmentation being performed. This code is designed to extract just the cotton part from the given images. Following the reading of each image in an iterative for-loop, the images are segmented before being saved in a folder called "processed_images." After a lot of trial and error, a threshold value was finally settled upon, and it was set to 114 for the red channel of the image since the red channel of the image offered the best segmented output. This decision came about as a result of the fact that the red channel of the image gave the best segmented output. It was determined that the threshold value for both the green channel and the blue channel should be 0.

In addition, a structuring component with the parameters of 'disk' and the value of ‘2’ was constructed. In order to obtain the best possible segmented image, several different morphological processes were performed utilizing the structural element.
After removing the fourth channel from the image, the red, green, and blue channels were successfully extracted. A thresholding operation was carried out on each and every cotton image by applying the threshold values to each of the image's three channels. The color thresholds are used as the basis for the creation of a binary mask. Creating a logical array for each color channel that returns a value of true if the intensity value is greater than the relevant threshold value and a value of false in any other case is what is required to accomplish this goal. The three logical arrays are then combined using element-wise AND using the '&' operator, which results in a single binary mask that is true only in the case where all three conditions are satisfied.

After that, a binary mask is constructed by employing the thresholding, and each pixel of the RGB image is multiplied by that mask. The binary mask is replicated along the third dimension by the repmat() function so that it perfectly fits the dimensions of the RGB image. The new image that is produced when the RGB image and the binary mask are multiplied together is one in which the only pixels that remain are those that correspond to the color that is specified by the mask. This is due to the fact that in the RGB color space, only the pixels that have values that are greater than zero in each of the three channels will contribute to the final output image. The resulting image is then changed from color to grayscale with the help of a built-in function in MATLAB known as rgb2gray() [Figure 1].

Figure 1

In addition to that, the grayscale image has been binarized. Following the process of binarization, a morphological operation known as bwareaopen() [Figure 2] is carried out.

Figure 2

This operation accepts the binarized image as a parameter, and the value of the second parameter, which specifies the number of pixels, is set to 100. This is done in order to remove from the image any small items that have a lower number of pixels than the amount that has been chosen.

During the process of segmentation, a number of morphological operations, such as imclose(), imfill(), imnoise(), imopen(), imerode(), and imdilate(), were carried out. However, only a few of these operations were successful in producing a good segmented image. Therefore, following the execution of the bwareaopen() function, the imerode() function [Figure 3] was used.

Figure 3

This function, which performs erosion on an image, was used to remove the boundary pixels of the objects contained within an image. This is accomplished with the help of the organizing element defined at the beginning. The following step was to apply the imopen() function [Figure 4] to the image that was obtained following the erosion process.

Figure 4

imopen() will first perform an operation known as erosion, and then it will perform an operation known as dilation on the image that is produced. This procedure will eliminate little things from the image while maintaining the shape and size of the larger objects. The goal of this operation is to remove small objects from the image.

After finishing the procedure described above, you will do the imerode() operation [Figure 5] once more so that the image's noise can be reduced even further.

Figure 5

Last but not least, the imdilate() function [Figure 6] is utilized to smooth out the edges of the objects, fill any holes that may exist within the objects, and fill any minor gaps that may exist between the items.

Figure 6

At the conclusion of this procedure, the correct segmented picture will have been obtained, and it will be put to use in the repmat() function [Figure 7] so that masking may be carried out.

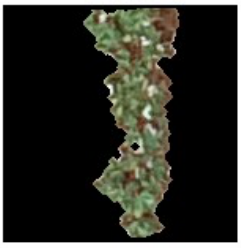


Figure 7

When calling MATLAB's repmat() function with a segmented image as a parameter, the image passed in is treated as a binary mask. The repmat() function replicates the binary mask which is

the segmented image three times along the third dimension to make it the same size as the RGB image. This is done to ensure that the binary mask can be used to set the RGB values of the pixels in the final image.
The non-zero RGB values of the matching pixels in the replicated dilate mask are then set to zero in the final image, effectively masking them out. The original image is retrieved, but just the cotton portion is displayed, creating a black background where the segmented object was. Finally, the segmented images that will be used to train our machine learning model have been saved to the 'processed_images' folder.

