Final Report

12/13/2024

CRAFT (Ceramic Recording Automation and classiFication Team)

Project Sponsor:

Dr. Leszek Pawlowicz

Faculty Mentor:

Vahid Nikoonejad Fard

Course Organizer:

Isaac Shaffer

Team Members:

Kimberly Allison, Aadarsha Bastola, Alan Hakala, Nicholas Wiley

Table of Contents

1. Introduction……..3
2. Process Overview...3
3. Requirements..5
4. Architecture and Implementation..6
5. Testing...12
6. Project Timeline...13
7. Future Work..14
8. Conclusion..14
9. Appendix 15
Appendix A: Development Environment and Toolchain..15

1. Introduction

Ancient pieces of ceramic vessels, or sherds, are still being discovered throughout the southwest. Anthropologists use these sherd's characteristics to attribute the piece to a specific region and period. However, discrepancies in these determinations are incredibly common, with anthropologists disputing up to 40% of classifications. The process of cataloging and classifying sherds is time-consuming, costly in resources, inconsistent, and requires extensive knowledge of specific areas and communities.

Our sponsor, Dr. Pawlowicz, is a veteran archeologist and Research Professor at NAU's Department of Anthropology in Flagstaff, AZ. Dr. Pawlowicz recognized a need for a consistent system for sherd classification, leading him to personally research and develop a Convolutional Neural Network (CNN) capable of accurately and reliably identifying sherds from a photo. When implemented into a simple mobile application, the model became an accessible, reliable tool for professionals and hobbyists. However, the model was restrained by Dr. Pawlowicz's personal computer, resulting in slow training cycles and the use of an outdated model. Alongside team CRAFT, he aims to create a refined CNN and make it accessible to the archeological community.

With an outdated model and a lack of fine-tuned hyperparameters, the CNN has not reached the level of accuracy and reliability needed for application in the field. The refined model will be implemented into a mobile application and used with a conveyor belt system. A user-friendly, intuitive application will be developed to allow anthropologists in the field to utilize the power of CNN from a mobile phone. To classify a sherd, a user only needs to take or upload a photo into the application. The simplicity of the process will allow for quick cataloging, require fewer resources, and allow for the immediate return of sherds to an archeological site. Implementing the model into the conveyor belt system will allow anthropologists to quickly catalog mass amounts of sherds. A camera situated over a conveyor belt will take photographs of sherds as they pass underneath, continuously saving and classifying photos.

2. Process Overview

Prior to the start of our development, our sponsor had made some initial attempts to develop the areas of our project (deep learning model, conveyor belt program, mobile app) but all pieces were in either incomplete or very rough state. Our development process started with examining the starting point that had been provided to us and deciding what to keep, and what to not keep. Since our capstone is clearly divided into three different products, we will discuss the process overview for each product respectively.

2.1 Mobile App

The CRAFT mobile app is a robust and versatile tool designed to classify images using an advanced Image Classification Model, offering seamless functionality both online and offline. Users can input images via the camera or local storage, and the app delivers classification results with associated confidence levels. It ensures offline accessibility by performing classifications without an active internet connection, storing the results in a data buffer, and automatically uploading them to a database once connectivity is restored. The app includes location services to log the geolocation of classified images, with the ability for users to edit this information if necessary. Additionally, the app supports editing classification results, enabling users to correct any inaccuracies. With an intuitive feedback mechanism and reliable offline capabilities, CRAFT empowers users to manage image classifications and associated data efficiently, ensuring flexibility and precision in all environments.

2.2 Conveyor Program

The first stage of the conveyor's development lifecycle starts with the work our sponsor had developed on the program up until he handed it off to us. In the case of the conveyor belt a complete alpha had not been developed. The work that had been done up until this point amounted to a python program which used OpenCV to remove the background from an object detected in an image. While no code was reusable from this initial program it presented us with the basic technologies we would use throughout the development process. These being Python and the OpenCV library for Python.

