

# AI-ENABLED BIG DATA PIPELINE FOR PLANT PHENOTYPING AND APPLICATION IN COTTON BLOOM DETECTION AND COUNTING

by

AMANDA ISSAC

(Under the Direction of Javad Mohammadpour Velni)

## ABSTRACT

With a rapidly growing global population, meeting increasing agricultural demands has become imperative. Smart farming, powered by machine learning, has the potential to address this issue but faces hurdles in managing big data with high velocity. In this study, we implement a comprehensive big data pipeline for cotton bloom detection that utilizes Azure cloud computing resources and employs YOLOv5 for real-time and batch processing. The model achieves a high mean Average Precision (mAP) score of 0.96 for cotton bloom classification using 2021 data. We also explore the use of Principal Component Analysis (PCA) as a compression method to optimize pipeline execution time and storage space. Rigorously tested for scalability on distinct 2022 data, our pipeline incorporates downsampling with masking as an effective pre-processing step to reduce computational overhead while preserving accuracy. This research underscores the potential of cloud computing in driving efficient big data processing in precision agriculture, enabling accurate crop yield prediction through advanced plant phenotyping techniques.

INDEX WORDS: Big Data Pipeline, Lambda Architecture, Plant Phenotyping, Cotton Bloom Detection, Computer Vision, Principal Component Analysis

AI-ENABLED BIG DATA PIPELINE FOR PLANT PHENOTYPING AND APPLICATION IN  
COTTON BLOOM DETECTION AND COUNTING

by

AMANDA ISSAC

B.S., University of Georgia, 2021

A Thesis Submitted to the Graduate Faculty of the  
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

MASTER OF SCIENCE

ATHENS, GEORGIA

2023

©2023  
Amanda Issac  
All Rights Reserved

AI-ENABLED BIG DATA PIPELINE FOR PLANT PHENOTYPING AND APPLICATION IN  
COTTON BLOOM DETECTION AND COUNTING

by

AMANDA ISSAC

Major Professor: Javad Mohammadpour Velni

Committee: Glen C. Rains  
Khaled Rasheed

Electronic Version Approved:

Ron Walcott  
Dean of the Graduate School  
The University of Georgia  
May 2023

# ACKNOWLEDGMENTS

First and foremost, I want to express my heartfelt gratitude to my advisor, Dr. Javad Mohammadpour Velni. He has been an incredible source of guidance, support, and encouragement throughout my graduate studies, and I could not have completed this journey without him. I am also immensely grateful to my committee members, Dr. Glen C. Rains and Dr. Rasheed Khaleed, for their invaluable support and guidance during my Master's research and coursework. I would like to take this opportunity to extend my sincere thanks to all of my instructors for their exceptional teaching and unwavering assistance throughout my degree program. My lab mates have also been an integral part of this experience, and I am grateful for their help in the lab. Last but not least, I would like to express my wholehearted appreciation to my family and friends. Their constant support, encouragement, and faith in me have been an immense source of strength and motivation, and I am deeply grateful for their presence in my life.

# CONTENTS

<b>Acknowledgments</b>	<b>iv</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>ix</b>
<b>1 Introduction and Motivation</b>	<b>1</b>
<b>2 Development and Deployment of a Big Data Pipeline for Field-based High-throughput Cotton Phenotyping Data</b>	<b>3</b>
2.1 Introduction . . . . .	4
2.2 Dataset . . . . .	7
2.3 Development of Lambda Architecture Pipeline . . . . .	10
2.4 Offline YOLOv5 Model Training . . . . .	18
2.5 Finetuning and Results . . . . .	20
2.6 Future Directions . . . . .	24
2.7 Conclusion . . . . .	24
<b>3 Dimensionality Reduction of High-throughput Phenotyping Data in Cotton Field</b>	<b>26</b>
3.1 Introduction . . . . .	27
3.2 Data Collection . . . . .	28
3.3 Data Processing . . . . .	29
3.4 Conclusion . . . . .	38
<b>4 Optimizing a Build Pipeline for Processing Large-scale 2022 Data</b>	<b>39</b>
4.1 Introduction . . . . .	40
4.2 Dataset . . . . .	43
4.3 Methods . . . . .	43
4.4 Discussion and Future Work . . . . .	53
4.5 Conclusion . . . . .	53
<b>5 Conclusion</b>	<b>54</b>



# LIST OF FIGURES

2.1	Aerial view of our cotton farm in Tifton, GA displaying 40 rows of cotton plants, treatments of two planting populations (2 and 4 seeds per foot), HD (Hilldrop), and single planted cotton seed. Two-row spacing of 36 inches and 40 inches were also used as treatment. Each treatment was 4 rows wide and 30 feet long. There were three repetitions per treatment. . . . .	8
2.2	Front view of the rover with the robotic arm, vacuum, and sensors mounted on the rover (see [1] for details) that was used to collect video streams of cotton plants in Tifton, GA. . . . .	9
2.3	Example of cotton field dataset image after image extraction from original bag files. . . . .	9
2.4	Example of cotton field pipeline input image after preprocessing prior to data ingestion into the pipeline. . . . .	10
2.5	Illustration of the proposed pipeline utilizing Azure resources . . . . .	11
2.6	Screenshot of the parameterization process for the stream and batch triggers to automate the pipeline for continuity . . . . .	13
2.7	Screenshot of Azure Data factory when setting up the Databricks linked service connection to ADF. The credentials required are as follows: Databricks workspace URL, Authentication type, Access token. Initially, we decided to create a new job cluster; however, based on the results, we shifted to existing interactive cluster; hence, we input the existing cluster ID. . . . .	16
2.8	The figure illustrates the tasks within Databricks notebook: compression (pre-processing), connection to AI Model, and creation of output with results (post-processing) . . . . .	18
2.9	Example of cotton bloom bounding box annotations for one cotton field sliced image. . . . .	19
2.10	Table displays results from tuning hyperparameters. The F1 score and mAP was the highest when utilizing the large YOLOv5 model with a threshold 0.55. We also tuned the number of epochs, but AutoML would terminate after 30 epochs due to no significant improvement. . . . .	20
2.11	Figure illustrates the definition of IOU which takes into account the area of overlap and the area of union. The higher the area of overlap between the detected bounding box and the ground truth box, the higher the IOU. . . . .	21
2.12	Example of pipeline output after post-processing and adding bounding boxes for cotton bloom detection visualization. . . . .	22



3.1	Aerial view of our cotton farm in Tifton, GA displaying 40 rows of cotton plants, treatments of two planting populations (2 and 4 seed per foot), HD (Hilldrop), and single planted cotton seed. Two row spacing of 35 inches and 40 inches were also used as treatment. Each treatment was 4 rows wide and 30 feet long. There were three repetitions per treatment. . . . .	29
3.2	Front view of the rover with the robotic arm, vacuum, and sensors mounted on the rover (see [1] for details) that was used to collect video streams of cotton plants in Tifton, GA.	30
3.3	An illustrative (original) image from the cotton blooms dataset displaying 3 cotton bloom flowering from August 6, 2021 prior to implementing bitwise masking. . . . .	31
3.4	Manipulated image from August 6, 2021 after implementing bitwise masking. . . . .	31
3.5	An illustrative image from August 6, 2021 with the contour highlighting all pixels within the white detection range. . . . .	32
3.6	An illustrative image from August 6, 2021 with the contouring of white pixels with the area less than 150 pixels. . . . .	32
3.7	An illustrative image of 4 out of 11 slices from the full frame image in Figure 3.3. . . . .	33
3.8	The result of applying bitwise masking to the slices from Figure 3.7. . . . .	33
3.9	Explained variance screen plots for each color channel for a given image. . . . .	34
3.10	The original slice 7 of the cotton bloom dataset on August 6. . . . .	36
3.11	The reconstructed image using the original slice from Figure 3.10 after being transformed through PCA and inverse-transformed displaying 3 cotton blooms. . . . .	36
3.12	The masked image (23.74 kB) using an original slice of the cotton bloom dataset on August 6 from Figure 3.10. . . . .	37
3.13	The masked image (23.29 kB) using the reconstructed slice 7 of the cotton bloom dataset on August 6 from Figure 3.11. . . . .	37
3.14	Cotton flower detection using bitwise masking on reconstructed images (reduced by more than 93% from the original byte size) has a true positive rate of 0.8913. Cotton flowering detection using bitwise masking on the original images leads to a true positive rate of 0.9022. . . . .	38
4.1	A graphical depiction of the Downsampling Algorithm. . . . .	41
4.2	An example of cotton field image data extracted from the ZED2 and ZEDM camera for 2022 cotton field data. . . . .	43
4.3	The positioning of ZED2 and ZEDM camera for 2022 cotton field data. . . . .	44
4.4	An example of the extracted bagfile frame of 2022 Cotton Field data with the ZED2 camera.	45
4.5	A example of the extracted bagfile frame of 2022 Cotton Field data with the ZEDM camera.	45
4.6	A display of a pre-processed sliced 2021 image data. . . . .	46
4.7	An illustration of the results of the YOLOv5 model on detecting cotton blooms on 2022 Cotton Field data with the ZED2 camera integrating both right and left lens from August 26, 2022. . . . .	47

4.8	A display of the right frame of 2022 Cotton Field data with the ZED2 camera from August 26, 2022. . . . .	47
4.9	A visual of the cotton bloom detection results of the right frame of 2022 Cotton Field data with the ZED2 camera from August 26, 2022. . . . .	48
4.10	An example of the right frame of 2022 Cotton Field data with the ZEDM camera. . . .	48
4.11	A visualization of results of cotton bloom detection using the ZEDM camera on the right frame of the 2022 Cotton Field data are shown. The output displays only the bounding boxes. We added a red arrow added for visual clarity to indicate false negatives, and a red X was added to highlight false positives. . . . .	49
4.12	An illustration of the cotton bloom detection output from the YOLO model after the pipeline execution of a sliced 2021 downsampled ZED2 image. The output displays only the bounding boxes. We added a red arrow added for visual clarity to indicate false negatives	50
4.13	A visual of the output of YOLOv5 model on 4.8 image data after downsampling. . . .	50
4.14	An example of pipeline output on 2022 Data with Masking and Downsampling, Preserving 50 Non-White Regions around Binary Classified White Region. . . . .	51
4.15	An example of pipeline output on 2022 Data with Masking and Downsampling, Preserving 50 Non-White Regions around Binary Classified White Region. . . . .	52

# LIST OF TABLES

3.1	Using the nbytes command in Python, the counts for the number of bytes in a directory containing $n$ sliced cotton data images, and a directory containing the numpy arrays representing the images in the compressed form of their top 100 principal components.	35
3.2	The total number of bytes in a directory containing $n$ original cotton field images compared with a directory containing $n$ reconstructed images, after PCA compression. . . .	35

# CHAPTER I

## INTRODUCTION AND MOTIVATION

As the global population continues to grow, meeting the increasing demand for food and fiber production presents significant challenges for farmers. Cotton, being one of the world’s most important fiber crops, plays a crucial role in meeting this demand. However, cotton crop productivity can be significantly impacted by pests, diseases, and environmental conditions. Therefore, effective crop monitoring techniques are needed to detect and address these factors in a timely manner. Phenotyping, the process of measuring and analyzing plant characteristics, plays a crucial role in providing real-time information about the physiological state of plants, enabling timely interventions to optimize crop growth and yield. This becomes particularly valuable in large-scale farming operations where manual monitoring can be labor-intensive and time-consuming.

Recent studies have demonstrated the potential of powerful machine learning tools, such as deep learning methods, in uncovering phenotypic-genotypic associations. Computer vision, pattern recognition technologies, data mining methods, and machine learning algorithms have been employed to analyze image-based non-invasive phenotype data in the past [2]. In particular, [3] provides a comprehensive summary of studies that have utilized deep convolutional neural networks (CNNs) for plant phenotyping, covering various applications such as plant stress evaluation, plant development, and postharvest quality assessment. These studies are organized based on technical advancements in imaging classification, object detection, and image segmentation. Furthermore, potential future research directions in leveraging CNN architecture for plant phenotyping purposes are discussed, showcasing the promising role of machine learning in advancing the field of plant phenotyping.

However, it is important to note that deep learning algorithms demand substantial data and computational resources, and the phenotyping process can generate large volumes of data, posing challenges in data management and analysis. Addressing these issues, [4] proposes a Smart Farming-Oriented Big Data Architecture (SFOBA) that addresses various aspects of smart farm data management, including technical constraints, data quality, modeling, and machine learning. To fully harness the potential of deep learning algorithms in high-throughput phenotyping (HTP) applications, a framework or platform that can effectively handle the challenges of data and computational resources is crucial. [5] proposes a fine-tuned Kappa architecture for behavior classification in Precision Livestock, building on the comparative

performance analysis of the Lambda and Kappa architectures presented by Santa [6] for real-time Big Data analytics. This underscores the need for a robust big data pipeline that can automate the phenotyping process and effectively manage the generated data.

We present an exploration of the application of artificial intelligence in the field of cotton phenotyping, which is the process of assessing and measuring the physical characteristics of cotton plants. With the increasing demand for cotton and the need for more efficient and effective cotton cultivation practices, cotton phenotyping has become an essential aspect of agricultural research. Chapter 1 serves as an introduction, outlining the objectives of the study. Chapter 2 presents the key technical aspects of the research, including the development and deployment of a Big Data Pipeline for field-based high-throughput cotton phenotyping data. Chapter 3 expands on the concept of Dimensionality Reduction of High-throughput Phenotyping Data in Cotton Fields, and focuses on the methods used to reduce the complexity of the high-throughput phenotyping data. In Chapter 4, we address the issue of scaling the pipeline for a more recent dataset collected from the field, and the potential benefits of data compression through downsampling and masking. Lastly, we discuss our findings and conclusions.

## CHAPTER 2

# DEVELOPMENT AND DEPLOYMENT OF A BIG DATA PIPELINE FOR FIELD-BASED HIGH-THROUGHPUT COTTON PHENOTYPING DATA

1

---

<sup>1</sup>Issac, Amanda. Submitted to *Smart Agricultural Technology*, March 27, 2023.

## Abstract

In this study, we propose a big data pipeline for cotton bloom detection using a Lambda architecture, which enables real-time and batch processing of data. Our proposed approach leverages Microsoft Azure resources such as Data Factory, Event Grids, Rest APIs, and Databricks. This work is the first to develop and demonstrate the implementation of such a pipeline for plant phenotyping through Azure’s cloud computing service. The proposed pipeline consists of data preprocessing, object detection using a YOLOv5 neural network model trained through Azure AutoML, and visualization of object detection bounding boxes on output images. The trained model achieves a mean Average Precision (mAP) score of 0.96, demonstrating its high performance for cotton bloom classification. We evaluate our Lambda architecture pipeline using 9,000 images yielding an optimized runtime of 34 minutes. The results illustrate the scalability of the proposed pipeline as a solution for deep learning object detection, with the potential for further expansion through additional Azure processing cores. This work advances the scientific research field by providing a new method for cotton bloom detection on a large dataset and demonstrates the potential of utilizing cloud computing resources, specifically Azure, for efficient and accurate big data processing in precision agriculture.

High-throughput Cotton Phenotyping; Big Data Pipeline; Lambda Architecture; Computer Vision; Deep Neural Networks

## 2.1 Introduction

The demand for sustainable agriculture has put significant pressure on the agriculture sector due to the rapid growth of the global population. Precision farming techniques enabled by Computer Vision (CV) and Machine Learning (ML) have emerged as promising solutions where crop health, soil properties, and yield can be monitored and lead to efficient decision-making for agriculture sustainability. Data would be gathered through heterogeneous sensors and devices across the field like moisture sensors and cameras on the rovers. However, the huge number of objects in farms connected to the Internet leads to the production of an immense volume of unstructured and structured data that must be stored, processed, and made available in a continuous and easy-to-analyze manner [7]. Such acquired data possesses the characteristics of high volume, value, variety, velocity, and veracity, which are all characteristics of big data. In order to leverage the data for informed decisions, a big data pipeline would be needed.