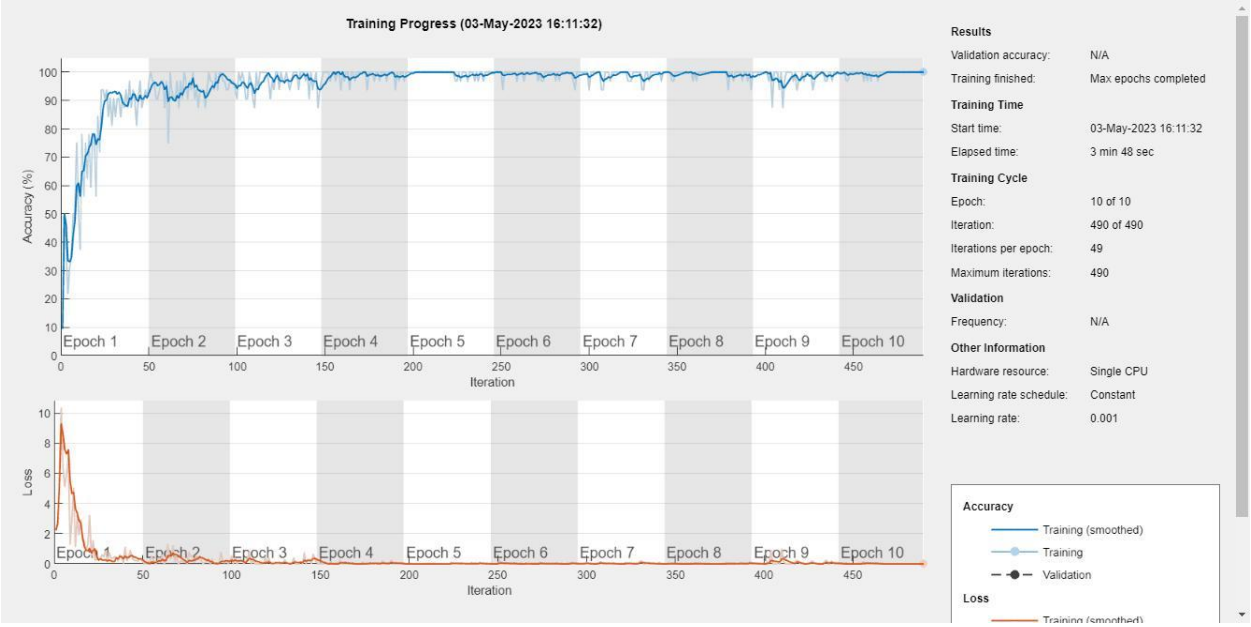
Training and predictions:-

Once the images are preprocessed, we need to create a datastore for these images. This datastore will be used by the machine learning model to train. To create this datastore, we need to associate labels with the image name. Since matlab reads all images in the directory starting from image names "...1 ", thus all images numbered 1000+ are read first. To accommodate for this, we created a final_labels.csv file which has images ordered by the order in which matlab fetches these files from the directory and their specific labels. This file was created manually. This completes the setup required for creating the datastore.

Once the setup was completed, the script datastore.m was used to create the datastore. This script first reads the labels from the file final_labels.csv. Then it reads images from the "cotton images" directory. Since our neural network was trained for images of size 149x149, the code then resizes the read images to this dimension. It then adds the label information to every image file and creates the image datastore that our machine learning model will need. This datastore is then saved to the project folder.