From here the second phase of the lifecycle began where methods for the program's most basic features were definitively decided. Here the basic computer vision tools from OpenCV were tested. Once the program was capable of basic motion tracking, object detection, the ability to register when objects pass through certain locations, and the ability to take consistent centered photos without repeats. Since this phase of development saw much more regular updates and changes it was at this point that version control was considered. For version control we used GitHub to ensure our code base's stability.

Once these basic features were tested and definitively developed, they could be enhanced further to approach the program's final state. These enhancements include removing the background from a detected object and replacing it with a white background. While this task is straightforward for images applying this consistently to a video feed with a constant stream of moving objects proved to be difficult. Establishing a solid general method for this and for other higher-level details took quite some time and took up defined this part of the development life cycle.

In the last stage of development, the program was made much more consistent and accurate. Early in the beginning of the second semester development was focused on establishing the previously described baseline functionality. In this phase we sought to significantly refine the program. Before over half of images captured had some level of artificing (by artifacts here we mean spots of the background unintentionally not masked and shown in the image alongside the sherd), by the end of this phase this problem was eliminated. In this phase we also transitioned the rough, less user friendly CLI program used with a GUI using PySide6's QtCore functions.

2.3 Deep Learning Model

Our team's deep learning model's lifecycle began with research. Since much of the team had little to no experience with AI in general, we needed to learn the foundations. The most important piece of literature in our research was "Deep Learning with Python" which provided us with the knowledge to understand the inner workings of deep learning models.

Once we had sufficient knowledge, experiments on different architectures began to determine which would be the most accurate. The four main contenders were ResNet, ConvNeXt, Swin Transformer, and a custom CNN network utilizing residual blocks. Most of the last semester of capstone was spent building models for each of the four architectures to test and see which would give us the best accuracy. It was determined that ConvNeXt will give us the best accuracy and that is the model we ended up going with.

3. Requirements

3.1.1 Mobile App

1. **Image Classification**
   The application should be able to classify the images provided by the user via camera/local storage using an Image Classification Model.
2. **Offline Functionality**
   The application should be able to perform image classification without an active internet connection. The application should be able to perform all the functions except uploading data to the Database without an internet connection.
3. **Save Results in a Database**
   The application should be able to save the classified results into a Database.
4. **Location Services**
   The application should be able to record the obscured geolocation of the device where the classification is performed to record where the sherd was found, the user should be able to edit the location if the found location and classified locations are different.
5. **Edit Classification**
   The application should be able to edit the classification if a user believes that the classification results produced by the Classification Model are incorrect.
6. **Feedback Mechanism**
   Once the model has classified the provided image, the application should be able to show the results with respective confidence levels.

3.1.2 Conveyor Belt

1. **Image Classification**
   The application should be able to use an image classification model to classify sherds that are passing through a conveyor belt.
2. **Bulk Image Processing**
   The application should be able to process multiple pieces of sherds passing through a conveyor belt.
3. **Real-Time Processing**
   The application should be able to process images of sherds passing through the conveyor belt in real time to keep up with objects moving in the conveyor belt continuously.
4. **Bulk Data Output**
   The application should be able to output all the results of sherds classified in one session into a CSV file.
5. **Image Pre Processing**
   The application should be able to pre-process the images before classification to ensure improved accuracy and consistency.

3.1.3 Deep Learning Model

1. **Improved Accuracy**
   The deep learning model needs to be more accurate than the model given to us by the client.