One area of agriculture that faces particular challenges with regard to yield prediction is cotton production. The operation of cotton production is faced with numerous challenges, a major one being the timely harvesting of high-quality cotton fiber. The lack of skilled labor and external factors such as climate change, decreasing arable land, and shrinking water resources hinder sustainable agricultural production [8]. Moreover, delayed harvesting can lead to the degradation of cotton fiber quality due to exposure to unfavorable environmental conditions. Therefore, to avoid degradation, it is vital for harvesting cotton when at least 60% to 75% are fully opened, but also prior to the 50-day benchmark when bolls begin to degrade in quality [9]. In addition, cotton harvesting is costly, as the machines used for their processing can weigh over 33 tons and can also cause soil compaction, hence reducing land productivity [10]. In this

context, heterogeneous and large-volume data is collected using various static and moving sensors. Therefore, it is imperative to develop a platform that can handle real-time streams and manage large datasets for High-Throughput Phenotyping (HTP) applications. However, most conventional storage frameworks adopted in previous studies support only batch query processing and on-premise servers for data processing. Rather than implementing on-premise processing, the adoption of cloud computing can help prevent over- or under-provisioning of computing resources, reducing costly waste in infrastructure for farmers as shown in [11] who introduced a cost-optimized architecture for data processing through AWS cloud computing resources. Therefore, leveraging cloud computing could be a viable option for developing an efficient and scalable platform for HTP applications.

In this paper, we aim to implement batch and real-time processing using cloud computing which can help prevent over- or under-provisioning of computing resources. For that, we propose a big data pipeline with a Lambda architecture through Azure which allows for the cohesive existence of both batch and real-time data processing at a large scale. This two-layer architecture allows for flexible scaling, automated high availability, and agility as it reacts in real time to changing needs and market scenarios. For testing this pipeline, we train and integrate a YOLOv5 model to detect cotton blooms using the gathered dataset.

### **2.1.1 Lambda Architecture**

Lambda architecture, first proposed in [12], is a data processing architecture that addresses the problem of handling both batch and real-time data processing by using a combination of a batch layer and a speed layer. In the context of agriculture, various research studies have implemented Lambda architecture pipelines to process and analyze large amounts of sensor data, such as weather data and crop yields, in order to improve crop forecasting and precision agriculture. Very recent work [13; 14] have demonstrated the feasibility of using a Lambda architecture framework in smart farming.

### **2.1.2 Cloud Computing**

Previous research on big data pipelines has employed on-premise servers for data processing, while the use of cloud computing can substantially reduce the cost for farmers. Cloud providers, such as Microsoft Azure, offer various data centers to ensure availability and provide better security compared to on-premise servers. We propose the adoption of Microsoft Azure Big Data resources to implement a Lambda architecture pipeline in the agriculture industry. Azure Big Data Pipeline is a cloud-based processing service offered by Microsoft that can be utilized for analyzing, processing, and implementing predictive analysis and machine learning-based decisions for agricultural operations. By integrating Azure Big Data resources, farmers can substantially increase the efficiency of data processing, while reducing costs to further enhance agricultural productivity.



## **Azure Data Factory**

Azure Data Factory (ADF) allows for the creation of end-to-end complex ETL data workflows while ensuring security requirements. This environment enables the creation and scheduling of data-driven workspace and the ingestion of data from various data stores. It can integrate additional computing services such as HDInsight, Hadoop, Spark, and Azure Machine Learning. ADF is a serverless service, meaning that billing is based on the duration of data movement and the number of activities executed. The service allows for cloud-scale processing, enabling the addition of nodes to handle data in parallel at scales ranging from terabytes to petabytes. Moreover, one common challenge with cloud applications is the need for secure authentication. ADF addresses this issue by supporting Azure Key Vault, a service that stores security credentials [15]. Overall, the use of ADF in our pipeline allows for efficient and secure data processing at scale.

### **2.1.3 Related Work**

Previous studies have utilized traditional pixel-based CV methods, such as OpenCV, to identify cotton bolls based on their white pixel coloring [16]. Another study has explored the use of YOLOv4 in order to detect cotton blooms and bolls [17]. Moreover, the integration of big data architecture has been suggested in previous research to optimize agricultural operations [18]. Parallel studies have explored the use of Lambda architecture pipelines as a viable approach to process and analyze large amounts of sensor data, such as weather data and crop yield, in order to improve forecasting for specific crops. For instance, Roukh presents a cloud-based solution, named WALLeSMART, aimed at mitigating the big data challenges facing smart farming operations [13]. The proposed system employs a server-based Lambda architecture on the data collected from 30 dairy farms and 45 weather stations. Similarly, Quafiq integrates a big data pipeline inspired by Lambda architecture for smart farming for the purposes of predicting drought status, crop distributions, and machine breakdowns [14]. The study suggests the benefits of flexibility and agility when utilizing a big data architecture. Furthermore, cloud-based solutions have become increasingly popular in agriculture due to their scalability and cost-effectiveness. Another study employs big data in the cloud related to weather (climate) and yield data [19].

### **2.1.4 Summary of Contributions and Organization of the Paper**

This paper focuses on the use of Microsoft Azure resources to implement and validate a Lambda architecture High-throughput Phenotyping Big Data pipeline for real-time and batch cotton bloom detection, counting, and visualization. We develop data reduction and processing to transfer useful data and separately train a YOLOv5 object detection model and integrate it into our big data pipeline. The pipeline was thoroughly tested and demonstrated through the analysis of a set of 9000 images.

New, noteworthy, and useful: Despite existing research work on the use of Lambda architecture and its benefits, there is still a lack of studies that elaborate on the development process and tools to construct this architecture. Moreover, there has been limited research on the application of Lambda architecture

utilizing cloud computing resources, as most are server based. **To the best of our knowledge, there is no previous study that elaborates on the implementation of a big data Lambda architecture pipeline utilizing cloud computing resources, specifically Azure, while integrating advanced machine learning models for plant phenotyping applications.** While big data analytics and cloud computing have become increasingly popular in precision agriculture, the integration of these technologies with Lambda architecture for plant phenotyping (our case, cotton) remains an open research area. Our approach demonstrates the efficacy of utilizing cloud-based resources for the efficient and accurate analysis of large-scale agricultural datasets.

This paper makes several contributions to the research field, which are listed as follows:

1. Introducing a Lambda architecture pipeline that takes into account batch and real-time processing, providing an efficient and scalable solution for data analysis.
2. Utilizing cloud computing resources, specifically Microsoft Azure, to improve the performance and reliability of the proposed pipeline.
3. Demonstrating the actual implementation tools and processes used to build the proposed pipeline, enabling other researchers to replicate and build upon our work.
4. Integrating a big data pipeline for cotton plant phenotyping, which enables the efficient analysis of large volumes of data and provides new insights into the growth and development of cotton plants.
5. Contributing a new cotton field dataset to the research community, which is currently limited, enabling other researchers to validate and build upon our findings.

The remainder of the paper is organized as follows: Section 2 describes data retrieval; Section 3 summarizes the implemented Lambda architecture pipeline; Section 4 provides a summary of offline AI-based object detection model training; Section 5 discusses fine-tuning methods to optimize the continuous pipeline run-time and showcases final results, and lastly in Section 6, we discuss areas for future work and concluding remarks.

## **2.2 Dataset**

In this study, we employed our own cotton field dataset to evaluate the proposed pipeline for phenotyping analysis. The cotton field dataset will be further elaborated in subsequent sections, including data collection procedures and data preprocessing steps.

### **2.2.1 Cotton Research Farm**

The cotton data was collected using a stereo camera that was installed on an autonomous ground vehicle deployed in a research farm at the University of Georgia’s Tifton campus in Tifton, GA. Figure 3.1 illustrates an aerial view of the farm. The treatments described in Figure 3.1 are 4-row wide, but we collected data on the inner 2-rows for post-analysis as discussed later.

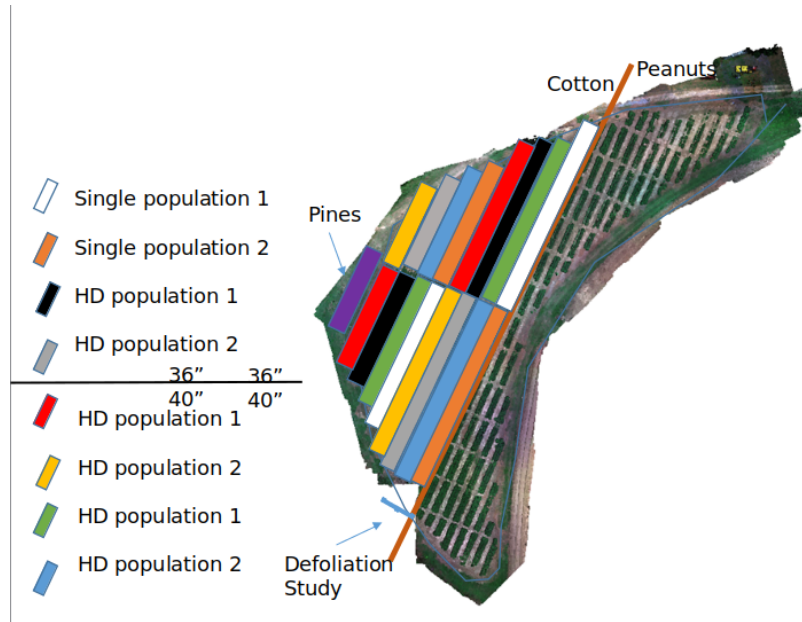


Figure 2.1: Aerial view of our cotton farm in Tifton, GA displaying 40 rows of cotton plants, treatments of two planting populations (2 and 4 seeds per foot), HD (Hilldrop), and single planted cotton seed. Two-row spacing of 36 inches and 40 inches were also used as treatment. Each treatment was 4 rows wide and 30 feet long. There were three repetitions per treatment.

### 2.2.2 Cotton Field Data Collection

In our data collection efforts, we employed a rover developed by West Texas Lee Corp. (Lubbock, Texas). As described in [1], this rover is a four-wheel hydrostatic machine with a length of 340 cm, front and rear axles 91 cm from the center, and a ground clearance of 91 cm. It was powered by a Kohler Command 20HP gasoline engine driving an axial-Piston variable rate pump and equipped with the Nvidia Jetson Xavier for remote control and vision and navigation systems. With a top speed of approximately 2 kilometers per hour, the rover was able to efficiently traverse the study area. To power electronics and ZED camera, the rover utilized two 12-Volt car batteries.

The ZED stereo camera, with left and right sensors 120 cm apart and mounted 220 cm above the ground facing downward [1] was chosen for its ability to perform effectively in outdoor environments and provide real-time depth data. It captured 4-5 frames per second and recorded a video stream of each 4-row treatment from June 2021 to October 2021, 2-3 days per week, as a ROS bag file.

### 2.2.3 Dataset Creation

In this study, a camera equipped with two lenses was utilized to capture images of cotton plants. The camera captured both left and right views of the plants, with a total of 765 image frames extracted from

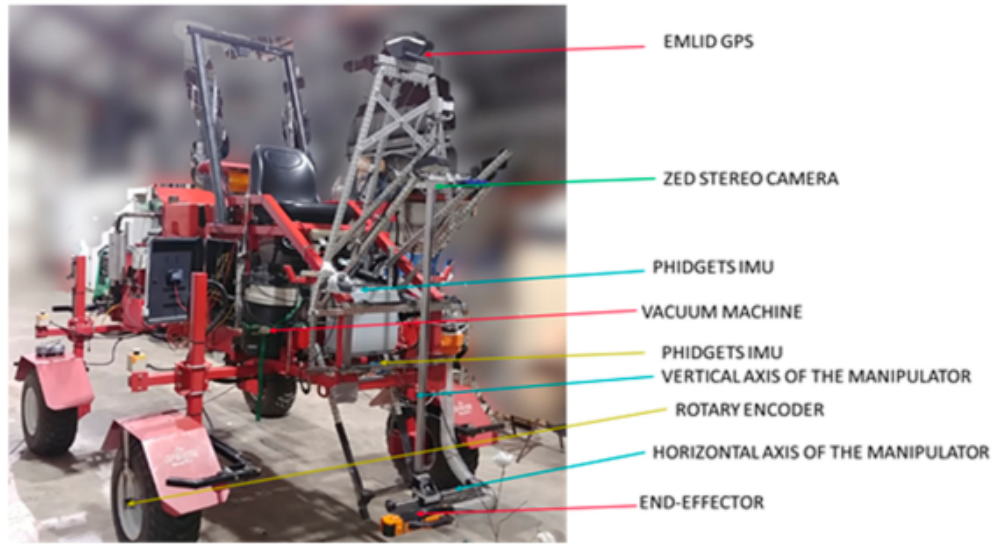


Figure 2.2: Front view of the rover with the robotic arm, vacuum, and sensors mounted on the rover (see [1] for details) that was used to collect video streams of cotton plants in Tifton, GA.

sixteen 4-row treatments on each data collection day between July 14, 2021 and August 6, 2021, when blooms began to appear. These frames were labeled in ascending numerical order to ensure proper correspondence with the video stream and prevent any overlapping. The 765 image frames were subsequently divided into separate sets for the left and right lens views, resulting in a total of 1,530 frames. An example of the image frames captured by the left and right lens can be seen in Figure 2.3.



Figure 2.3: Example of cotton field dataset image after image extraction from original bag files.

Previous research has shown that the small proportion of blooms relative to the background in cotton field images can make it difficult for neural network models to accurately detect the blooms [17]. To address this issue, we pre-processed the images by dividing them into five equal slices. The treatments described in Figure 3.1 are 4-row wide, but we collected data on the inner 2-rows for analysis when slicing. An example of the resulting images is shown in Figure 2.4 used as input for the subsequent analysis pipeline.

We selected a dataset consisting of sliced images from 10 specific days in 2021: July 8, July 14, July 16, July 19, July 23, July 26, July 29, August 4, August 6, and September 9. This resulted in a total of 9,018 images with 3 color channels (RGB) with dimensions of  $530 \times 144$  for testing batch processing and creating the offline object detection model. The dataset in this study comprised diverse cotton plant data, locations, and treatments, as the video streams were collected from various rows on different days.



Figure 2.4: Example of cotton field pipeline input image after preprocessing prior to data ingestion into the pipeline.

### 2.3 Development of Lambda Architecture Pipeline

In this work, we propose a Lambda architecture to enable real-time analytics through a distributed storage framework, which traditionally is only capable of batch processing. The proposed architecture consists of three main layers: batch, speed, and serving. The batch layer is responsible for processing large amounts of historical data on a schedule, while the speed layer handles real-time streams of data. The serving layer serves the processed data to clients, applications, or users. This approach allows for the efficient handling of both historical and real-time data, enabling a wide range of analytical capabilities. We illustrate the

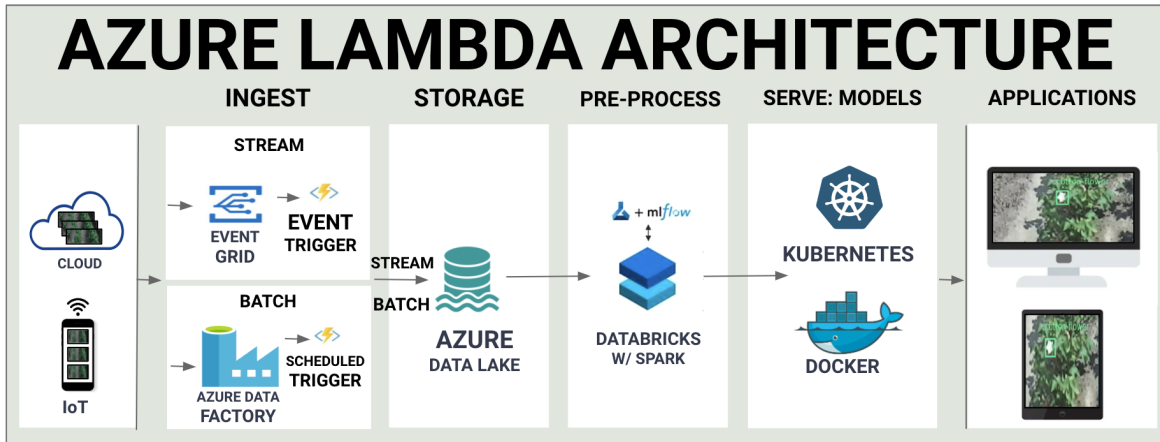


Figure 2.5: Illustration of the proposed pipeline utilizing Azure resources

Lambda architecture using Azure resources in Figure 2.5. In order to achieve real-time ingestion, we utilize the Azure Data Factory’s event-based trigger which sends an event when an image is uploaded to the storage account. This event is handled by Azure’s Event Grid for real-time streams. In comparison, batch ingestion is triggered by a scheduled event. Once ingested into the Azure Data Factory, the pipeline connects to Databricks for preprocessing of the image data. The processed data is then forwarded to a deployed AI object detection model, which is running on a Kubernetes cluster, to retrieve the designated bounding box coordinates for the image. Finally, Databricks draws the bounding boxes and outputs the image. The development process will further be elaborated.

