Software Design Document

Version 1.0 – 9/27/2024

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. Implementation Overview…………………………………………...…………………...4
    2.1. Mobile App……………………………………………………………………......4
    2.2. Conveyor Belt…………………………………………………………………......4
    2.3. Image Classification Model…………………………………..……………...5
3. Architectural Overview…………………...…………………………………………….6
    3.1. Mobile App…………………………...……………………………………....6
        3.1.1. Architectural Diagram……………………………..…………………......6
        3.1.2. Discussion of Architecture…………………...…………………………......6
    3.2. Conveyor Belt……………………………...…………………………………....8
        3.2.1. Architectural Diagram………………….……………………………......8
        3.2.2. Discussion of Architecture…...…………………...……………………...8
    3.3. Image Classification Model……………...……………………………………10
        3.3.1. Discussion of Architecture……………………...……………………....10
4. Module and Interface Description……………………...……………………………..10
    4.1. Mobile App…………………………...……………………………………..10
        4.1.1. Select Image Module…………...……………………………………....10
        4.1.2. Classify Image Module…….………………...…………………………....12
        4.1.3. Local Storage Module………...……………...…………………….…....13
        4.1.4. Local Storage Query Module..……………...………………………….…14
        4.1.5. Authentication Module……………………….……………………...14
        4.1.6. Cloud Storage Module……………………......……………………..16
        4.1.7. Cloud Storage Query Module………...……………………………..….16
        4.1.8. Display Query Module…………………………………......………..17
    4.2. Conveyor Belt…………………………………………..………………..18
        4.2.1 Formatter Module……………………………………………………18
        4.2.2 Motion Detection Module…………………………………………………19
        4.2.3 Image Capture Module…………………………………………………20
    4.3. Image Classification Model……………......…………………………………21
        4.3 Training Module………………………………………………………..22
5. Implementation Plan…………………………………...……………………………...22
6. Conclusion…………………………………………...…………………………….…23

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.

For our proposed solutions to function to our sponsor's expectations team CRAFT has determined key requirements to ensure the development of an optimum solution. For each part of our final product the key requirements are as follows:

1.1. Mobile Application

- **Image Classification**

The application should be able to classify the images provided by the user via camera/local storage using an Image Classification Model.

- **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.

- **Save Results in a Database**

The application should be able to save the classified results into a Database.

- **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.

- **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.

- **Feedback Mechanism**

Once the model has classified the provided image, the application should be able to show the results with respective confidence levels.

- **Offline Data Buffer**

The application should be able to classify images. Then, if there is no active internet connection, the application should hold the classified data in a buffer until there is an active internet connection. Once the application detects an active internet connection, the application should be able to upload the results to a database.

1.2. Conveyor Belt:

- **Image Classification Automation**

The application should be able to help automate image classification by saving captured sherd images to a database. The image classifier can then be configured to use data from the database.

- **Bulk Image Processing**

The application should be able to process multiple pieces of sherds passing through a conveyor belt.

- **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.

- **Bulk Data Output**

The application should be able to output all the results of sherds classified in one session into an appropriate data file format.

- **Image Pre-Processing**

The application should be able to pre-process the images before classification to ensure improved accuracy and consistency.

1.3. Image Classification Model

- **Sherd Classification with highest practical accuracy**

The image classification model needs to accurately identify sherds from images. To do this each potential model needs to be trained on images of sherds. These images need to be modified to suit the specific model requirements. For example, CNN models need the training data to be a set resolution usually much smaller than modern resolutions or else training fails

2. Implementation Overview

Team CRAFT has been meticulously working on plans to implement the requirements drafted earlier with our sponsor Dr. Leszek Pawlowicz. Our implementation can be divided into smaller subprojects that are connected. Team CRAFT's envisioned solution is to provide an accessible and consistent baseline for sherd classification for mobile and desktop applications with the help of custom-trained deep learning models.

2.1. Mobile App

Team CRAFT's mobile app is designed with the workflow of researchers in mind, providing a seamless experience for those working in the field. The app enables researchers to utilize our deep learning model directly from their phones, even in remote locations without internet connectivity. This functionality ensures that research can continue uninterrupted, making it a valuable tool for fieldwork in various environments. The app aims to help field researchers consistently classify and properly document the sherds that they find during their fieldwork.