4. Architecture and Implementation

4.1. Mobile App

4.1.1. Architectural Diagram

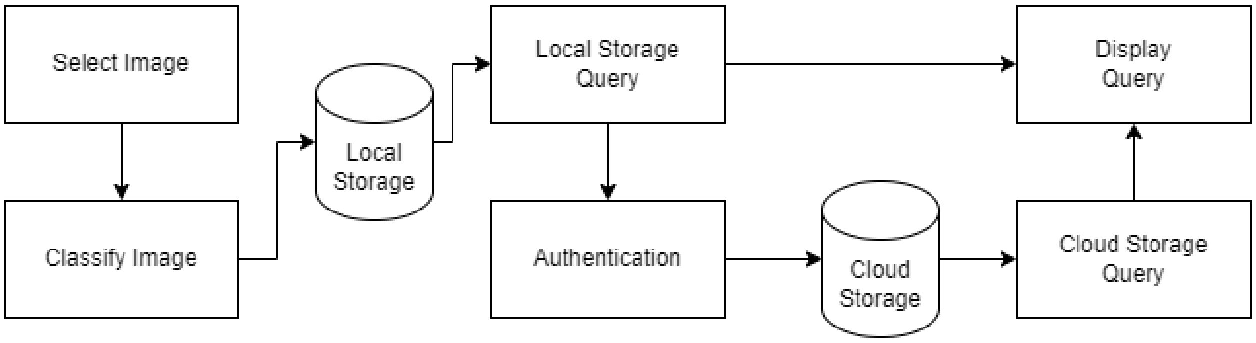


Figure 1: High-level system architecture of the mobile application.

4.1.2. Discussion of the Architecture

1. **Select Image**

   - Allows the user to choose an image from their device, or the camera.
   - Functionality to pick, crop, and resize an image according to the user's need.
2. **Classify Image**
   - Processes the selected image using our Image Classification model to predict the type of sherd.
   - Integration of a custom-trained TensorFlow model for real-time classification.
3. **Local Storage**
   - Store the selected image and classification results for offline access/caching.
4. **Authentication**
   - Manage login, logout, registration, and authentication status to write data in a cloud database.
5. **Cloud Storage**
   - Stores the selected images and their respective classification results in a Database for cloud-based sharing, archiving, and/or backup.
6. **Display Query**

Communication Mechanisms and Information/Control Flows of each component in the high-level architecture:

1. **User Interaction**
   - The user selects an image; this triggers the Classify Image module.
   - The user selects the option to log in/register, which triggers the Authentication module.
   - The user selects the option to see classification history, which triggers the Display Query module.
2. **Data Flow**
   - The Classification module classifies the image and then saves the image and classification results locally.
   - If authentication is successful, the Cloud Storage module is triggered.
   - Classification results are displayed to the user.
3. **Control Flow**
   - The app's logic determines the available sequence of actions based on user input, authentication status, and availability of data.

Architectural styles and influences from the architectural styles embodied by the high-level architecture:

1. **Layered Architecture:**
   This architectural style is seen in the separation of architectural components. The UI layer interacts with the application layer for data processing, storage, and authentication.
   - **Influence:** Promotes modularity, maintainability, and scalability.

2. **Client-Server Architecture:**
   The Client, which is the mobile application, communicates with the Server, which is Firebase, for authentication, cloud storage, and database.
   - **Influence:** Enables cloud-based services, remote data access, and archiving
3. **Event-Driven Architecture**
   Use of event-driven mechanisms (like callbacks, streams, etc.) to manage interaction, data, and control flow.
   - **Influence:** Improves responsiveness and efficiently handles architectural control flow.