To initiate our analysis, we established an Azure Data Factory workspace. The Azure Data Factory portal allows monitoring the pipelines’ status in real time. In order to use the Data Factory, we had to create a resource group, a container for holding related resources for our Azure solution. For this work, we opted to ingest binary unstructured data from Azure Blob storage into Azure Data Lake. This allowed us to efficiently process and store large volumes of data for subsequent analysis.

Azure Blob storage is a highly scalable unstructured data object storage service. To use Blob storage and create an Azure Data Lake, we first had to initialize a storage account. Azure Storage is a Microsoft-managed service that provides cloud storage and provides REST API for CRUD operations. For this project, we configured the storage account to use locally redundant storage (LRS) for data replication, as it is the least expensive option. We also set the blob access tier to ‘hot’ to optimize for frequently accessed and updated data. The storage account’s data protection, advanced, and tags settings were left as their default values. Overall, the use of Azure Blob storage and the creation of an Azure Data Lake allowed us to efficiently store and process large volumes of unstructured data for our analysis.

Microsoft Azure Data Lake is a highly scalable data store for unstructured, semi-structured, and structured data [15]. It is compatible with Azure services and a variety of additional tools, making it

capable of performing data transformation and handling large volumes of data for analytics tasks. To separate the stream and batch processing in our pipeline, we created two separate blob containers labeled batch and stream. Files ingested into the 'batch' folder are processed by a scheduled trigger designed for batch processing, while files ingested into the 'stream' folder trigger real-time processing. This allows us to efficiently handle both historical and real-time data in our analysis.

### **2.3.1 Speed Layer**

The speed layer of the Lambda architecture is designed for real-time analysis of incoming data streams. It is generally not used for training machine learning models, but rather for applying pre-trained models to classify or predict outcomes for the incoming data. This allows to provide real-time insights which are crucial when timely action is required depending on the data. For example, real-time analysis of cotton bloom location and density can enable farmers to take immediate action. Another benefit of the stream layer is its ability to handle high-volume data streams with low latency, which can be a challenge for traditional batch processing systems that may suffer from delays in the availability of insights. Furthermore, this would be best if combined with previous bloom data to create a time series of bloom count and location, that provides a way to forecast yield and/or plant health.

#### **Ingestion**

To enable real-time processing in our pipeline, we implemented a file storage trigger in the stream layer. This trigger initiates the pipeline in real time whenever a new image is added to the blob storage. This approach allows us to automate the data processing and analysis pipelines, hence reducing the need for manual intervention. Additionally, the file storage trigger is compatible with other services such as Azure IoT Hub, enabling us to process data ingested from IoT devices for scalability. This approach allows to efficiently and effectively analyze data as it is generated in near real time.

The creation of a real-time trigger in Azure Data Factory also generates an event grid in Azure. Event Grid is a messaging service in Azure that enables the creation of event-driven architectures. It can be used to trigger actions such as running a pipeline. In our case, the event grid listens for events in the input source (blob storage) and, upon detecting a new event, sends a message to the Data Factory service to trigger the execution of the pipeline. This allows for the automation of the pipeline process. For the transfer of data from blob storage into the data lake, we must create a connection between the Data Factory and the Data Lake. We used a Copy Activity in a Data Factory pipeline to copy data from a Data Lake store to a different store or data sink.

In our pipeline, we use two separate folders as input sources, each with its own trigger (batch and stream). To facilitate this configuration, we parameterized the input file name to accommodate the separate cases of the stream and batch layers. By adopting the parameterization of the data folder input as dynamic, we were able to alter the folder used as the input source without modifying the pipeline itself. This approach allows us to flexibly configure the input sources for our pipeline without the need for additional maintenance or modification.

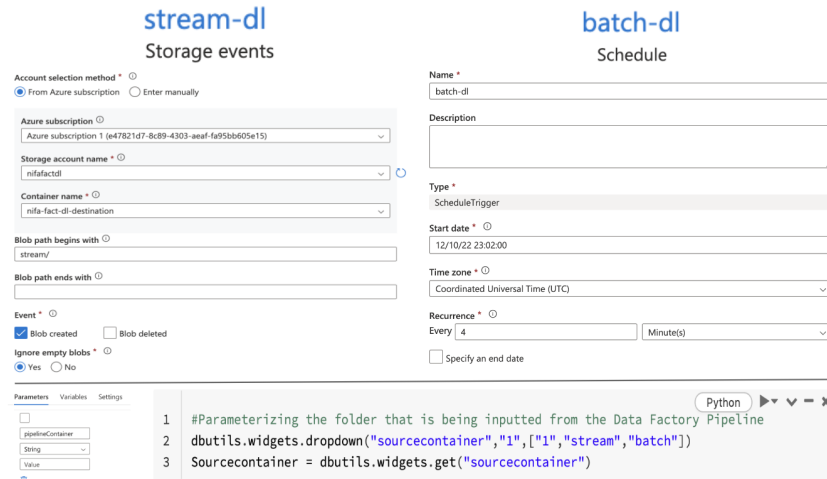


Figure 2.6: Screenshot of the parameterization process for the stream and batch triggers to automate the pipeline for continuity

### 2.3.2 Batch Layer

The batch layer of our pipeline serves as the primary repository for the master dataset and allows us to view a batch of the data prior to computation. The layer plays a crucial role in managing and organizing the dataset, enabling efficient analysis and processing. We can divide this batch data into smaller batches to train machine learning models on large datasets quicker, independently, parallel, and through fewer computational resources. It also helps with the scalability of a machine learning system as the system will be able to handle larger datasets, optimize the training process, and improve the performance of the resulting model.

### Ingestion

For our experiments, we selected a dataset consisting of sliced images from 10 specific days in 2021, which resulted in a total of over 9,000 images. The dataset is stored in Azure Blob Storage, a scalable cloud-based object storage service that is capable of storing and serving large amounts of data. This scalability, compatible with terabytes of data, makes it well-suited for use in data-intensive applications such as ours. To accommodate larger volumes of data, Blob Storage is engineered to scale horizontally by automatically distributing data across multiple storage nodes. This allows it to handle increases in data volume and access requests while eliminating additional manual provisioning or configuration.

In our pipeline, we integrated a batch trigger in addition to the stream layer trigger. This trigger is of the batch type, allowing us to specify a predetermined schedule for execution. The schedule can be fixed, such as running every day at a specific time, or dynamic through a CRON expression, which is a job scheduler used within Azure. For the purposes of our experimentation, the trigger is calibrated to run



every 3 minutes. However, the flexibility of the batch trigger schedule allows for the customization of the frequency of execution to meet our specific data collection and processing needs. For example, the trigger can be executed on a weekly or hourly basis when collecting data on-site. The use of a batch trigger in Azure Data Factory allows us to scalably process large volumes of data. We can ingest data into the batch layer at a rate that meets our specific needs, and schedule the trigger to execute at appropriate intervals to ensure that the data is processed and analyzed in a timely manner. The ability to adjust the schedule of the batch trigger allows us to fine-tune the performance of our pipeline and ensure that it is able to handle the volume and velocity of our data effectively.

### **Azure Data Factory Connection**

The batch layer follows the same process as the speed layer for the Azure Data Factory connection. If an image is ingested into the batch folder, the batch trigger sends the parameter of the batch which will be used in the remainder of the pipeline for organizing data.

### **2.3.3 Pre-process/Analyze**

In the analysis of high-volume data, pre-processing is a vital step. Raw data from devices may contain inconsistencies and noise which can depreciate the quality of results and decision-making insights. These issues are addressed through the cleansing, normalization, and reduction of data. Furthermore, the pre-processing step of images integrates various techniques such as noise reduction, image enhancement, and feature extraction. These methods assist with streamlining decision-making and interpretation. We decided to incorporate image compression into our pipeline as it can significantly reduce the size of images. This downsizes storage size and costs of processing large volumes of data. By integrating image compression methods to eliminate image data redundancy, it is possible to represent an image through fewer bits, resulting in a smaller file size. However, there are trade-offs in terms of image quality and compression ratios, thus it is imperative to select an image compression algorithm that does not deteriorate the quality when compressing. These steps are crucial in the context of big data pipelines, where storage space is often a limiting factor.

### **Databricks Connection to Data Lake**

To facilitate pre-processing, we incorporate Databricks, a cloud-based platform that integrates Apache Spark, a powerful open-source data processing engine [20]. Apache Spark is optimized to handle substantial amounts of data quickly and efficiently, making it ideal for real-time data processing applications. It boasts the capability for in-memory processing, rendering it significantly more efficient than disk-based systems, especially when working with vast amounts of data, resulting in reduced computation time. Moreover, Apache Spark supports parallel processing, permitting it to divide data into smaller chunks and process them simultaneously to enhance performance even further if needed.

In the realm of pre-processing tasks, a popular alternative to Databricks is the open-source big data processing framework, Hadoop. Hadoop utilizes the MapReduce programming model, which has been



shown to be challenging to work with in comparison to the Spark engine utilized by Databricks [21]. Furthermore, Hadoop requires significant configuration and maintenance efforts to set up and run properly, whereas Databricks offers a user-friendly interface and requires less infrastructure [20]. In addition, Databricks provides a range of additional tools and features, such as integration with data storage platforms like Amazon S3 and Azure Blob Storage, as well as the ability for data scientists and analysts to collaborate through notebooks and dashboards [22], making it a more convenient platform for handling big data.

For our experiments, we configured our Databricks cluster to use Databricks Runtime version 11.3 LTS. The worker and driver type is Standard DS3 v2 which contains 14 GB memory and 4 cores. The range of workers was set to be between 2 to 8 with auto-scaling enabled, where the cluster configures the appropriate number of workers based on the load. Once the data is ingested into the Data Lake, we decide to compress the image by storing the image into a jpeg file with 30% quality. This pre-processing stage is flexible and scalable where we can also implement other pre-processing and data transformation techniques such as image slicing. Furthermore, we checked the image dimensions to be a valid input for our model.

The next step is to configure the Databricks linked service connection. The Databricks linked service connection in Azure is a way to connect to a Databricks workspace from Azure. It allows users to easily access and integrate data stored in their Databricks workspace with Azure Data Factory. When configuring the Databricks linked service, we enter the Databricks workspace URL and authentication access token. We first selected the method of having a new job cluster created anytime there was an ingestion trigger.

In order to improve the efficiency of the data processing pipeline, we decided to switch from creating a new job cluster for each ingestion trigger to using existing interactive clusters. This approach reduces the time required for the pipeline to start processing, as the interactive cluster is already active when new data is ingested. This process saves the average 3 minutes of restarting a new cluster for every single trigger. However, when a file is first uploaded, there exists a delay while the inactive interactive cluster is first started. To minimize this delay, we calibrate the interactive cluster to terminate if no activity has been detected for a period of 20 minutes. This configuration can be easily adjusted to meet the needs of different use cases. This results in the first image ingestion taking 3 minutes to begin the cluster, however, subsequent image ingestions demonstrated a significant reduction in connection time to the cluster, with a duration of fewer than 10 seconds.

To enable Databricks to access the Azure Data Factory, we mounted the Data Lake Storage Gen2 (ADLS Gen2) file system to the Databricks workspace. This allows us to use standard file system operations to read and write files in the ADLS Gen2 file system as if it were a local file system. Mounting the ADLS Gen2 file system to Databricks enables us to access data stored in ADLS Gen2 from Databricks notebooks and jobs, and facilitates integration between Databricks and other tools and systems that use ADLS Gen2 as a storage backend. Furthermore, the parameterization of the input folder (batch vs. stream folder) allows the databricks notebook to use this Dynamic input to make changes to the correct data lake folder.

**Edit linked service**  
 Azure Databricks [Learn more](#) 

Connect via integration runtime \* ⓘ  
 AutoResolveIntegrationRuntime ▼

Account selection method \*  
 From Azure subscription  Enter manually

Databrick Workspace URL \* ⓘ  
 https://adb-5415516892690559.19.azuredatabricks.net

Authentication type \*  
 Access Token ▼

Access token  Azure Key Vault

Access token \* ⓘ  
 .....

Select cluster  
 New job cluster  Existing interactive cluster  Existing instance pool

Existing cluster ID \* ⓘ  
 1122-193842-wjg6axr4 ▼

Figure 2.7: Screenshot of Azure Data factory when setting up the Databricks linked service connection to ADF. The credentials required are as follows: Databricks workspace URL, Authentication type, Access token. Initially, we decided to create a new job cluster; however, based on the results, we shifted to existing interactive cluster; hence, we input the existing cluster ID.

### 2.3.4 AI Model/APIs

In this section, we describe the process of deploying the trained AI model.

#### Deployment with Kubernetes

To deploy a trained Object Detection model in the pipeline, we utilized Azure Kubernetes Service (AKS) [23]. Microsoft Azure’s AKS simplifies the process of deploying and scaling containerized applications on the cloud platform through its managed Kubernetes service. By leveraging the benefits of the open-source Kubernetes container orchestration platform, AKS creates a consistent and predictable environment for managing these applications. With features like automatic bin packing, load balancing, and secret and configuration management, AKS enhances the management of containerized applications. The service achieves this by creating and managing clusters of virtual machines that run these applications, making the deployment and scaling process easier and more efficient.

Kubernetes is highly scalable, and its platform allows for the management of applications across multiple nodes in a cluster, making it a versatile solution for managing containerized applications in the cloud [23]. It provides a consistent and predictable environment for deploying and scaling containerized applications. Secret and configuration management provides secure, encrypted storage for sensitive data such as passwords and API keys, improving application security. Automatic bin packing allows Kubernetes to schedule containers to run on the most appropriate nodes in a cluster, maximizing cost efficiency. Load balancing distributes incoming traffic across multiple replicas of an application to handle high volumes.

This cluster uses a Standard D3 v2 virtual machine which has 4 cores, 14 GB RAM, and 200GB storage. We retrieve the score Python script from the best AutoML YOLOv5 run, and use it to deploy the model as an AKS web service. In order to assess the performance of our machine learning model, we utilized a Python script. This script contains code for loading the trained model, reading in data, making predictions using the model, and calculating various performance metrics such as accuracy and precision. It also includes provisions for saving the predictions made by the model and the calculated performance metrics to a file or database for further analysis. By running our script on a separate dataset, known as the test dataset, we were able to obtain an unbiased estimate of the model's performance and assess its ability to generalize to new data.

To enhance the capability of our object detection model in handling elevated workloads, it was deployed with autoscaling enabled. This allows for dynamic adjustment of computing resources, such as CPU Processing Nodes and memory, in response to incoming requests. The initial configuration was set to 1 CPU core and 7 GB of memory. To secure the model, an authentication key system was implemented, requiring the provision of a unique key with each request. This key system ensures only authorized access to the model. Subsequent sections of this research will elaborate on the training process of the object detection model and its integration into the workflow.

### **Azure Data Factory Connection**

In order to optimize the efficiency of our pipeline, we made the decision to include the AKS connection credentials within the initial Databricks notebook where the data is pre-processed. This approach was chosen as an alternative to utilizing Azure's ADF web service option for REST API connection in the Azure Data Factory, which would have required the creation of another Databricks notebook to draw the bounding boxes from the output of bounding box coordinates. By integrating the AKS connection credentials directly into the primary notebook, we were able to streamline the process while eliminating the need for an additional Databricks compute cluster and cluster connection time. This avoided the added overhead of creating an additional notebook in ADF, which would have slowed down the pipeline. Overall, we have one Databricks notebook which will conduct the pre-processing and post-processing of data. Figure 2.8 illustrates the tasks of Databricks in the pipeline.

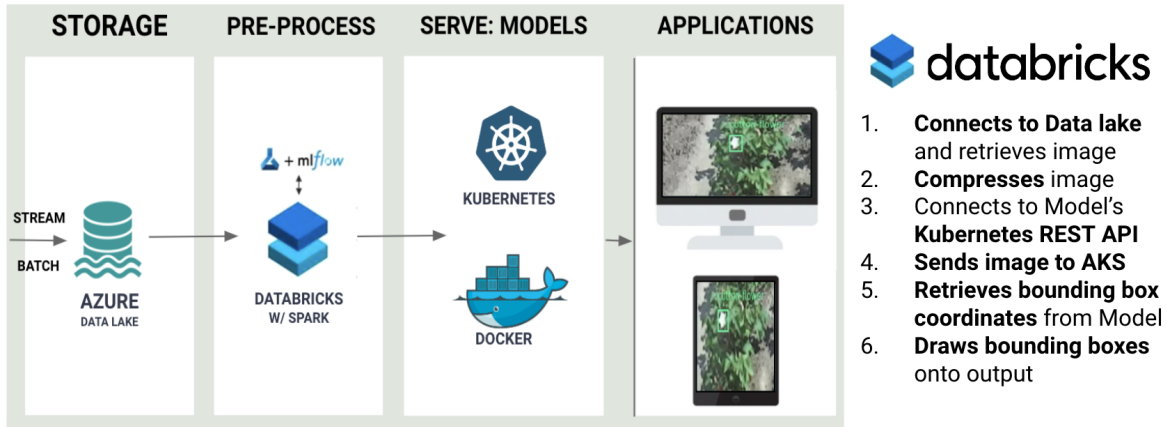


Figure 2.8: The figure illustrates the tasks within Databricks notebook: compression (pre-processing), connection to AI Model, and creation of output with results (post-processing)

### 2.3.5 Output

We developed an object detection model that is capable of identifying and counting cotton blooms in images. When the model is run on an input image, it returns the bounding box coordinates of any cotton blooms that it detects. To visualize the results of the model, we retrieve these bounding box coordinates and use them to create visual bounding boxes over the input image. This output image, which shows the detected cotton blooms overlaid on the original image, is then stored in a blob storage account. By using a blob storage REST API, we can easily send this output image to any other device for further processing or analysis. This approach allows us to scale the output of the model to meet needs.