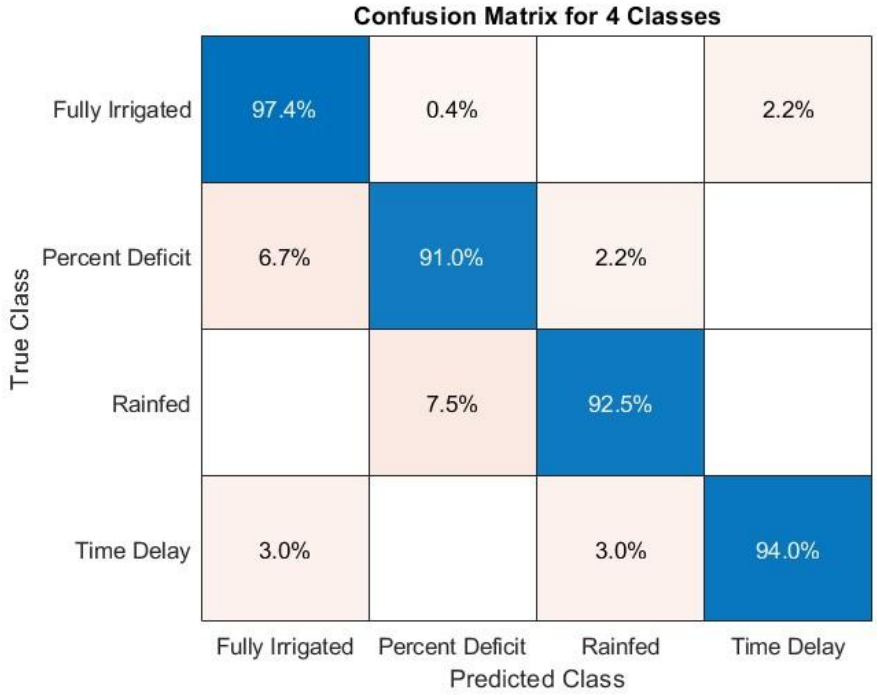
After the datastore is created, we can use the datastore to train the machine learning model. The script training.m is used to train the model. We used the splitEachLabel() function in matlab to split the datastore into training and testing data sets. This function takes all the different labels into account so that both our training and testing datasets contain each label. We split the data 70% into the training dataset and 30% into the validation dataset. We decided to use Matlab's inbuilt Deep learning toolbox for implementing various machine learning models. After trying different models like the Naive Bayes classifier and SVM classifier, we decided to use a Convolutional Neural Network as our machine learning model. We wrote a custom network for the model which contains 3 convolutional layers and 2 pooling layers. The softmax activation function was used for the last layer, which was also modified to output the 4 classes we have in our label files. We tried many different settings for training this machine learning model and found that stochastic gradient descent method works best for loss minimization. We trained the model for 10 epochs with a learning rate of 0.001 and since we have approximately 450 images of each class, we decided to use a batch size of 32.

The accuracy and loss plot we got from training the model is shown below:-



As you can see in the above image, the loss was minimized after epoch number 5. We trained this model on a single CPU instead of using the GPU as our training dataset was comparatively small.

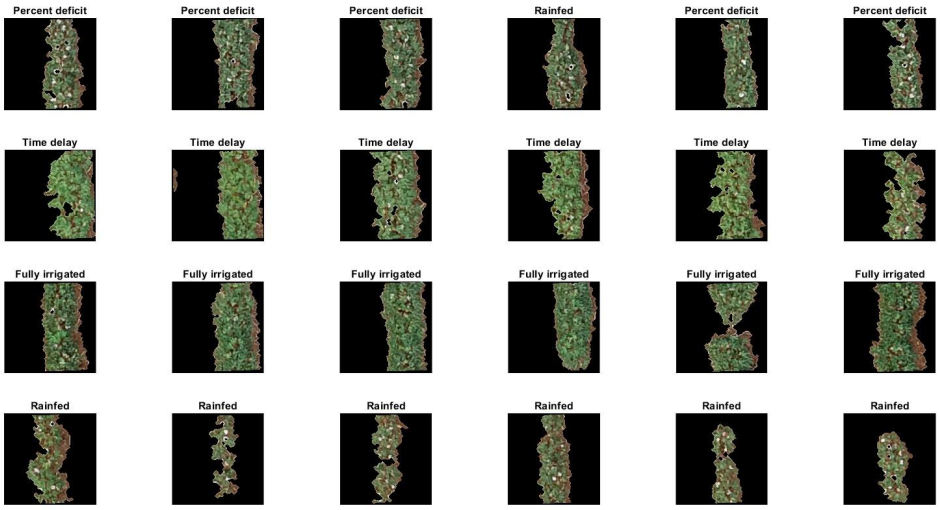
Once the model was trained, we evaluated the performance of the model on the validation dataset and got an accuracy of 94.486%. The confusion matrix for every predicted label is shown below:-



As shown in the above matrix, the model has a 90%+ accuracy of predicting the irrigation classes correctly. However, the model sometimes makes wrong predictions on neighboring

classes because there are many similarities in the images (width of the cotton) in neighboring classes.

Once this model was trained, we saved the model in the project folder. Then we used the script make_montage.m to create a visual representation for the predictions the model has made. In this script, we selected 6 images at random from every different class in the processed_images directory. We fed these images into our machine learning model and created a subplot of the predictions our model makes. The montage we got from this is shown below:-



As seen in the above image, the model is good at differentiating the irrigation methods required based on the width of the cotton detected.