4.2. Conveyor Belt

4.2.1. Architectural Diagram:

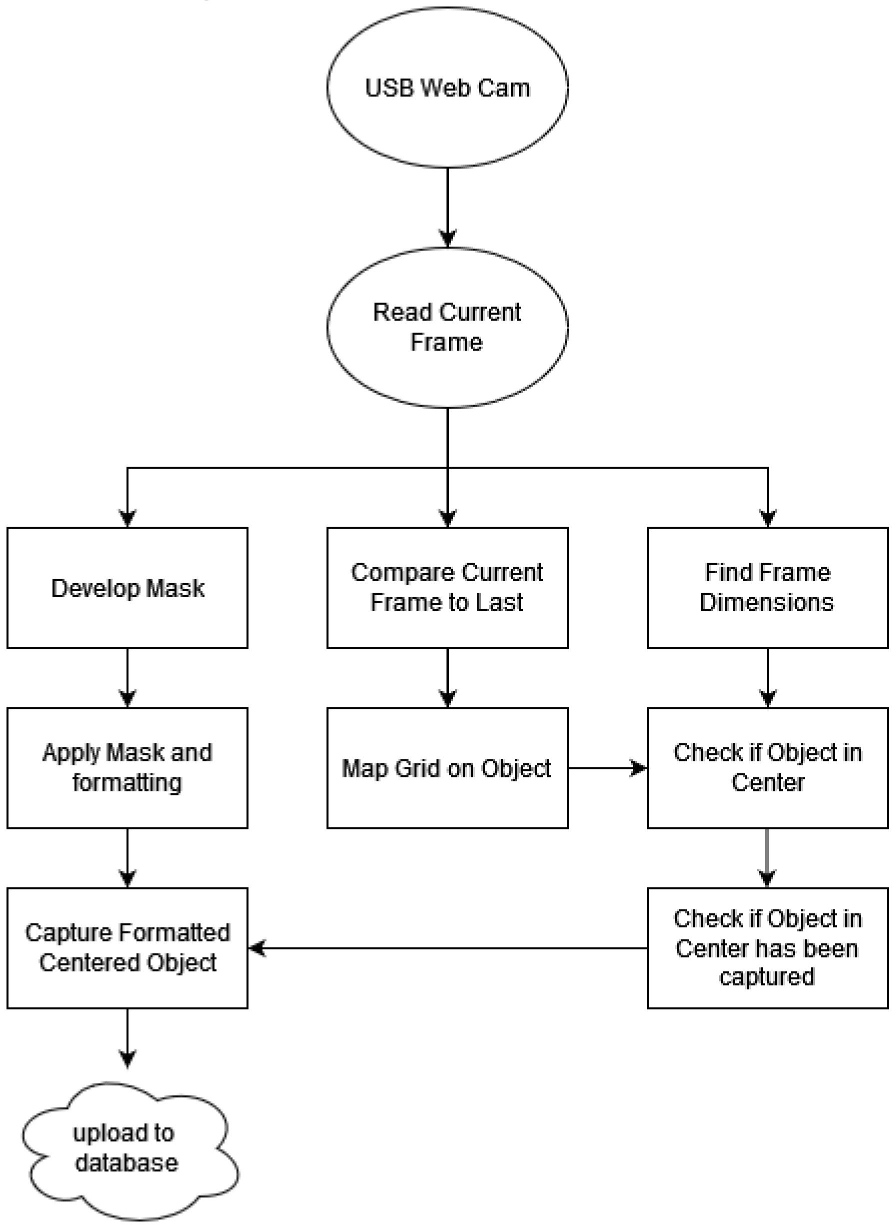

USB Web Cam
Read Current Frame
Develop Mask
Compare Current Frame to Last
Find Frame Dimensions
Apply Mask and formatting
Map Grid on Object
Check if Object in Center
Capture Formatted Centered Object
Check if Object in Center has been captured
upload to database

Figure 2 : High-level system architecture of the conveyor belt software.

4.2.2. Discussion of the Architecture:

The architecture of this program has been structured around three tasks: one that formats a given frame in the image classifier’s expected format, one that handles tracking objects and motion, and another that handles taking good images. Although there is a division of labor, all tasks collaborate to contribute to the final product. This can be seen in the forked nature of the diagram above.

Responsibilities of the high-level steps mapped in the diagram above:

1. **Read Current Frame:**
   - The raw camera footage is read as individual frames.
2. **Develop Mask:**
   - A mask is developed over the frame, it covers everything but an object.
3. **Apply Mask and Format:**
   - This gives us a different frame that is formatted properly for our image classifier. When we are ready to take a picture, we need only grab this frame to screenshot.
4. **Compare Current Frame to Last:**
   - If we find a difference between frames, this suggests something being recorded has moved.
5. **Map Grid on Object:**
   - We roughly map a rectangle grid around our object so we can easily track its position.
6. **Check if the Object is in the center:**
   - Check if we have something to screenshot.
7. **Check if the Object in the Center has been captured:**
   - Check if we have not already screenshotted this object.
8. **Capture Formatted Centered Object:**
   - Now that our object is in our center, we grab the formatted frame we got in step Apply Mask and Format.
9. **Upload to Database:**
   - Upload centered, formatted sherd image to the database.