## 2.4 Offline YOLOv5 Model Training

Object detection is a key task in computer vision, which involves identifying and locating objects of interest in images or video streams. One popular object detection model is YOLO (You Only Look Once), which was first introduced by Redmon in 2015 [24]. Since then, the YOLO model has undergone several revisions, and one key difference between YOLOv5 and its predecessor, YOLOv4, is the training process. While previous versions of YOLO, including YOLOv4, were trained using the Darknet framework, YOLOv5 utilizes the TensorFlow backend. This allows YOLOv5 to benefit from the advanced optimization and acceleration techniques provided by TensorFlow, which can improve the model's performance and speed. YOLOv5 also introduces several other improvements and new features compared to YOLOv4. These include more efficient network architecture and support for a wider range of input sizes [25].

### 2.4.1 Data Labeling

We utilized AutoML and Azure Machine Learning Studio to train a YOLOv5 model for cotton bloom detection. AutoML automates the process of selecting and training the most suitable machine learning model for a given dataset. It allows users to easily train, evaluate, and deploy machine learning models without the need for extensive programming knowledge or machine learning expertise [26]. To train the YOLOv5 model using AutoML, we first set up a connection between our data lake (which contained the images used for training) and Azure Machine Learning Studio. Azure Machine Learning Studio is a cloud-based platform that provides tools for developing, deploying, and managing machine learning models in Azure [27]. Once the connection was established, we were able to use AutoML and Azure Machine Learning Studio to train and evaluate the YOLOv5 model on our dataset. The platform provided a range of tools and resources for optimizing the model's performance, including the ability to tune hyperparameters, apply data augmentation techniques, and evaluate the model's performance using a variety of metrics which will be further discussed.

After creating the Machine Learning studio workspace, we need to create a Datastore which connects to our Data Lake Storage Container. From the Datastore, we created a Data Asset. Data stores and data assets are resources in Azure Machine Learning studio that allows us to store and access data for machine learning experiments. We created the Data Asset through 1300 images saved in a Data Lake that was compressed prior. In our case, we decided to reduce the quality of the images by compressing prior to training the model. This way, our model would provide better accuracy when implementing the full pipeline which compresses the images prior to being send into the AI Model. Using the Azure ML Studio Labeler tool, we annotated 1300 images through bounding boxes that can be used to identify the location and size of the cotton blooms in the image. The AutoML labeler tool is part of the Azure Machine Learning platform. After the annotations were complete, we exported them into AzureML Dataset format. Figure 2.9 is a screenshot of an example of annotating one image through Azure's Image Labeler tool.



Figure 2.9: Example of cotton bloom bounding box annotations for one cotton field sliced image.

## 2.4.2 Model Hyperparameters and Training

In this work, the model utilized 80 percent of the dataset for training, and 20 percent for validation purposes. Furthermore, the YOLOv5 model was trained using a learning rate of 0.01, a model size of large which contains 46.5 million training parameters, and a total of 70 epochs. However, the training process was terminated early when the mean average precision (mAP) metrics stopped improving. This resulted in the training process stopping early at 30 epochs in our experiment. The number of epochs used for training is important, as it determines the number of times that the model sees the training data and can influence the model's performance. Figure 2.10 shows results from our hyperparameter tunings.

Model Size	Threshold	Precision	Recall	F1	mAP
Medium	0.45	0.68	0.99	0.806	0.96
Medium	0.5	0.72	0.98	0.830	0.96
Medium	0.55	0.76	0.94	0.840	0.93
<b>Large</b>	<b>0.55</b>	<b>0.84</b>	<b>0.99</b>	<b>0.908</b>	<b>0.96</b>

Figure 2.10: Table displays results from tuning hyperparameters. The F1 score and mAP was the highest when utilizing the large YOLOv5 model with a threshold 0.55. We also tuned the number of epochs, but AutoML would terminate after 30 epochs due to no significant improvement.

One key aspect of the training process was the use of the Intersection over Union (IOU) threshold, which is a measure of the overlap between the predicted bounding boxes and the ground truth bounding boxes (see Figure 2.11). The IOU threshold was set to 0.55 for both precision and recall, which means that a predicted bounding box was considered correct if the overlap with the ground truth bounding box was greater than or equal to 0.55. The use of the IOU threshold is important because it allows the model to be evaluated using a standard metric to compare the performance of different models. In addition to the IOU threshold, the training process also involved setting the batch size to 10, where the model parameters were updated for each batch of 10 images. This training was performed using a computing cluster with 6 cores, 1 GPU, 56 GB of RAM, and 360 GB of disk space through Azure Machine Learning studio. The overall training process took 1 hour and 10 minutes to complete. [28].

## 2.5 Finetuning and Results

### 2.5.1 YOLOV5 Model

In this work, the trained YOLOv5 AutoML model achieved a mean average precision (mAP) score of 0.96. The mAP score is a metric that is commonly used to evaluate the performance of object detection models. It measures the average precision across all classes of objects in the dataset and takes into account the overall precision and recall of the model. Precision is a measure of the accuracy of the model's predictions and defined as the number of correct predictions divided by the total number of predictions. In comparison,

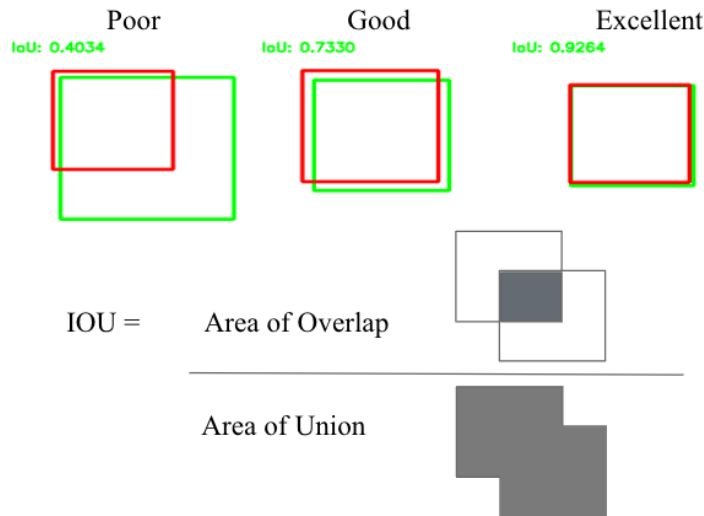


Figure 2.11: Figure illustrates the definition of IOU which takes into account the area of overlap and the area of union. The higher the area of overlap between the detected bounding box and the ground truth box, the higher the IOU.

recall calculates the model's ability to capture all relevant instances in its predictions. It can be determined by dividing the number of correct predictions by the total number of instances in the actual data [28].

In this case, the YOLOv5 model had a precision value of 0.84 and a recall score of 0.99 when using an IOU validation threshold of 0.55. The F1 score, which is a measure of the harmonic mean of precision and recall, was also calculated and found to be 0.904. The importance of precision, recall, and the F1 score lie in their ability to provide a comprehensive evaluation of the model's performance. High precision is essential for ensuring that the model does not produce false positives. A high recall is essential for ensuring that the model does not produce false negatives. The F1 score, which takes into account both precision and recall, provides a balanced evaluation of the model's performance [28]. Below displays the formulas mentioned above which consider the True Positive ( $TP$ ), False Positive ( $FP$ ), and False Negative ( $FN$ ):

$$\text{precision} = \frac{TP}{TP+FP} \quad (2.1)$$

$$\text{recall} = \frac{TP}{TP+FN} \quad (2.2)$$

$$\text{F1 Score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (2.3)$$

The model itself returns back the bounding box coordinates. When integrating the model into the pipeline, we conduct post-processing to draw and visualize the bounding boxes on top of the input image. Figure 2.12 displays the output of the cotton bloom detected image from the AI model.



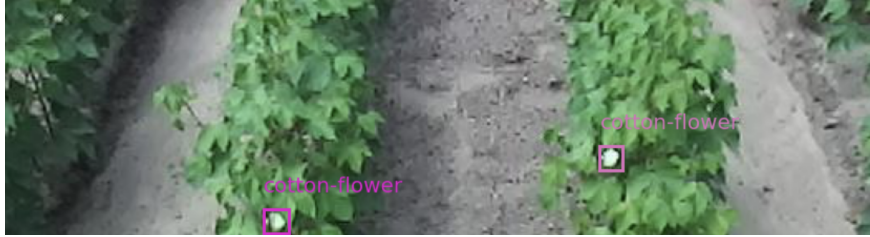


Figure 2.12: Example of pipeline output after post-processing and adding bounding boxes for cotton bloom detection visualization.

## 2.5.2 Azure Data Factory

In order to connect the trained AI model into the rest of the Azure Data Factory Pipeline, we first created a Standard Kubernetes cluster. We then deployed the model into Kubernetes which provides a REST API to interact with.

### REST API vs. Blob Storage Ingestion

Previously, we ingested data from Azure Blob Storage into Azure Data Lake to demonstrate the feasibility of ingesting data from external IoT devices. To assess the performance of the ingestion process, we conducted an experiment using image data and the REST API connection provided by the Data Lake. Initially, we utilized the popular API development and testing tool, Postman, to conduct a synchronous request and observed a substantial improvement in ingestion time. It is commonly used for testing and debugging API applications, and can be used to make both synchronous and asynchronous requests [29]. The implementation of this reduced the stream ingestion time from 12 seconds to 150 ms. This not only highlights the applicability of the REST API connection but also its efficiency in speeding up the ingestion process.

While Postman is a useful tool for testing and debugging APIs, it is not the only option for making HTTP requests to devices. To scale up for batch processing, we adopted asynchronous Python code for HTTP connection. The original ingestion time from blob storage to the data lake took around 2 minutes for 9,000 images (157.6 MB). With the optimization of the REST API and asynchronous Python code, the batch ingestion process was completed in just 8.62 seconds, a marked improvement from the previous ingestion time.

For future purposes, one can use the REST API and HTTP connection with other devices and systems (mobile devices or IoT devices). The pipeline is compatible with the integration of machine learning models into a wide range of applications and systems.

## Kubernetes

We also optimized our Kubernetes configurations by increasing the number of nodes and node pools. When testing on a smaller batch amount of 100 images, integrating 5 nodes rather than 3 nodes in the Kubernetes cluster decreases runtime from 32 minutes to 28 minutes. Increasing the number of node pools from 1 pool to 2 pools decreased runtime from 28 minutes to 22 minutes.

### Asynchronous vs. Synchronous Processing

Asynchronous programming allows the execution of multiple tasks to run concurrently without waiting for the completion of prior tasks. The `asyncio` library, a built-in library in Python, provides the infrastructure for writing asynchronous code with the help of the `async` and `await` keywords [30]. Additionally, the `aiohttp` library enables asynchronous support for HTTP requests, allowing for concurrent processing of multiple requests without waiting for responses [31].

The `aiofiles` library, on the other hand, offers asynchronous support for file operations such as reading and writing to files. This can be useful in programs that need to perform numerous file operations simultaneously, such as our program that handles a significant amount of images [32]. Our pipeline runtime for processing 9,000 batch images was found to take approximately 3 hours and 50 minutes with synchronous code. After optimizing the pipeline with asynchronous code, the execution time was reduced to 34 minutes, which represents a substantial improvement. This demonstrates the potential benefits of implementing asynchronous processing in our pipeline

### 2.5.3 Cost

Although Azure is not an open-source environment, the pay-as-you-go service makes sure to charge resources that are effectively used. With Microsoft Azure, we can spin a 100-node Apache Spark Cluster in less than ten minutes and pay only for the time the job runs on the specific cluster [15].

We used a computer cluster that had the GPU infrastructure for the YOLOv5 training. This costs \$1.14 per hour. The total time spent training was 1 hour and 6 minutes. The total cost is as follows: using the virtual machines led to a cost of about \$101.41 due to the offline model training, storage cost \$13.98, container registry was \$2.49, utilizing a virtual network was \$1.33, Azure Databricks connection was \$0.30, and Azure Data factory led to an additional cost of \$0.30. Moreover, the deployment of the Kubernetes cluster proved to be the most expensive component, incurring a cost of 20 per hour when the cluster is deployed and active. As for scaling considerations, the primary increase in cost would primarily stem from data storage requirements. Azure's pricing for 1 terabyte of storage amounts to approximately \$15.56 per TB.

## 2.6 Future Directions

The pipeline can be further optimized by updating computing clusters with higher computing power and incorporating GPU processing to reduce the total processing time. Moreover, the pipeline currently takes approximately three minutes to reactivate the terminated Databricks interactive cluster, which could be improved through the use of pools.

A bottleneck encountered during the data processing was the connection to the Internet to send images to the Kubernetes cluster through REST API. To address this issue, we can utilize Databricks MLFlow by downloading the model within the Databricks environment itself rather than having to create a separate Internet connection. We refer back to Figure 2.8 to gain a better understanding of the bottleneck at step 3 to where the cluster must create an Internet connection to the REST API URL. If we wanted to scale up with more nodes, the price of Kubernetes would increase to even up to \$1,000 monthly. This further suggests the benefit of utilizing Databricks MLFlow and downloading the model itself rather than using Kubernetes' REST API for AI Model connection. Another bottleneck encountered is the limitations of OpenCV when drawing bounding boxes. Despite our efforts to optimize results through asynchronous Python code, OpenCV does not have the capability for asynchronous processing. As a result, it is incapable of performing the task of producing bounding boxes concurrently for images. This is because OpenCV relies heavily on the CPU, which is fully operated without waiting for any external input. This results in a linear process when drawing bounding boxes, despite the rest of the code being optimized for concurrent execution. To overcome this issue, we can incorporate PySpark, a Python library for distributed data processing using Apache Spark. PySpark allows us to leverage the power of Spark, which is a distributed computing platform that enables fast and flexible data processing. This is compatible with our pipeline because our Databricks runtime version 11.3 LTS includes Apache Spark 3.3.0, and Scala 2.12. With the use of PySpark, we can employ the parallel computing power of our Databricks cluster and enhance the speed and efficiency of our data processing operations.

Overall, these optimization strategies can be used to scale up the pipeline and decrease the total processing time, making it more efficient and effective for handling much larger datasets.

## 2.7 Conclusion

This study has presented a new big data pipeline for cotton bloom detection using a Lambda architecture and Microsoft Azure's cloud computing resources. The pipeline fulfills data preprocessing, object detection using a YOLOv5 neural network trained through Azure AutoML, and visualization of object detection bounding boxes. The results of the study demonstrate the high performance of the neural network with a Mean Average Precision (mAP) score of 0.96 and an optimized runtime of 34 minutes when evaluated on over 9,000 images. This work showcases the scalability of the presented pipeline as a solution for deep learning-based object detection and emphasizes the potential of employing cloud computing resources for big data processing in precision agriculture. This study advances the field by expanding and demonstrating the big data pipeline implementation of a new method for cotton bloom detection from

images collected on cotton. The results obtained in this study suggest a scalable Lambda architecture that can be implemented for big data processing using Azure resources.

## CHAPTER 3

# DIMENSIONALITY REDUCTION OF HIGH-THROUGHPUT PHENOTYPING DATA IN COTTON FIELD

1

---

<sup>1</sup>Issac, Amanda. Published in the Proceedings of IFAC, Accepted by *Agricontrol*. Reprinted here with permission of publisher. March 30, 2023

## Abstract

In this paper, we implement data reduction methods to reduce the size of image datasets from cotton fields for use in a high-throughput phenotyping (HTP) pipeline in order to allow for data transfer more quickly over poor internet connections. We investigate dimensionality reduction methods to accomplish this goal. Specifically, we utilize Principal Component Analysis (PCA) to compress image data into a smaller dimension space, which when uncompressed retains significant variability from the original image. To demonstrate the ability of PCA to produce quality reconstructions, we consider the example use case of detecting cotton bloom flowering patterns with reconstructed images. We employ Open Source Computer Vision (OpenCV) to generate pixel-wise masks which both further reduces the byte size of data and successfully identifies cotton bloom flowering. The results indicate a high amount of data reduction from the original to the reconstructed images; byte sizes reduce 93% through PCA while preserving around 98% variance when using a much smaller number of components. Bitwise masking with OpenCV yields a 99% reduction in file size. The results demonstrate great potential in employing machine learning techniques for the data reduction pre-processing step prior to performing subsequent analysis. This data reduction is a crucial step in developing a field-based HTP big data pipeline.