To make testing easier, we decided to write one more script called predict.m which preprocesses the images and makes predictions on all images which are in the folder named “images_for_prediction”. This script first loads the model from the project folder and then loads the images from the “images_for_prediction” folder. It then preprocessed these images according to the morphological operations we performed for segmentation. Once the segmented image is created, it feeds this segmented image into the machine learning model and makes a prediction on that image. An example output of this script is shown below:-

Results & Discussions

In this project, we aimed to develop a machine learning model for accurately classifying cotton plants into four classes namely Rainfed, Fully irrigated, Percent deficit, and Time delay. The first step in this process was to segment the cotton leaves from the background in the images, so that the images could be fed into the machine learning model. To achieve this, we developed a program using various morphological operations.

The first step was to drop the last channel (4th channel) of all the images, and extract the red, green and blue channels of the images. We then set threshold values for each of the color channels and performed thresholding on the images, creating a binary mask based on the color thresholds. The tricky part here was to get the best threshold value that would work for majority of the images, if not all. This was achieved by trying different values and checking the output to see if it was best suited in this scenario.

The binary mask was then used to extract the cotton part of the image, while removing the background.

During this step, it was important to make sure that none of the segmented images contained any of the grass surrounding the cotton plant.

Furthermore, various morphological operations were utilized, like erosion, dilation and opening, to further clean up the image and remove any noise that might interfere with the accuracy of the machine learning model. While performing the morphological operations we realized that imnoise(), imfill() and imclose() didn't give us the expected results. We tried a combination of the aforementioned morphological MATLAB functions to get the most optimal output. This was a fun but tedious task which gave us a better understanding of how these functions can be used in combination to get a better segmented image.

After performing these operations, the final segmented image was stored in the folder named 'processed_images'.

The segmentation process that we performed was successful in separating the cotton part from the background in the images. The machine learning model, a custom Convolutional Neural Network (CNN) model built using the Deep learning toolbox in matlab, was then trained on these segmented images, which allowed it to accurately classify the different types of cotton plants into the four classes mentioned above.

Our model achieved a high accuracy of about 94% on the validation dataset and was able to differentiate between the irrigation methods required based on the cotton detected. However, the model sometimes made wrong predictions on neighboring classes due to the similarities in the images. One possible solution to this problem is to use a larger dataset with more diverse images to train the model. Another solution is to use more advanced techniques such as transfer learning to improve the model's performance.

Overall, the segmentation process was an important first step in the development of the machine learning model, and we believe that the resulting segmented images greatly improved the accuracy of the model in classifying cotton leaf disease

Conclusion

The overall method we followed for this project can be described as follows. We first identified the color channel and thresholding parameters to segment the cotton part from the image. We then used morphological operations to properly segment out the cotton part from the image. Then we used these images to train a Convolutional Neural Network to predict the class of each image. We then tweaked the training parameters of the CNN to get an accuracy of over 94%. Our overall results after this process were pretty good, that is our program could correctly determine the irrigation technique that had been used on the cotton plants.

We are proud of the accuracy of the machine learning model because we created the entire CNN from scratch. Writing the model according to the input images, and then tweaking the meta parameters of the model to get a 94%+ accuracy was really satisfying.

Despite having enough object segmentation experience from our labs, working on the object segmentation part in this project to separate out the cotton plant was challenging. We had to use multiple morphological techniques like erosion, closing, opening and dilation and had to experiment around with these techniques in order to get a good segmentation for the cotton. However, working on this real world problem of detecting cotton was thoroughly enjoyable and fun.

Our results for the project were really good. We could get the cotton part correctly in almost every image and after feeding this image to the neural network, it was really accurate in determining the irrigation technique used. One more thing we experimented with is feeding the full image to the neural network instead of the segmented image, since we thought that there would be more parameters that the neural network could accurately identify in the full image. This hypothesis turned out to be true as after feeding the full image instead, we got an average accuracy of more than 98%. However, to go with the theme of the project, we used the segmented image for training our model.

We learned a lot from this project. Making a project based on real world problems always poses unique problems that we have to think out of the box to solve. We learned how to segment irregular objects from an image. We also learned how to make the object segmentation code work more dynamically and make it give clearer and better segmentations. We learned how to create a custom neural network model for our requirements, and how to train it and improve its accuracy. Thus, after completing this project we got the confidence of being able to solve real life problems using computer vision and machine learning and we are really excited to try out our knowledge on more real world problems.

Working on this project was really enjoyable as it tested us and taught us to look at problems from a different perspective. We enjoyed experimenting and finding out the parameters for object segmentation and the prediction part made us combine our machine learning knowledge with computer vision, thus widening our thought process.