Communication Mechanisms and Information/Control Flows of each component in the high-level architecture:

1. **User Interaction**
   - The user presses the start button on the conveyor belt system.
   - The user ends the conveyor belt program when their sherds have all passed.
2. **Data Flow**
   - A constant stream of data flows from the webcam.
   - The data stream is read in as individual frames.
   - Frames flow to three tasks; each task does something different with its frame.
   - If in the center, the formatted frame of the object is captured and stored, ending the data flow.
3. **Control Flow**

- Three tasks occur at once in the conveyor belt program, and the task that manages capturing images dictates the control flow. The program will remain in its same state of execution until the image capture task registers a new object centered in view. Upon registering a new centered object, the control flow changes from the uneventful loop to a new flow where:
- An image is captured.
- The program records that it has already photographed the object in the center.
- This continues until the object is out of the center.

After which the normal control flow continues.

Architectural styles and influences from the architectural styles embodied by the high-level architecture:

1. **Concurrency:**
   As mentioned previously, three general concurrent processes are running within this program. This is especially useful when our data comes in a constant stream. It would be tedious to have each process halted, only starting them when we determine a given frame(s) may need it.
   - **Influence:** concurrency, parallel programming.
2. **Layered Architecture:**
   Some aspects of this program's architecture are layered. For example, the image capture actions all exist on top of the base layer of motion detection. While image capture actions need motion detection, motion detection is enough to handle on its own. This is why the image capture tasks are designed to be layered on top of the motion detection.
   - **Influence:** Promotes modularity, maintainability, and scalability.
3. **Event-Driven Architecture:**
   Actions like taking a new image, archiving our data, registering an object, etc... are determined by observed events from our video feed.
   - **Influence:** The program itself waits for events to drive its actions. This type of architecture was a given for a program like this. Easy-to-map flow of control. Simpler debugging. Improved Responsiveness.

Over the course of the development of the conveyor belt program several things devitated from our initial plans. While the program's core functionality followed our planned progression, as it was developed and planned iteratively, we did not initially anticipate how we would handle user interaction with the program. While our program was initially planned to be operated via simple command line inputs as the scope grew our program became increasingly more difficult to operate via the command line. We solved this issue and deviated from our original plans by implementing a GUI with PySide's QtCore functions.

4.3. Deep Learning Model

4.3.1. Discussion of the Architecture:

In this section, we will discuss the Training module and its responsibilities for the image classification model.

Responsibilities of each component in the high-level architecture:

1. **Training**
    - Train the deep learning model using supplied training data.
    - Evaluate the model using supplied testing data.

Communication Mechanisms and Information/Control Flows of each component in the high-level architecture:

1. **User Interaction**
    - User starts the program and begins training the model.
2. **Data Flow**
    - Images are loaded into memory and moved into datasets.
    - The model is created and stored in memory.
    - Training data is fed into the model throughout the training process.
3. **Control Flow**
    - Program moves from loading data to creating the model to training the model and lastly to testing the model.

Due to the experimental nature of the deep learning model creation process, there was never an initial plan for how it would function. Because of this, there could not be any differences between what we planned on building and what the final product became.

5. Testing

The goal of team CRAFT's testing plan was to ensure consistent, effective test coverage for each piece of the project. Unit tests, or tests that validate the smallest possible pieces of source code, were conducted on each major project piece. Integration tests, or tests that ensure proper communication and data passage between program modules, were conducted on both the mobile application and the conveyor belt system. Usability tests, or tests that confirm the software's ability to provide an intuitive user experience, were also conducted on both the mobile application and conveyor belt system.

The implemented testing plan helped to ensure that program modules worked and interacted reliably and effectively, while also validating the software's ability to provide a seamless user experience.

5.1Mobile Application

Unit Testing

- **Tools:** flutter_test, mockito, firebase_auth_mocks
- **Focus Areas:**
  - Firebase authentication
  - TensorFlow Lite image classification
  - Data storage and archival
- Verifies individual component functionality through isolated testing

Integration Testing

- **Objectives:**
  - Validate module interactions
  - Ensure data flow between:
    - Flutter UI
    - Firebase authentication
    - TensorFlow Lite model
  - Test network reliability
  - Implement error handling

Usability Testing

- **Methods:**
  - Client testing sessions
  - Internal team testing
- **Goals:**
  - Assess interface intuitiveness
  - Verify user workflow efficiency
  - Collect user experience feedback

Key Testing Outcomes

- Robust and reliable app functionality
- Seamless component communication
- User-friendly interface
- Accurate classification and data management

The comprehensive testing approach ensures a high-quality, reliable mobile application for archaeological sherd classification.