Computer Vision; Cotton Phenotyping; Principal Component Analysis

### 3.1 Introduction

The agriculture sector faces immense pressure to be more sustainable through the increase in demand due to the rapid growth of population. Specifically, one large sector of agriculture which encounters immense challenges regarding yield prediction is cotton production, which is commercially cultivated in the world with an annual economic impact. Cotton production operation faces numerous challenges, of which timely harvesting of quality cotton fiber is the most pressing. The waiting time exposes the open bolls to harsh conditions that degrade quality. Furthermore, the cotton harvesting is expensive. Cotton harvesting combines weigh more than 33 tons and can also generate soil compactness that would further reduce land productivity [10]. Lastly, the lack of the skilled workforce along with other external factors such as climate change, the decrease in arable land and shirking water resources continue to challenge sustainable agricultural production [8].

Agricultural production benefits from precision farming, where crop health, soil properties, and yield can be monitored and lead to efficient decision-making for sustainable agriculture production through Computer Vision (CV). CV techniques and data-driven approaches for precision farming have proven to yield sustainable agriculture production and management. However, agricultural models from the past suffer due to data scarcity impacted by the dependence on availability and data variety. Huge number of objects in farms connected to the Internet can produce an immense volume of data that must be stored, processed, and made available in a continuous and easy-to-analyze manner. High masses of features within data often lead to “the curse of dimensionality” where classifiers are hindered by the imbalance between the training set and features [7]. Therefore, one of the main steps in data processing is dimensionality

reduction, which permits the identification of redundant information that could minimize classifier performance. Within dimensionality reduction, methods are either supervised or unsupervised. Supervised methods incorporate datasets with labeled samples to infer class separability [33]. However, often times, prior labeling is not available, and therefore unsupervised dimensionality reduction has gained popularity for remote sensing applications.

Several previous studies have examined and addressed the challenges of data reduction in the agriculture sector. [7] incorporated dimensionality reduction for images used in crop differentiation in Mediterranean climate using PCA. [34] employ High Order Singular Value Decomposition (HOSVD) for data reduction in precision agriculture. However, the purpose of this work is to **employ unsupervised data dimensionality reduction methods as part of a big data pipeline our team is developing for field-based cotton phenotyping**. While there exists a few studies with data reduction within the agriculture sector, **this is the first application of data reduction tools for field-based cotton phenotyping by incorporating CV and unsupervised machine learning**.

The remainder of this paper is organized as follows: Section 2 describes the field under study, equipment to collect data from the cotton field and the generated dataset. Section 3 presents our pre-processing experimental results based on the proposed methodology and the implementation of different methods for reducing data dimension. Section 4 provides a summary of our work presented in this paper and areas for future work, and concluding remarks.

## 3.2 Data Collection

### 3.2.1 Description of our cotton research farm

The cotton bloom data was collected from a stereo camera that was installed on an autonomous ground vehicle deployed in a research farm at the University of Georgia's Tifton campus in Tifton, GA. An aerial view of the farm is shown in Figure 3.1. The treatments described in Figure 3.1 are 4-rows wide, but we collected yield on the inner 2-rows for analysis later.

### 3.2.2 Cotton field data collection

The rover utilized for the cotton plant data collection was a four-wheel hydrostatic machine (West Texas Lee Corp., Lubbock, Texas) as described in [1]. The rover's length was 340 cm, and its front-axle and rear-axle were 91 cm from the center with a 91 cm ground clearance. This vehicle is powered by the Predator 3500 Inverter generator, and can also be remotely controlled and uses the Nvidia Jetson Xavier for vision and navigation systems. The rover travels at approximately 2 kilometers per hour and operates with two 12-V car batteries to power the electronics, along with a ZED RGB stereo camera.

The RGB stereo camera's left and right sensors were 120 cm apart. The sensors were placed 220 cm above the ground, facing downward [1]. We selected the ZED camera due to its efficiency in operating outdoors and providing depth data in real time. The ZED camera collects 4-5 frames per second, and

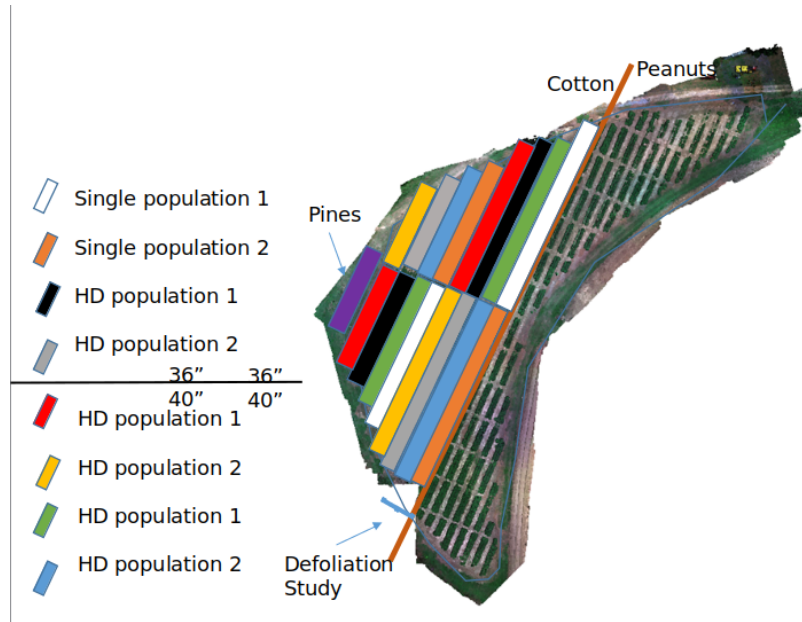


Figure 3.1: Aerial view of our cotton farm in Tifton, GA displaying 40 rows of cotton plants, treatments of two planting populations (2 and 4 seed per foot), HD (Hilldrop), and single planted cotton seed. Two row spacing of 35 inches and 40 inches were also used as treatment. Each treatment was 4 rows wide and 30 feet long. There were three repetitions per treatment.

recorded a video stream of each 4-row treatment from June 2021 to October 2021 2-3 days per week as a ROS bag file.

### 3.2.3 Dataset creation

The camera had two lenses and captured a left view and right view of the cotton plants. A total of 765 image frames were extracted from sixteen 4-row treatments on each data collection day between July 14, 2021 and August 6, 2021. Blooms began to grow on July 14, and the 765 image frames were split in half to separate the left and right lens views and resulted in 1,530 frames.

## 3.3 Data Processing

The use of large datasets has led to great potential for abundant applications such as urban mapping, environmental management, vegetation, and crop supervision. Dimensionality reduction is one of the main hindrances in image processing due to the high variability and high dimensional nature of the images. The aim of dimensionality reduction is to eliminate redundant information and simplify subsequent processing steps.



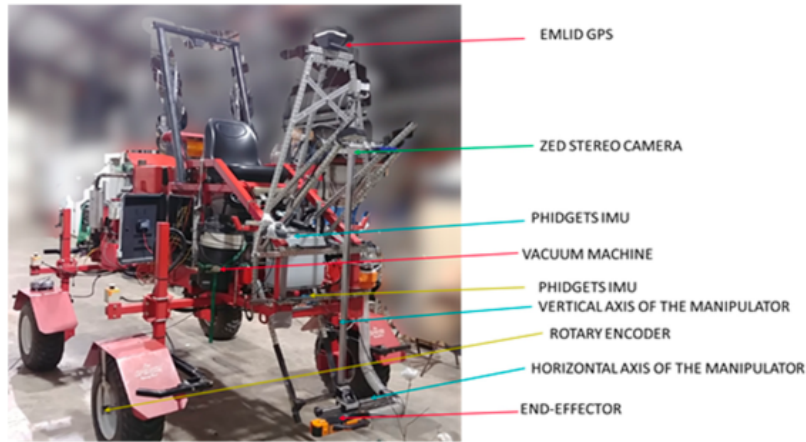


Figure 3.2: Front view of the rover with the robotic arm, vacuum, and sensors mounted on the rover (see [1] for details) that was used to collect video streams of cotton plants in Tifton, GA.

### Data reduction:

Real-time analysis, data aggregation, tagging, and data reduction may be performed locally where the data is collected. Some sensors also extract an extensive amount of data that does not necessarily contain rich information. Data reduction minimizes data volume without removing significant components. Dimensionality reduction is useful to handle the massiveness of big data by reducing the many-variables data into a manageable size. Some components which alter the variability of data include volume, velocity, and variety. The data size correlates with the big data volume, and the big data velocity depends on the data stream's frequency when entering the system. [35].

### 3.3.1 The use of OpenCV for real-time data processing

OpenCV is a library of programming functions mainly aimed at real-time computer vision. It pre-processes and prepares raw images that are required to be transformed into specific features which can later be incorporated into computer vision or machine learning algorithms. For the purpose of this work, we implemented the masking technique through OpenCV, which highlights a specific object from the image. This approach enables us to focus solely on the portions of the image that are of interest, while effectively reducing the byte size of the image. In Python, OpenCV Color Detection represents colors using a color-space or color model, which defines a range of colors as tuples [36]. Specifically, RGB color model categorizes colors as tuples with three components, each ranging from 0 to 255, where (0,0,0) denotes black and (255,255,255) signifies white. Given that the cotton bloom dataset images contain cotton flowering that can be detected by its white-colored pixels, we opted to implement a mask to isolate the background of the image, including leaves, ground, and shadows, in order to solely highlight the white flowering for further analysis.

We employed the lower hsv (hue, saturation, and value) range of [30, 0, 255-sensitivity] and an upper range of [255, sensitivity, 255]. The sensitivity was set to 30 in order to take into account a variety of pixel coloring due to shadows and the variation of sunlight. This tuple was calculated by taking the average hsv range between a subset of 184 images (left and right lens images for one 30 ft treatment). Figure 3.3 shows an example from August 6, 2021 prior to implementing bitwise masking using OpenCV. Figure 3.4 illustrates the post-processed (after bitwise masking) results of Figure 3.3.



Figure 3.3: An illustrative (original) image from the cotton blooms dataset displaying 3 cotton bloom flowering from August 6, 2021 prior to implementing bitwise masking.

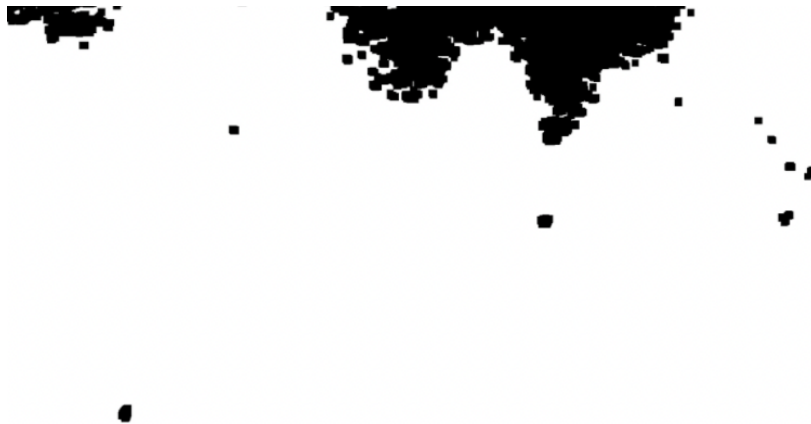


Figure 3.4: Manipulated image from August 6, 2021 after implementing bitwise masking.

The masked image led to a reduction from 171.47 KB to 34.17 KB, which is a byte size reduction of 80.07%. In addition, the majority of the false positives in the output were clusters of white-colored pixels from the soil. Figure 3.5 illustrates the contours of all white-colored pixels detected by OpenCV. Therefore,

we altered the contouring of classified pixels by only selecting pixel clusters that had an area of over 150 pixels. Figure 3.6 portrays the image contours with the pixels that are now detected after including the bound of pixel areas over 150.



Figure 3.5: An illustrative image from August 6, 2021 with the contour highlighting all pixels within the white detection range.



Figure 3.6: An illustrative image from August 6, 2021 with the contouring of white pixels with the area less than 150 pixels.

Although this significantly reduced the amount of false positives, there still continued to be a high rate of false positive pixels. A majority of these false positives resulted from the computer vision technique characterizing the lighter colored soil as white cotton bloom pixels. We can see this on the top and right section of Figure 3.6.

When examining the dataset, the size of the white cotton flowering appeared to be much smaller in comparison to the image background. This, therefore, made it difficult to accurately mask the white flowering, as they were classified as the background and led to an increase in false negatives. Furthermore,

using the more wide-spanned image frame (e.g., as seen in Figure 3.3) failed to portray certain white blooms towards the edges of rows and hidden by the surrounding green vegetation. Because a large portion of the dataset had overlapped frames, we sliced the image frame with sections that were at a closer angle to the camera. We cropped the image frames manually to increase the size of the cotton flowering with respect to its background. This would allow us to get a higher proportion of the flowering in comparison to its background, and therefore output a more clear image that was easier to detect the flowering while retaining the detailed shape of the cotton flowers.

In examining our dataset containing 184 images for August 6, we realized that 11 close-up sliced images would capture all cotton flowerings for the entire treatment. A large portion of the dataset of 184 images were repetitive images, frames that overlapped, blurred images, or images that did not adequately depict the rows of the cotton blooms, such as images showing only the ground.



Figure 3.7: An illustrative image of 4 out of 11 slices from the full frame image in Figure 3.3.



Figure 3.8: The result of applying bitwise masking to the slices from Figure 3.7.

Reducing the file size of 184 images to 11 sliced images led to the reduction from 75,500 KB to 3,834 KB - a reduction of 94.92%. Within the 11 sliced images using masking, we were able to reduce the overall size from 3,834.37 KB to 508.46 KB – a percent reduction of 86.74%. Overall, the masking technique

using OpenCV led to 99.33% decrease from the 184 original images to the masked sliced images within the dataset used from August 6.

Although the portion of the dataset we experimented on was reduced by about 99% in byte size, there are instances where the shape of the blooms is not accurately retained due to discoloration from sunlight and shadows. Masking portrays a high data reduction, as well as accurately depicting a vital component of the dataset, cotton bloom flowering. However, we wanted to test a method that would guarantee the visibility of the cotton flowering within the reduced image. Such a method would also allow us to extract other components from the image such as leaf coloration for future studies in the cotton bloom dataset.

### 3.3.2 The use of PCA for data reduction

We investigated the use of Principal Component Analysis (PCA) to examine its potential for data dimensionality reduction aiming at reducing the complexity in high-dimensional data. High-resolution image data often leads to high feature counts, where each pixel is considered its own dimension. This may potentially hamper the performance of classifiers by the imbalance between the samples and features [7]. A solution for this problem is feature extraction, which involves replacing the original data with a new collection of features. A common feature extraction method is PCA which transforms the data by projecting it into lower dimensions, called principal components, with the goal of summarizing the structure of the dataset using only a limited number of principal components [37].

PCA finds the top eigenvalues or components that capture the image data variability, which then can be used to reconstruct the image. This is a lossy compression method, but if enough of the variability is captured in the transformation process, then the process yields good quality results after reconstruction. We incorporate this process into our pre-processing pipeline to further reduce the dimensionality of the data, ultimately speeding up file transfer speeds over an internet pipeline.

For our purposes, we used the scikit-learn default implementation of PCA [38]. Scikit-learn uses the LAPACK implementation of the SVD in order to project the data linearly into a lower dimensional space. In order to determine an acceptable number of principal components to keep, we visualize the explained variance as a function of the number of components. To accomplish this, we applied PCA to one of our images (in our case, the slice tested was of dimensions  $600 \times 2,832$ , so the total number of components were 600) and created a plot to see to what extent PCA captured variance in the data. This is shown in Figure 3.9. According to the plots, we determined that after about 200 principal components, the

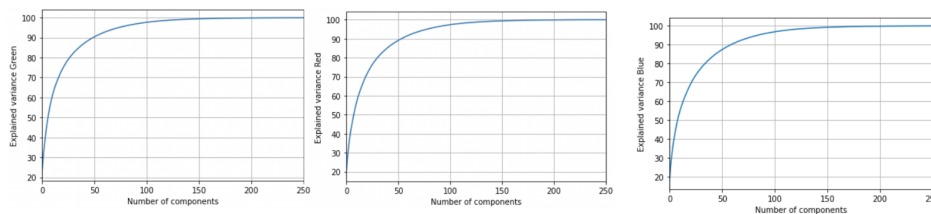


Figure 3.9: Explained variance screen plots for each color channel for a given image.

explained variability approached 100 percent. Meanwhile, 100 principal components was sufficient to keep roughly 97% to 99% of variability in our image, an acceptable amount of compression. Thus, we initialized our PCA models with 100 components.

The cotton bloom dataset consists of  $720 \times 1280$  color images. Keeping in mind the three color channels for each image, the original image data contained  $720 \times 1280 \times 3$  or 2,764,800 data points. As each pixel is an integer of one byte, this means opening each image from the dataset takes up 2.7648 MB. Given a  $720 \times 1280$  image from the dataset, we first split up the images into its three color channels, red, green and blue. This yields three arrays, which we normalize by dividing by 255, obtaining floating point values from 0 to 1.

Three PCA models – one for each color channel – are then initialized with  $n$ -components set to 100. The arrays for each color are fed into the corresponding PCA. The three resulting transformed arrays are of size  $720 \times 100$ . These floating point values are by default stored in float64 type variables, taking up 8 bytes each. We convert these to type float16, only 2 bytes. Of course, this conversion loses some precision, but as we are interested in data compression, we are not concerned as long as the resulting reconstructed image is still reasonably clear. An example of these reconstructed images is shown in Figure 3.11. As is evident from the figure, the reconstructions clearly preserve the cotton blooms when present. Unlike binary masks, PCA reconstructions preserve a majority of the original image’s attributes such as leaf coloration and texture. While binary masks alone display a high level of compression, they only provide shape attributes of objects which can only be suitable for identifying cotton blooms in this case. Utilizing PCA method to provide compressed versions of the original images lends itself to additional use cases and multiple avenues for future analysis, and therefore is an ideal pre-processing step for the data.

Table 3.1: Using the nbytes command in Python, the counts for the number of bytes in a directory containing 11 sliced cotton data images, and a directory containing the numpy arrays representing the images in the compressed form of their top 100 principal components.

	Bytes
Image directory size	51,608,256
Components directory size	3,590,400
Percent reduction	93.043

Table 3.2: The total number of bytes in a directory containing 11 original cotton field images compared with a directory containing 11 reconstructed images, after PCA compression.

	Bytes
Original directory size	3,834,364
Reconstruction directory size	1,673,296
Percent reduction	56.361

This dimensionality reduction process led to a 93% data size reduction by the number of bytes in Python, as shown in Table 3.1. As observed from Table 3.2, JPEG compression is quite efficient and we found that when stored on the computer, each image averaged around 348.58 KB. Compared to the approximately 152.12 KB size of each reconstructed JPEG, these results indicate that there is a 56% data

size reduction. Thus, our methods were successfully able to reduce the dimensionality of the dataset and the number of bytes of data quite drastically.

As each PCA model in this method is fit to each particular image, in an actual end-to-end pipeline, we would have to save the PCA models themselves for each image, to reconstruct directly from the compression vector in Python. In this case, we could not use just one saved PCA model to inverse-transform the whole set, because PCA does not generalize across the image dataset. Storing and sending the PCA models themselves adds to the amount of data we must send to effectively implement the PCA dimensionality reduction strategy. Alternatively, the reconstructed images were still on average quite reduced from the original slices, and the reconstructions could be sent over an HTTP connection directly. This spares the need to send a PCA model, and still leads to a good file size reduction. Figure 3.10 illustrates the original image, while Figure 3.11 shows the reconstructed one for comparison.



Figure 3.10: The original slice 7 of the cotton bloom dataset on August 6.

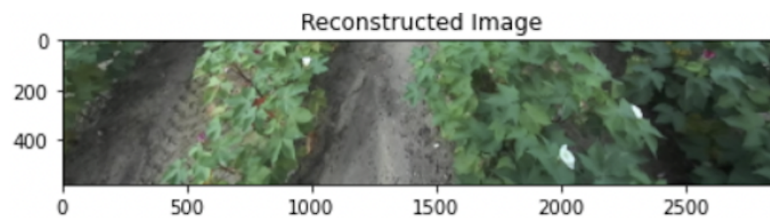


Figure 3.11: The reconstructed image using the original slice from Figure 3.10 after being transformed through PCA and inverse-transformed displaying 3 cotton blooms.

### 3.3.3 Combining PCA and masking

We then evaluate the usability of PCA-reconstructed images for post-processing by applying masking techniques for cotton bloom detection and assess the feasibility of obtaining desirable results. Upon conducting masking on the original Slice 7, as depicted in Figure 3.10, the output JPEG image was significantly reduced from 337.65 kB to a mere 23.74 kB. In contrast, when applying bitwise masking on the reconstructed Slice, as shown in Figure 3.11, the JPEG image size decreased from 146.58 kB to 23.29 kB. This highlights that implementing masking on the reconstructed image with a dpi of 100, compared

to the original 144 dpi, resulted in a further reduction in byte size of approximately 2%. Moreover, the bitwise mask applied on the reconstructed image successfully detected all three cotton blooms with 100% accuracy, while preserving the shape of the flowering, as illustrated in Figure 3.12 for the original masked image, and Figure 3.13 for the masked image using the PCA-based reconstructed image.



Figure 3.12: The masked image (23.74 kB) using an original slice of the cotton bloom dataset on August 6 from Figure 3.10.

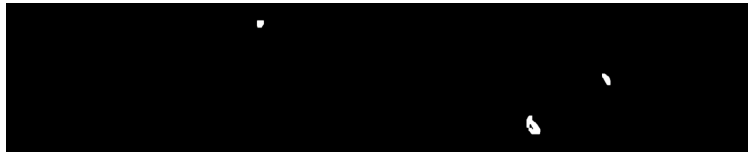


Figure 3.13: The masked image (23.29 kB) using the reconstructed slice 7 of the cotton bloom dataset on August 6 from Figure 3.11.

We finally test the post-processing bitwise masking on additional reconstructed slices using images from July 26, July 29, and August 6. Figure 3.14 shows the results of cotton flower detection. Bitwise masking on the original slices led to a true positive rate of 0.9022, while the bitwise masking for the reconstructed images led to a true positive rate of 0.8913. Although there seems to be a very small drop in the positive rate for the reconstructed images compared to the original slices, the reconstructed images do well considering that the byte size was reduced by 93 percent. In addition, the reconstructed images yield less false positives. This may be due to the smoothing effect of the reconstruction; by smoothing an image prior to applying post-processing techniques, we are able to reduce the amount of high-frequency content noise.

### 3.3.4 Discussion and future work

Ultimately, this work is aimed at an application in a high-throughput phenotyping (HTP) pipeline. The various data reduction results we examined appear to be effective. Thus, with the help of our methods, we would be able to reduce the image size prior to sending it over to the HTP pipeline, and then decompress the image to retain significant variability in the data. While bitwise masking with OpenCV proved to be difficult due to the amount of unwanted false positives, masking combined with image slicing seemed to be effective. Still, we believe that this type of masking may be an effective strategy for future investigation without the slicing pre-processing step.



	Reconstructed Images	Original
Total TP	82	83
Total FN	10	9
Total FP	9	12
True positive Rate	0.8913043478	0.902173913

Figure 3.14: Cotton flower detection using bitwise masking on reconstructed images (reduced by more than 93% from the original byte size) has a true positive rate of 0.8913. Cotton flowering detection using bitwise masking on the original images leads to a true positive rate of 0.9022.

Alternate methods for bitwise masking may be effective and considered in future work for post-processing. For instance, deep learning-based methods common in image segmentation or object detection problems, such as U-NET, could be used with hand-labeled bitwise masks to more easily identify the cotton blooms in the training and validation sets for post-processing. These methods are more robust than HSV filters but require much more labeled training data. We can use the existing OpenCV method to speed up creating the labeled masked dataset. PCA could be applied as a pre-processing step to reduce image size before passing in these images to a more advanced image segmentation network. However, developing and testing such networks is not in the scope of this paper, in which we primarily aim to validate the use of PCA as a data reduction step.

### 3.4 Conclusion

In this paper, we presented a method for data reduction pre-processing, while also showcasing the detection of cotton blooms. To reduce the dimensionality of the data, we leverage the power of Principal Component Analysis (PCA). Additionally, we utilize OpenCV to process the full frame of the cotton field image and accurately detect cotton blooms. Despite achieving a reduction in size, we acknowledge that there were some false negatives and positives in the initial bloom detection results. To address this, we implement image-slicing techniques to further enhance the masking of the cotton flowering. Notably, each step of PCA, bitwise masking, and image slicing independently results in a significant decrease in file size, surpassing the performance of standard JPEG compression. By integrating the processes of bitwise masking with PCA on sliced images, we successfully achieve a joint reduction in dimensionality and file size, thereby elevating the effectiveness of our proposed method.

## CHAPTER 4

# OPTIMIZING A BUILD PIPELINE FOR PROCESSING LARGE-SCALE 2022 DATA

1

---

<sup>1</sup>Issac, Amanda. To be submitted to *Agricultural Systems*

## Abstract

We present additional contributions to our proposed image processing pipeline for handling large datasets. In addition to testing the scalability of the pipeline on new 2022 data, which has distinct characteristics from the 2021 data on which the pipeline was initially built, we showcase its effectiveness in processing big data with high accuracy and efficiency. To further optimize the pipeline, we introduce a combined compression approach of downsampling and masking as a pre-processing step, which leads to a reduction in file size and pipeline execution time while preserving detection/counting accuracy. The results demonstrate that the compression technique achieves a significant reduction in image size without sacrificing detection model accuracy. The proposed pipeline with the added pre-processing step is expected to enable high-throughput phenotyping with faster data transfer, facilitating improved decision-making. Our findings contribute to the field of image processing and have practical implications for improving the efficiency and scalability of image processing pipelines in various domains.

## 4.1 Introduction

The increasing availability of high-resolution imagery in agriculture has led to a growing demand for efficient image-processing pipelines that can handle large datasets. Such pipelines are critical for enabling high-throughput phenotyping and facilitating improved decision-making in precision agriculture. Such acquired data possesses the characteristics of high volume, value, variety, velocity, and veracity, which are all characteristics of big data. In order to leverage the data for informed decisions, a big data pipeline would be needed. Therefore, it is imperative to develop techniques that can process big data quickly and accurately, but also reduce the high dimensions while retaining the essential components to avoid resource consumption for storage.

One area of agriculture that faces particular challenges with regard to yield prediction is cotton production. Cotton production is a complex and labor-intensive process that requires careful management to ensure high-quality fiber yield. One of the unique challenges in cotton production is determining the optimal time for harvesting. Harvesting cotton at the right stage of maturity is crucial to avoid the degradation of cotton fiber quality due to exposure to unfavorable environmental conditions [9]. Cotton production also faces challenges related to harvesting costs and limited arable land. To address these challenges, image processing pipelines that can handle large datasets of high-resolution imagery are essential in cotton production. However, the sheer volume of high-resolution imagery data generated from cotton fields poses challenges in data handling and processing. This necessitates the need for an efficient pipeline that can handle big data with high accuracy and efficiency, including the use of data compression techniques.

In this paper, we present additional contributions to our proposed cotton bloom detection pipeline, which is designed to handle big data with high accuracy and efficiency. We extend our previous work by testing the scalability of the pipeline on new 2022 data, captured using ZED2 and ZEDM cameras, which have distinct characteristics from the 2021 data on which the pipeline was initially built. Moreover, we introduce a combined compression approach of downsampling and masking as a pre-processing step to

further optimize the pipeline. This approach aims to reduce file size and pipeline execution time while preserving model accuracy.

### 4.1.1 Downsampling

Downsampling is a technique used in image processing to reduce the resolution of an image while preserving its essential features. The downsampling algorithm averages the pixels in the image by taking the mean value of a block of neighboring pixels. The size of the block depends on the downsampling factor, which is the ratio of the original image size to the downsampled image size. The averaging process is performed separately for each color channel in the image (red, green, and blue in a color image), so the resulting downsampled image has the same number of color channels as the original image. Refer to Figure 4.1 for a visual representation of the downsampling algorithm.

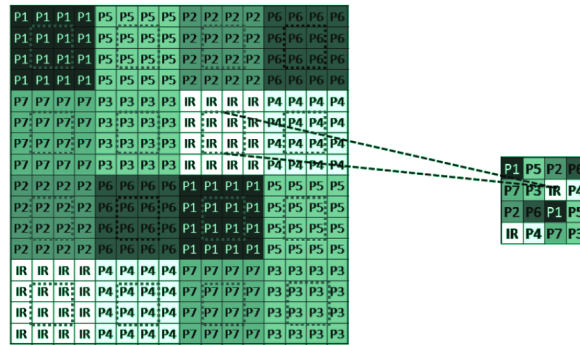


Figure 4.1: A graphical depiction of the Downsampling Algorithm.

### 4.1.2 Masking

Masking is a widely-used technique in computer vision and machine learning that allows for the isolation of specific regions of interest in an image while eliminating irrelevant background details. It creates a binary mask that highlights the regions of interest by converting them into white pixels while marking the non-regions of interest as black pixels. Once the mask is created, it is applied to filter and isolate the regions of interest, enabling the extraction of meaningful information from complex image data. Masking also plays a crucial role in image compression, as it helps remove irrelevant background information and isolates only the regions of interest, resulting in more effective compression of the images. This makes the masked data ideal for use in environments with limited storage capacity.

### 4.1.3 Related Work

Several studies have explored compression techniques to speed up pipeline execution time in various domains. [39] introduces the use of a comprehensive pipeline for brain image analysis that incorporates

compression methods to optimize pipeline execution time and storage space, and highlighted the effectiveness of the compression technique in managing big data in a resource-intensive image analysis pipeline. [40] compares the use of binary data format and NVIDIA DALI as compression techniques to accelerate the data loading and data augmentation pipeline in the training of large-scale deep neural networks. This suggests that compression techniques can be effective in speeding up the pipeline execution.

#### **4.1.4 Contributions**

In this study, we present our contributions to the development of an optimized big-data pipeline for detecting cotton blooms. The proposed pipeline leverages downsampling and masking for image compression to reduce the size of the image data and improve the computational efficiency of the detection algorithm.

To evaluate the performance of the proposed big data pipeline, we conducted extensive testing and optimization, focusing on its scalability and effectiveness in handling large volumes of data generated by monitoring cotton fields for blooms. We also evaluated the performance of the YOLO model trained on 2021 data on the 2022 dataset, which provided new insights into the detection of cotton blooms. This evaluation demonstrated the robustness of the YOLO model and its ability to detect blooms in new datasets.

Furthermore, we validated the performance of the proposed pipeline on new 2022 data and demonstrated its ability to accurately detect blooms in real time with high accuracy. Our findings indicate that the proposed pipeline can be applied to future and similar large-scale applications in the agriculture industry.

The testing of the proposed big data pipeline on larger scales of data was critical in ensuring its effectiveness in real-world applications. Our findings showed that the pipeline is capable of processing large amounts of data in a timely and efficient manner, making it highly suitable for use in large-scale applications.

Overall, our study not only optimized the big data pipeline for detecting cotton blooms using downsampling for image compression but also demonstrated its effectiveness in handling large volumes of data. Our research contributes to the development of accurate and efficient big data pipelines for monitoring cotton blooms, which has significant implications for the agriculture industry.

The rest of the paper is organized as follows. Section 2 provides an overview of the dataset used in this study, including details about the ZED2 and ZEDM cameras. Section 3 describes the methods employed in our proposed pipeline, including the post-testing on 2022 data, downsampling for compression, masking, and scaling up. Section 4 presents a discussion of our results and their implications, along with suggestions for future work. Finally, Section 5 concludes the paper and summarizes the contributions of our research to the field of image processing.

## 4.2 Dataset

We focus on the collection of data between July and October 2022. Specifically, data were collected on July 1, July 9, July 13, July 22, August 4, August 10, August 15, August 22, August 26, August 31, September 7, September 16, September 23, October 1, October 12, and October 21.