5.2 Conveyor Belt

1. **Unit Testing**
   Unit tests were conducted on the functions that take and store images, as well as functions that performed image pre-processing.
2. **Integration Testing**

Integration testing was performed to ensure that communication between the conveyor belt UI and implemented deep learning model produced expected results.

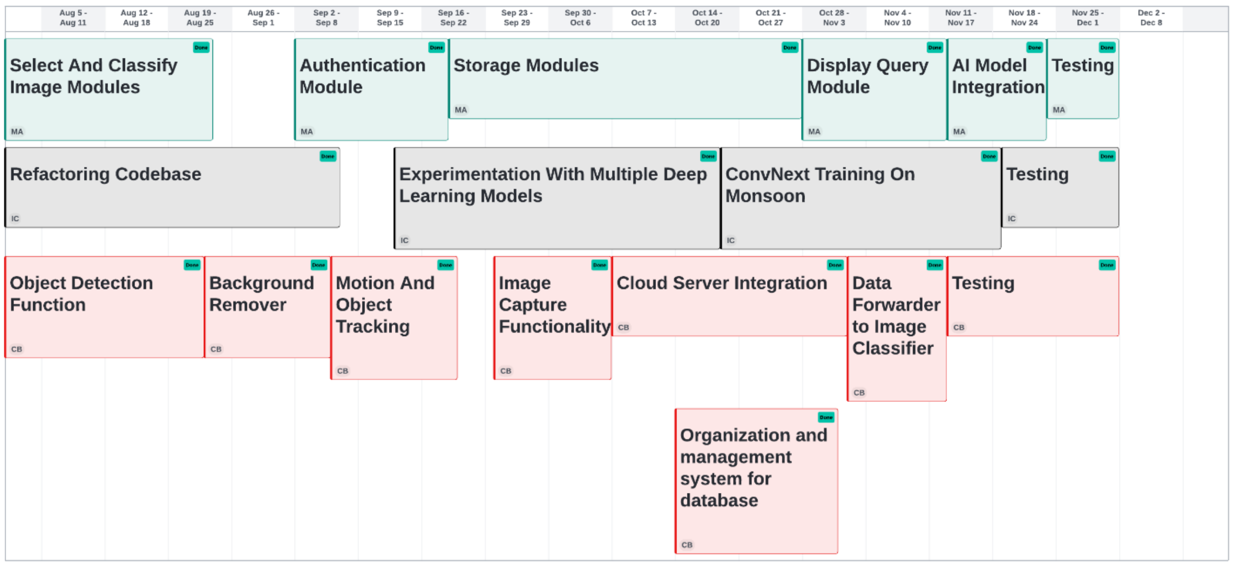
3. **Usability Testing**
   Testing the conveyor belt software with our team and a few non-technical users validated that the software was intuitive to operate.

The testing plan for the conveyor belt program yielded positive results and helped to identify several small bugs and errors. However, the main program components ran smoothly, leading the team to make no major changes to the software.

5.3 Deep Learning Model

1. **Unit Testing**
   Unit testing was conducted on input data preprocessing as well as dataset creation.
2. **Integration Testing**
   The final deep learning model was converted into a TFLite model for integration into the mobile app and conveyor belt system. Within these systems, it was verified that the image classification results were correctly displayed.

6. Project Timeline



The Gantt chart above provides a visual representation of team CRAFT's development goals and deadlines. Tasks for the mobile application, primarily led by Aadarsha Bastola, are highlighted in red, tasks for the deep learning model, worked on by Nick Wiley, are highlighted in gray, and tasks for the conveyor belt application, primarily worked on by Alan Hakala and Kimberly Allison, are highlighted in red.