In contrast to the 2021 data collection, the 2022 data were obtained using two cameras mounted on the rover, specifically, the ZEDM and ZED2 cameras. The cameras were positioned differently depending on the date of data collection. On July 1, the bottom camera, ZED2, was located 20 cm from the ground, while the top camera, ZEDM, was situated 1.5 meters above the ground. From July 9 to August 26, the top camera was the ZED2, positioned 1.64 meters above the ground, and the bottom camera was the ZEDM, placed 20 cm from the ground. From August 31 to October 21, the top camera was again the ZED2, located 1.79 meters above the ground, while the bottom camera was the ZEDM, positioned 28 cm from the ground. A detailed account of the positioning of the ZEDM and ZED2 cameras can be found in Table 4.3, and a representation of the camera positions with respect to the field is provided in Figure 4.2. Figure 4.4 illustrates an example of a 2022 cotton field data from the zed 2 camera after bagfile image



Figure 4.2: An example of cotton field image data extracted from the ZED2 and ZEDM camera for 2022 cotton field data.

extraction. The image displays both the right and left lenses. Figure 4.5 illustrates an example of 2022 cotton field data from the ZEDM camera after bagfile image extraction. The image displays both the right and left lenses.

## 4.3 Methods

### 4.3.1 Post Testing YOLO model on 2022 Data

#### ZED2 Camera

In the analysis of ZED2 camera data for cotton field imaging, the pre-processing of 2021 data involved two primary steps. Firstly, the frames obtained from the bagfile included both the left and right lenses,

Dates (in 2022)	Camera mounted position on the Rover	
	Zedm	Zed2
July 01	Top: 1.5m	Bottom: 20cm
July 09	Bottom: 20cm	Top: 1.64m
July 13	Bottom: 20cm	Top: 1.64m
July 22	Bottom: 20cm	Top: 1.64m
July 29	Bottom: 20cm	Top: 1.64m
August 04	Bottom: 20cm	Top: 1.64m
August 09	Bottom: 20cm	Top: 1.64m
August 15	Bottom: 20cm	Top: 1.64m
August 22	Bottom: 20cm	Top: 1.64m
August 26	Bottom: 20cm	Top: 1.64m
August 31	Bottom: 28cm	Top: 1.79m
September 07	Bottom: 28cm	Top: 1.79m
September 16	Bottom: 28cm	Top: 1.79m
September 23	Bottom: 28cm	Top: 1.79m
October 01	Bottom: 28cm	Top: 1.79m
October 12	Bottom: 28cm	Top: 1.79m
October 21	Bottom: 28cm	Top: 1.79m

Figure 4.3: The positioning of ZED2 and ZEDM camera for 2022 cotton field data.

necessitating their separation into distinct frames. Secondly, due to the relatively small scale of the cotton blooms in comparison to the rest of the frame, slicing the frames into smaller subsections was necessary to facilitate accurate detection by neural network models. However, in contrast to the 2021 data, the cotton blooms in the 2022 data are substantially larger with respect to the frame, indicating the potential for improved detection without slicing. We present figure 4.6 to showcase the pre-processed sliced 2021 data, providing a comparison with figure 4.4 which illustrates an example of an unprocessed 2022 bagfile image frame from the ZED2 camera in a cotton field setting. A total of 40 images acquired on August 26, 2022, were utilized for analysis. Initially, the testing was performed on image frames comprising both the left and right lenses. However, the model's performance decreased in detecting blooms. Figure 4.7 displays the results of the model detecting blooms on 2022 data where the image frame integrated both the left and right lens. Consequently, we pre-processed the image data, split the right and left lens frames separately, and added them to the cotton bloom dataset. An instance of a right frame from the 2022 cotton field data obtained through the zed 2 camera is depicted in Figure 4.8. The YOLO model, which was trained on 2021 data, correctly classified all the blooms from unseen 2022 data. Figure 4.9 showcases the model's classification results of the cotton field image data taken on August 26, 2022.



Figure 4.4: An example of the extracted bagfile frame of 2022 Cotton Field data with the ZED2 camera.



Figure 4.5: A example of the extracted bagfile frame of 2022 Cotton Field data with the ZEDM camera.

### ZEDM Camera

We opted to divide the right and left lenses for pre-processing for the ZEDM camera after observing that the accuracy of the model was maintained when we split the lenses in the 2022 ZEDM data. Figure 4.10 illustrates an example of a right frame of 2022 cotton field data from the ZEDM camera that will be utilized for testing purposes. Nevertheless, the model's performance was unsatisfactory, as it produced a considerable number of false positives and negatives. This can be attributed to the fact that the YOLO model was trained on 2021 data using a ZED2 camera. The frames obtained from the ZEDM camera display the field from a lower angle, which alters the appearance of the blooms compared to those captured using the ZED2 camera. The false negatives mainly resulted from shadows and variations in the orientation of the blooms, while the numerous false positives were partially due to the pixel areas in the image that exhibited varying sunlight contrast between gaps in the leaves. Refer to 4.11 for a display of the results from the YOLOv5 cotton detection on the 2022 ZEDM camera data.

### 4.3.2 Average Downsampling for Optimization

The optimization of the big data pipeline is one of the key objectives of this study. In this regard, we propose a technique to compress image data that can potentially reduce storage requirements and accelerate





Figure 4.6: A display of a pre-processed sliced 2021 image data.

pipeline execution time. Specifically, we explore the efficacy of downsampling as a compression technique. By reducing the size of the images, we aim to achieve a balance between storage size and image quality.

### **2021 Data**

We aimed to optimize our big data pipeline by compressing image data, which has the potential to reduce storage size and improve pipeline execution time. To test this, we first implemented a downsampling technique on the 2021 data collected from a zed2 camera. Previous research on the 2021 data has proven that accurately detecting blooms in the 2021 ZED2 cotton field images is challenging due to their small proportion relative to the background [17]. To address this issue, we pre-processed the 2021 images by dividing them into five equal slices and selected a dataset comprising sliced images from ten specific days in 2021: July 8, July 14, July 16, July 19, July 23, July 26, July 29, August 4, August 6, and September 9. This resulted in a total of 9,018 images with dimensions of 530 x 144 and 3 color channels (RGB) for testing. Before downsampling the 2021 data by a factor of 2, its size was 157.6 MB, and the end-to-end pipeline execution runtime took 34 minutes, including ingesting data, putting it through the trained model, retrieving output, and building visual bounding boxes. However, after downsampling, the 2021 data size was reduced to 52.3 MB, with a shape of 260 x 72, and the pipeline execution runtime was reduced to 22 minutes, representing a decrease of 66.82% and 34.29%, respectively. However, we observed that downsampling the already relatively small sliced 2021 images further, from the initial size of 530 x 144 to 260 x 72, made it difficult for the YOLO model to maintain its prior accuracy. As a result, the model

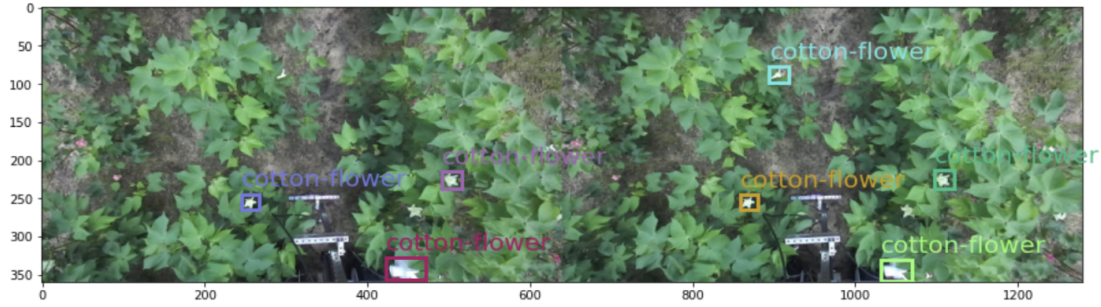


Figure 4.7: An illustration of the results of the YOLOv5 model on detecting cotton blooms on 2022 Cotton Field data with the ZED2 camera integrating both right and left lens from August 26, 2022.

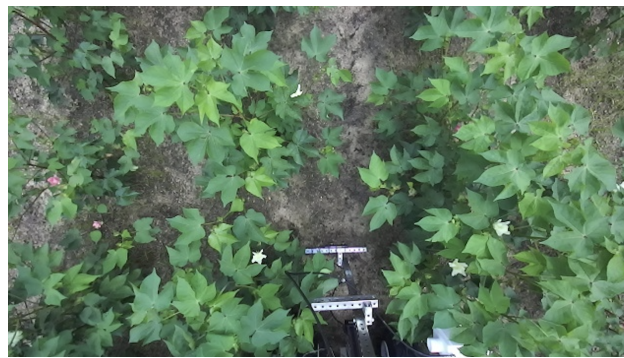


Figure 4.8: A display of the right frame of 2022 Cotton Field data with the ZED2 camera from August 26, 2022.

produced some false negatives due to the small size of the cotton pixels after downsampling. Figure 4.12 illustrates an example of the cotton bloom detection output from the YOLO model after the pipeline execution of the 2021 downsampled ZED2 image data.

### 2022 Data

While downsampling the 2021 data led to some false negatives, the data collection method has been improved for the 2022 data collection process. The orientation of the ZED2 camera is now much closer to the blooms, which has allowed us to capture higher-quality images with greater detail. As a result, we did not slice the 2022 data and the neural network model was still able to accurately detect all blooms. Therefore, we decided to test downsampling by a factor of 2 on the 2022 data, as the shapes of these images are much larger (1434 x 802) compared to the 2021 data. By testing downsampling on the 2022 data, we aimed to determine if downsampling can still be used as an effective technique for optimizing the big data pipeline while maintaining high accuracy in bloom detection for the future.



Figure 4.9: A visual of the cotton bloom detection results of the right frame of 2022 Cotton Field data with the ZED2 camera from August 26, 2022.



Figure 4.10: An example of the right frame of 2022 Cotton Field data with the ZEDM camera.

We first investigated the impact of downsampling on a smaller scale of 2022 data in the context of the pipeline. Prior to downsampling, the dataset had a size of 41.6 MB, a shape of 1434 x 802, and a pipeline execution time of 2 minutes and 22 seconds. After implementing downsampling as a pre-processing step, the size of the 2022 dataset was reduced to 4.8 MB and its shape became 717 x 401. The pipeline execution time also decreased to 2 minutes and 7 seconds. This led to a percent decrease of 88.46% for size, and a percent decrease of 10% for pipeline execution time. Importantly, despite downsampling, the neural network model maintained its accuracy and effectively detected cotton blooms in the dataset. A visual illustration of the pipeline output for the 2022 dataset is presented in Figure 4.13. These findings highlight the efficacy of downsampling in reducing computational time and storage requirements without sacrificing model performance.

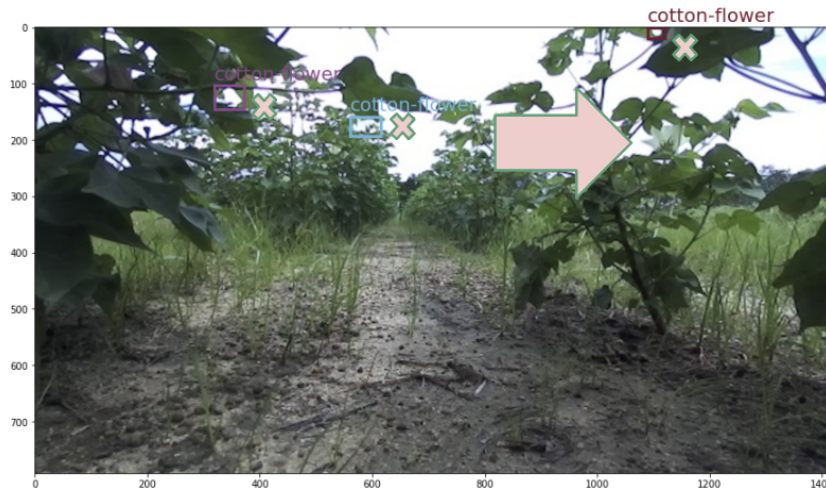


Figure 4.11: A visualization of results of cotton bloom detection using the ZEDM camera on the right frame of the 2022 Cotton Field data are shown. The output displays only the bounding boxes. We added a red arrow added for visual clarity to indicate false negatives, and a red X was added to highlight false positives.

### 4.3.3 Masking

While downsampling showed some promising results for 2022 and future data, we aimed to explore a more optimized approach for compression. Given that the objective of this study is to detect cotton blooms while preserving their location, maintaining high image quality in the surrounding non-bloom background leaf regions is not essential. To address this, we propose incorporating masking, which prioritizes the preservation of white-colored pixel regions corresponding to cotton blooms, while masking out non-white areas in the images. This approach aims to improve the compression efficiency of the cotton field dataset while still retaining the relevant information for cotton bloom detection. We recognized the importance of determining the appropriate extent to which non-white regions should be preserved when applying the masking method. Therefore, we conducted experiments to evaluate the impact of different levels of preservation of non-white pixels surrounding the white binary threshold. Specifically, we tested the masking approach in conjunction with our model's accuracy in detecting cotton blooms.

When utilizing a combined approach of masked addition and downsampling on a subset of data initially measuring 2.03MB, while preserving only 50 non-white pixels, a notable reduction in data size was observed. The resulting compressed size was reduced to 49.53KB, corresponding to a decrease in byte size of 97.65%. However, the model accuracy decreased, indicating that this threshold was too low for reliable cotton bloom detection. Refer to figure 4.14 to see an example of the model output after pipeline execution on the original pre-compressed image from Figure 4.8 This suggests that an approach, where a larger number of non-white pixels are preserved, may be necessary to ensure the accuracy and reliability



Figure 4.12: An illustration of the cotton bloom detection output from the YOLO model after the pipeline execution of a sliced 2021 downsampled ZED2 image. The output displays only the bounding boxes. We added a red arrow added for visual clarity to indicate false negatives

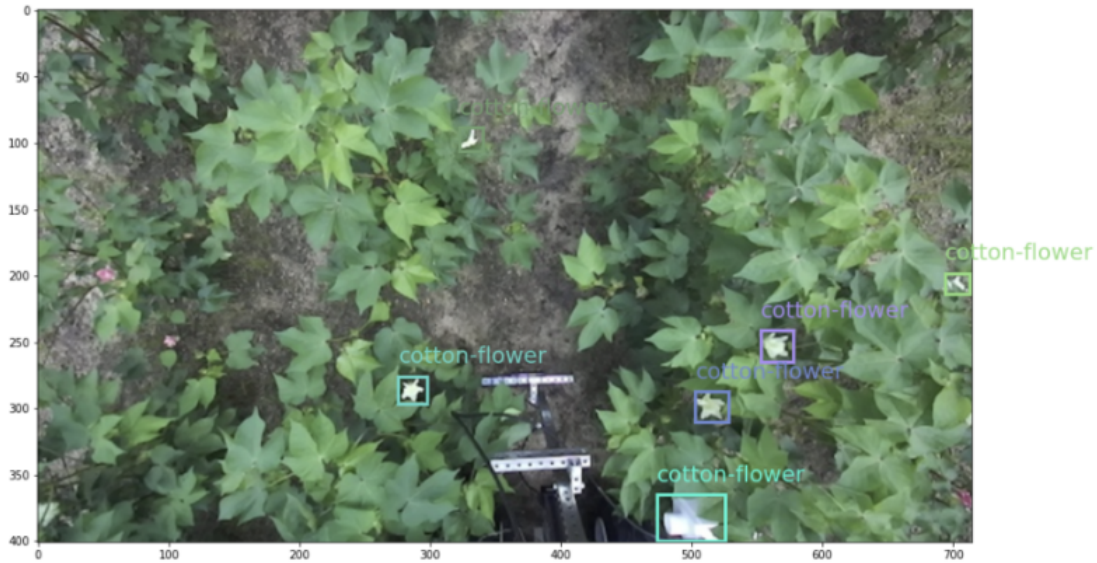


Figure 4.13: A visual of the output of YOLOv5 model on 4.8 image data after downsampling.

of the cotton bloom detection results. It highlights the importance of carefully selecting the optimal threshold for non-white region preservation in the masking method, as it directly affects the performance of the detection model. We opted to change the parameter to preserve 150 non-white regions around the white detected pixels. When testing this parameter on the original 2.03MB subset of data, in combination with masking and downsampling, we observed a significant reduction of 163.3 KB, corresponding to a 91.86% decrease in data size. Figure 4.15 illustrates an example of the output from our pipeline, showcasing the additional use of the 150 non-white region masking method. However, when preserving only 50 non-white regions, the reduction in data size was even higher at 97%. Despite the lower reduction achieved with the preservation of 150 non-white regions, the model accuracy remained unchanged from the original, and it accurately detected blooms in the same manner. We emphasize not only the reduction in data size but also the maintenance of model accuracy when executing the pipeline.

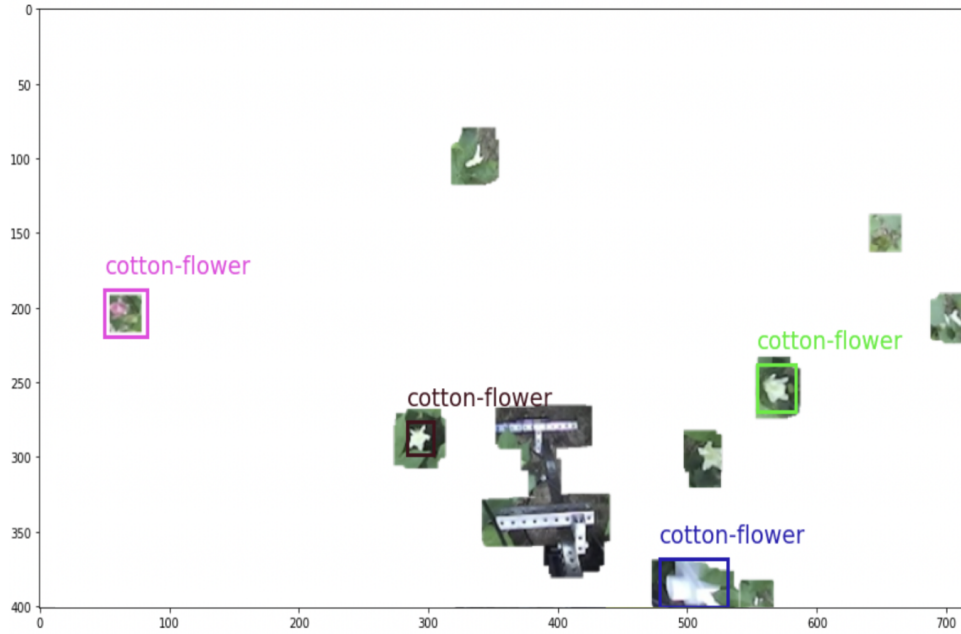


Figure 4.14: An example of pipeline output on 2022 Data with Masking and Downsampling, Preserving 50 Non-White Regions around Binary Classified White Region.

#### 4.3.4 Scaling up and Running Pipeline for 2022

In order to validate the scalability and effectiveness of our compression technique, we conducted experiments on a larger dataset comprising of 1,856 images from the year 2022. These images had a combined size of 242.6 MB and were extracted from data collected on August 4, 2022. Prior to applying our compression technique, which involves masking and downsampling, the images had a shape of 1280 x 360 pixels and the pipeline execution time was 5 minutes and 40 seconds.

Upon applying our compression technique, there were reductions in both size and execution time. The compressed dataset was reduced to 24.8 MB in size, with the shape of the images decreasing to 640 x 180 pixels. Furthermore, the pipeline execution time decreased to 4 minutes and 25 seconds. The overall results demonstrate the effectiveness of our compression technique, with a size reduction of 89.774% and a percent reduction in execution time of 22.06%. These findings highlight the scalability and efficiency of our approach when applied to a larger dataset, further validating its potential for real-world applications in scientific research and data processing. Furthermore, it is important to draw attention to how we conducted our analysis on the 2022 data, which originally amounted to approximately 250MB. The original execution time without employing any compression technique was 5 minutes and 40 seconds. In contrast, in a previous study where we tested the 2021 data, the file size was smaller, around 150MB, but the execution time prior to using any compression technique was considerably longer at 34 minutes. The



Figure 4.15: An example of pipeline output on 2022 Data with Masking and Downsampling, Preserving 50 Non-White Regions around Binary Classified White Region.

reason behind this discrepancy is that while the 2021 data had a smaller overall size, it comprised a larger number of instances or files, specifically 9000 images that required slicing. However, for the 2022 data, we had a lower number of instances, specifically 1,856 images, as slicing was not necessary to maintain model accuracy. This finding indicates that the pipeline execution time is influenced not only by the size of the data, but also by the number of files. The loading of instances during the pipeline processing phase can also impact the overall time taken. We decided to scale up our testing efforts by using a larger dataset of 17,332 images, totaling 1.06GB in size. This dataset consisted of all the data (all rows) from August 4, 2022, and 4 rows of data from August 22, 2022. Initially, without implementing any compression technique, the shape of the images was 530 X 144, and the total pipeline execution time was 63 minutes and 19 seconds.

However, we then applied a combination of masking and downsampling for compression, which reduced the file size to 178.1MB and the shape to 260 X 72. When we performed the image pre-processing using the compressed images synchronously, it took 4 minutes and 14 seconds. But, when we optimized the code using asynchronous coding, where operations are run concurrently, the pre-processing time decreased to 3 minutes and 18 seconds. The total time, including the pipeline execution time and the additional pre-processing compression time for the 2022 data, was reduced to 54 minutes and 55 seconds.

The results showed that incorporating the compression techniques resulted in a reduction in size by 83.20% and a decrease in execution time by 13.14%. On average, it took 0.19 seconds per image to go through the entire pipeline process, including compression pre-processing (masking and downsampling), data ingestion into the data lake, sending to Kubernetes for offline model processing, retrieving the output (bounding box coordinates) from the model, conducting post-processing to draw the bounding box visuals, and saving the output file back into the storage container.

## 4.4 Discussion and Future Work

Our research findings have significant practical implications for image processing pipelines in various domains, including computer vision, machine learning, and data analytics. The incorporation of compression techniques has been shown to yield substantial improvements in both file size and execution time, enabling more efficient and scalable processing of large image datasets.

For future work, addressing the challenges identified in this study, further data collection from the ZEDM camera could be pursued to fine-tune the model for accurate detection of cotton blooms in images captured from this camera. Additional data from the lower angle perspective of the ZEDM camera could enable the model to better learn and generalize to the unique characteristics of cotton blooms in images obtained from this camera. Once sufficient data is acquired, the model could be trained using transfer learning or other advanced techniques to further enhance its performance in detecting cotton blooms in images captured using the ZEDM camera. This could pave the way for even more accurate and robust detection of cotton blooms in real-world agricultural settings, with potential applications in crop monitoring, yield estimation, and decision-making for cotton farmers.

## 4.5 Conclusion

In conclusion, our research findings demonstrate the effectiveness of incorporating compression techniques, specifically masking and downsampling, in the image pre-processing pipeline for our dataset of 2022 data. The results showed a significant reduction in file size by 83.20% and a decrease in execution time by 13.14%. Additionally, by optimizing the code with asynchronous coding, we further improved the pre-processing time by 22.77%. The scalability of our approach was also demonstrated through testing on a larger dataset of 17,332 images, totaling 1.06GB in size. The average processing time of 0.19 seconds per image for the entire pipeline, including compression pre-processing, data ingestion, model processing, post-processing, and output storage, indicates the efficiency of our approach in real-world application scenarios. Moreover, our study highlights the importance of considering both the size and number of files in the pipeline execution time, as evidenced by the differences observed between the 2021 and 2022 datasets.

In summary, our research contributes to the field of image processing by demonstrating the benefits of incorporating compression techniques in the image pre-processing pipeline. Our findings have practical implications for improving the efficiency and scalability of image-processing pipelines in various applications. Further research in this area has the potential to advance the field and contribute to the development of more efficient and effective image-processing pipelines in the future.



# CHAPTER 5

## CONCLUSION

In conclusion, our study presents a big data pipeline for cotton bloom detection using a Lambda architecture and Microsoft Azure's cloud computing resources, with high performance achieved through the use of YOLOv5 neural network. Our results demonstrate a Mean Average Precision (mAP) score of 0.96 and an optimized runtime of 34 minutes when evaluated on over 9,000 images for 2021 data. Unlike previous research, our proposed pipeline incorporates both batch and real-time processing, providing an efficient and scalable solution for data analysis. Furthermore, the utilization of cloud computing resources, specifically Microsoft Azure, improves the performance and reliability of the pipeline. Our paper also provides detailed information on the implementation tools and processes used to build the proposed pipeline, facilitating reproducibility and further research.

In addition, we showcase the scalability and efficiency of our approach in handling large datasets, with potential applications in precision agriculture. The performance of our pipeline is validated on new data, demonstrating its ability to accurately detect blooms in real-time with high accuracy, which indicates its potential for future large-scale applications in the agriculture industry. We propose a promising method of compression for pipeline optimization through downsampling and masking, and the findings demonstrate the effectiveness of incorporating compression techniques in the image pre-processing pipeline for our dataset of 2022 data. There was a significant reduction in file size by 83.20 percent and a decrease in execution time by 13.14 percent. The scalability of our approach is further demonstrated through testing on a larger dataset of 17,332 images, indicating the efficiency of the approach in real-world application scenarios.

Moreover, the approach for data reduction pre-processing and cotton bloom detection using techniques such as Principal Component Analysis (PCA) presents a promising solution for improving the efficiency of cotton bloom detection in image processing. The results show a high amount of data reduction from the original to the reconstructed images, with byte sizes reducing by 93 percent through PCA while preserving around 98 percent variance with a smaller number of components. Bitwise masking with OpenCV yields a 99 percent reduction in file size, showcasing the potential of employing machine learning techniques for data reduction pre-processing. Notably, this is the first application of data reduction tools

for field-based cotton phenotyping, incorporating cross-validation and unsupervised machine learning techniques.

### **5.0.1 Future Works**

A direction for future work would be to extend the application of our scalable pipeline to other crops, thereby optimizing crop management practices. To further improve the pipeline's performance, one can explore the use of additional computing resources, such as increasing the number of cores or utilizing more intensive virtual machines and GPUs. Moreover, one can investigate the incorporation of PySpark for additional parallel processing capabilities. Another potential enhancement involves integrating MLflow, which could reduce the cost compared to using Kubernetes and save time by directly saving the model in Databricks without the need for REST API connections.

Additionally, one area to explore is the integration of our pipeline with IoT devices for seamless data collection and analysis. Since our current pipeline relies on image frames extracted from the original bagfiles produced by the IoT device, one may investigate the possibility of applying the pipeline directly to these bagfiles. Furthermore, the integration of other trained models for cotton bloom detection can be explored, moving beyond the Azure AutoML YOLOv5 model, and examining transfer and ensemble learning techniques to enhance model accuracy.

Furthermore, given the promising results achieved in pipeline optimization and compression techniques, their applicability to other image-processing tasks in precision agriculture can be studied. Future research will involve integrating advanced machine learning models, such as autoencoders, to optimize the compression and processing steps. One can also assess the tradeoffs between compression rates and processing time associated with these techniques. In our current implementation of PCA, we apply it to the separate color channels of individual images, rather than the entire dataset. In future studies, PCA can be applied to the whole dataset to improve the generalizability of eigenvectors, resulting in a more consistent and compact representation of the images. This approach will enhance storage efficiency and transmission for pipeline runs while providing a more generalizable representation.

In conclusion, our research contributes to advancing image processing and big data processing in precision agriculture, with practical implications for improving efficiency and scalability in real-world applications. By demonstrating the benefits of incorporating compression techniques in the image pre-processing pipeline and showcasing the scalability and efficiency of our approach in handling large datasets, our findings have significant implications for the field of image processing and pave the way for further advancements in precision agriculture.

# BIBLIOGRAPHY

- [1] K. Fue, W. Porter, E. Barnes, C. Li, and G. Rains, “Center-articulated hydrostatic cotton harvesting rover using visual-servoing control and a finite state machine,” *Electronics*, vol. 9, no. 8, p. 1226, 2020.
- [2] F. Perez-Sanz, P. J. Navarro, and M. Egea-Cortines, “Plant phenomics: an overview of image acquisition technologies and image data analysis algorithms,” *GigaScience*, vol. 6, no. 11, 2017.
- [3] Y. Jiang and L. Changying, “Convolutional neural networks for image-based highthroughput plant phenotyping: A review,” 2020.
- [4] “Ai-based modeling and data-driven evaluation for smart farming-oriented big data architecture using iot with energy harvesting capabilities,” *Sustainable Energy Technologies and Assessments*, vol. 52, p. 102093, 2022.
- [5] F. Perez-Sanz, P. J. Navarro, and M. Egea-Cortines, “An Optimized Kappa Architecture for IoT Data Management in Smart Farming,”
- [6] N. Sanla, “A comparative performance of real-time big data analytic architectures,” *IEEE*, 2020.
- [7] J. K. Gilbertson and A. Van Niekerk, “Value of dimensionality reduction for crop differentiation with multi-temporal imagery and machine learning,” *Computers and Electronics in Agriculture*, vol. 142, pp. 50–58, 2017.
- [8] FAO, “Global agriculture towards 2050,” in *Food and Agriculture Organization of the United Nations, High level expert forum*, 2009.
- [9] UGA, “Georgia cotton production guide,” UGA Extension Team, Tifton, GA, 2019.
- [10] D. Antille, J. Bennett, and T. Jensen, “Soil compaction and controlled traffic considerations in australian cotton-farming systems,” *Crop and Pasture Science*, vol. 67, pp. 1–28, 01 2016.
- [11] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja, “Lambda architecture for cost-effective batch and speed big data processing,” in *2015 IEEE International Conference on Big Data (Big Data)*, pp. 2785–2792, 2015.
- [12] N. Marz and J. Warren, *Big data: principles and best practices of scalable real-time data systems*. Manning, 2013.

- [13] A. Roukha, F. Fote, S. Mahmoudi, and S. Mahmoudi, "Big data processing architecture for smart farming," *Procedia Computer Science*, vol. 177, pp. 78–85, 2020.
- [14] E. Ouafiq, R. Saadane, A. Chehri, and S. Jeon, "Ai-based modeling and data-driven evaluation for smart farming-oriented big data architecture using iot with energy harvesting capabilities," vol. 52, 2022.
- [15] S. Rawar and A. Narain, "Understanding azure data factory: Operationalizing big data and advanced analytics solutions," 2018.
- [16] F. Kadedge, R. Glen, and P. Wesley, "Real-time 3-d measurement of cotton boll positions using machine vision under field conditions," 2018.
- [17] V. Thesma, C. Mwitwa, J. M. Velni, and G. Rains, "Spatio-temporal mapping of cotton blooms appearance using deep learning," vol. 55, no. 32, pp. 36–41, 2022.
- [18] S. Wolfert, L. Ge, C. Verdouw, and M.-J. Bogaardt, "Big data analytics in smart farming-a review," *Agricultural Systems*, vol. 153, pp. 69–80, 2017.
- [19] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw Appl*, vol. 19, no. 171, p. 209, 2014.
- [20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A faulttolerant abstraction for in-memory cluster computing," *Proc. 9th USENIX Conf. Networked Systems Design and Implementation*.
- [21] J. E. Gonzalez, Y. Low, and T. Boulos, "Mapreduce for data intensive scientific analysis," *Future Generation Computer Systems*, vol. 36, pp. 17–27, 2014.
- [22] I. Databricks, "Databricks - the unified analytics platform." <https://databricks.com/>, 2021.
- [23] M. Corporation, "Azure kubernetes service (aks)." <https://azure.microsoft.com/en-us/services/kubernetes-service/>, 2021.
- [24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.
- [25] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.
- [26] M. Wachs and B. Kalyanasundaram, "Automl: An overview," *arXiv:2102.12064*, 2021.
- [27] K. Murphy, "Machine learning: A probabilistic perspective," *MIT Press*, 2012.
- [28] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

- [29] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” *PhD Thesis, University of California, Irvine*, 2000.
- [30] P. S. Foundation, “Asyncio library,” 2023.
- [31] aio libs, “aiohttp library documentation,” 2023.
- [32] aio libs, “aiofiles library documentation,” 2023.
- [33] D. R. Hidalgo, B. B. Cortés, and E. C. Bravo, “Dimensionality reduction of hyperspectral images of vegetation and crops based on self-organized maps,” *Information Processing in Agriculture*, vol. 8, no. 2, pp. 310–327, 2021.
- [34] Sabarina and Priya, “Lowering data dimensionality in big data for the benefit of precision agriculture,” 2015.
- [35] C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, S. U. Khan, *et al.*, “Big data reduction methods: a survey,” *Data Science and Engineering*, vol. 1, no. 4, pp. 265–284, 2016.
- [36] B. Nagarasu and M. Manimegalai, “Automatic irrigation and worm detection for peanut field using raspberry pi with opencv,” pp. 1–5, 2016.
- [37] J. Lever, M. Krzywinski, and N. Altman, “Principal component analysis - Nature Methods,” *Nature*, vol. 14, pp. 641–642, Jun 2017.
- [38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [39] “The loni pipeline processing environment,” *NeuroImage*, vol. 19, no. 3, pp. 1033–1048, 2003.
- [40] M. Zolnouri, X. Li, and V. Nia, “Importance of Data Loading Pipeline in Training Deep Neural Networks,”