Work on the deep learning model began with refactoring our client's initial codebase. Multiple deep learning models were tested before a final model was selected to be trained on Monsoon in October. Mobile app development began in early August with basic application functionality, followed by work on the authentication and database modules, and finishing with the implementation of the final deep learning model. Conveyor belt application development began with object detection and motion tracking, ending with the development of functionality that allows for the classification of taken images.

Testing for each piece of the project was underway by late November and complete by early December. As demonstrated by the Gantt chart above, project work was finalized by December 2nd.

6. Future Work

With our products successfully completed our sponsor has come up with several potential ideas to further the products we have developed. The most immediate future work we have planned with our sponsor is the further development of the deep learning model with more data, data that can now be efficiently collected with the conveyor belt program. Using these different parts of our final product together to improve the deep learning model, which both programs exist to implement in different ways, will be essential for furthering our overall product.

Our sponsor has also speculated other avenues for future work based on research he is currently conducting for NAU. Our product is built to classify Tusayan whiteware sherds, however our sponsor deals with a number of sherd categories on a daily basis. One area of further development our sponsor has identified is adapting our product to classify Little Colorado whiteware sherds.

8. Conclusion

Classifying and cataloging pieces of ancient history requires time, resources, and a trained eye. Even among experienced professionals, discrepancies in sherd classifications are incredibly common. Team CRAFT's client previously developed a deep learning model and basic mobile application to help alleviate resource strain and introduce consistency into the classification process. However, the model's accuracy rate could be improved, the application remained in an unfinished state, and sherds could not be processed in bulk.

Team CRAFT developed a deep learning model with a ~6% increase in accuracy compared to our client's previous model. The team developed a complete, accessible, cross-platform mobile application that puts the expertise of a veteran archeologist in your pocket for work in the field. Team CRAFT's conveyor belt system, capable of mass image processing and sherd classification, improves classification accuracy and consistency, and saves valuable time and resources.

Throughout the development process, our team was able to work alongside our client to devise and refine a project plan that proves just how helpful artificial intelligence can be within an archeological context. Although the project specifically identifies and classifies Tusayan White Ware, the system could be easily adapted to identify other kinds of sherds in the future. Team CRAFT is looking forward to seeing the impact of artificial intelligence and our project on both hobbyist and professional archeologists alike.

9. Appendix

A: Development Environment and Toolchain

9.1 Mobile App

Hardware:

- **Development Platform:** Windows PC, MacBook Pro (ARM)
- Testing/Debugging: Samsung A71 (Android), Apple iPhone 15 Pro Max (IOS)

Toolchain:

- VSCode (Code Editor, IDE)
- Android Studio (Android Development)
- Xcode (IOS Debugging/Build)

Setup:

- Copy the craft_v1.0.apk file to android device
- Open the .apk file and run the installer

Production Cycle:

1. Clone CRAFT Repo
2. Install Python 3.11 or a version sup
3. Set up IOS Simulator or Android Emulator if needed
4. Run from VSCode, select emulator or a connected physical device
5. Build release package using flutter release command

9.2 Conveyor Belt System

Hardware:

- **Development Platform:** Windows PC
- Testing/Debugging: Python unittesting library

Toolchain:

- VSCode (Code Editor, IDE))

Setup:

- Using the command line run the commands specified in a .txt document found in the conveyor program directory.
- Connect camera of choice to device and stage above conveyor.

Production Cycle:

1. Clone CRAFT Repo
2. Install Python 3.11 or any version supported by all the libraries specified in the conveyor's library document.
3. Run craft.py using the command line or any software of preference to start the program.

9.3 Deep Learning Model

Hardware:

- **Development Platform:** Windows Subsystem for Linux, Red Hat
- Testing/Debugging: Python unittesting library

Toolchain:

- VSCode (Code Editor, IDE)

Setup:

- Activate the appropriate anaconda environment.
- Run the command to start training the model.

Production Cycle:

1. Clone CRAFT Repo
2. Create a conda environment using the requirements.txt file found in the codebase.
3. Run the python training program from the command line.