For the mobile app, team CRAFT decided to create a cross-platform mobile application, which will use the integrated deep learning model to consistently identify sherds regardless of connectivity to the internet. The app aims to create an intuitive cross-platform UI/UX using Flutter. It will use Firebase for all cloud functionalities, especially database and authentication. Further details about these frameworks/tools are listed below:

- **Flutter:** Flutter is a cross-platform mobile app development framework that ensures that we can cater our app to both IOS and Android users with the same codebase. This allows us to maximize our time on one codebase, ensuring that our product is robust and reliable.
- **Firebase:** Firebase is a cloud platform for Database, Cloud Storage, and Authentication. It is a robust platform that can handle a large amount of data without any performance degradation.

- **Tflite:** Tflite is one of the most important libraries in our application, as it will help us to integrate the deep learning model into our application.
- **Hive:** Hive is an encrypted lightweight NoSQL database written in Dart, made for Flutter. We will be using Hive for all local database operations.

2.2. Conveyor Belt

Team CRAFT's conveyor belt system aims to save archeologists countless hours wasted doing tedious labor. A conveyor belt, a webcam, and computer vision software together form our conveyor belt model. The conveyor belt allows archeologists to easily archive large batches of sherd data and classify them with our image classification model.

Our image classifier requires the sherd image to be grayscale and have the sherd centered. Due to this our conveyor belt software will have to format and alter images of sherds to fit these specifications. The software should also manage the archiving of newly captured sherd images. While the physical conveyor belt and webcam have been provided by our sponsor team, CRAFT is developing the software. To do so we have chosen the following technologies:

- **Python:** Python has support for several computer vision libraries, including the one we selected, OpenCV. Python's support for math and science also aids us with our software's need for calculations in image-altering methods.
- **Numpy:** Numpy will add math utility functions needed for calculations in our conveyor belt program.
- **OpenCV:** Team CRAFT has decided to utilize OpenCV to build our computer vision software. OpenCV's capabilities will allow us to properly capture images and alter them for use in our image classifier.

2.3. Image Classification Model

Our image classification model is used to give archeologists a consistent tool for classifying sherds. To provide the archeologists with the most accurate model possible, we are researching multiple different deep learning architectures to determine which is best suited for this task. The creation and testing of each of these models requires the following frameworks:

- **Keras and Tensorflow:** Tensorflow provides the foundation that Keras builds upon to create an easy-to-use deep learning framework. Keras comes with many prebuilt classes that remove the need to code fundamental neural network components from scratch.
- **Numpy:** Numpy is the most widely used Python package for performing linear algebra operations. Without it, working with Keras becomes incredibly difficult.
- **Matplotlib:** Matplotlib is a visualization tool enabling the creation of crucial graphs for deep learning research. These graphs include validation loss over time and validation accuracy over time.

3. Architectural Overview

3.1. Mobile App

3.1.1. Architectural Diagram:

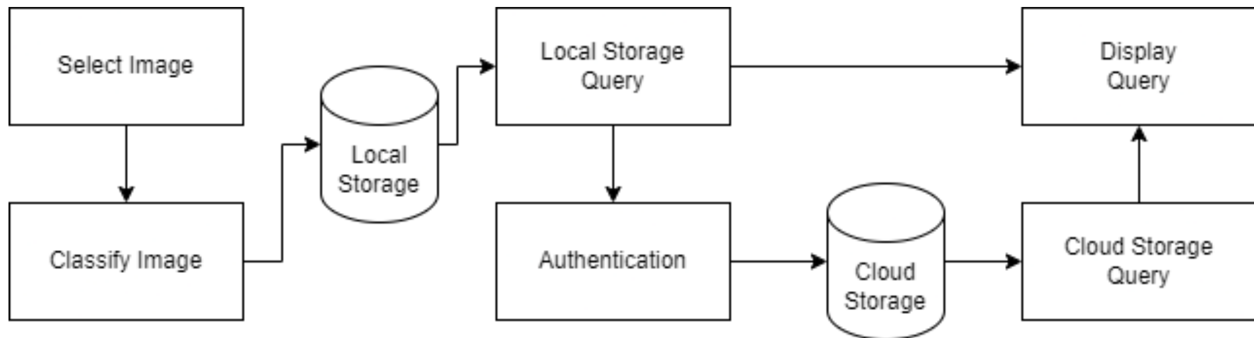


Figure 1: High-level system architecture of the mobile application.

3.1.2. Discussion of the Architecture:

In this section, we discuss the details of each component's responsibilities, communication mechanisms, and the influences of other architectural components for the mobile app.

Responsibilities of each component in the high-level 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**

   - Presents query fetched from cloud/local database to the user in an appealing format.

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.

3.2. Conveyor Belt

3.2.1. Architectural Diagram:

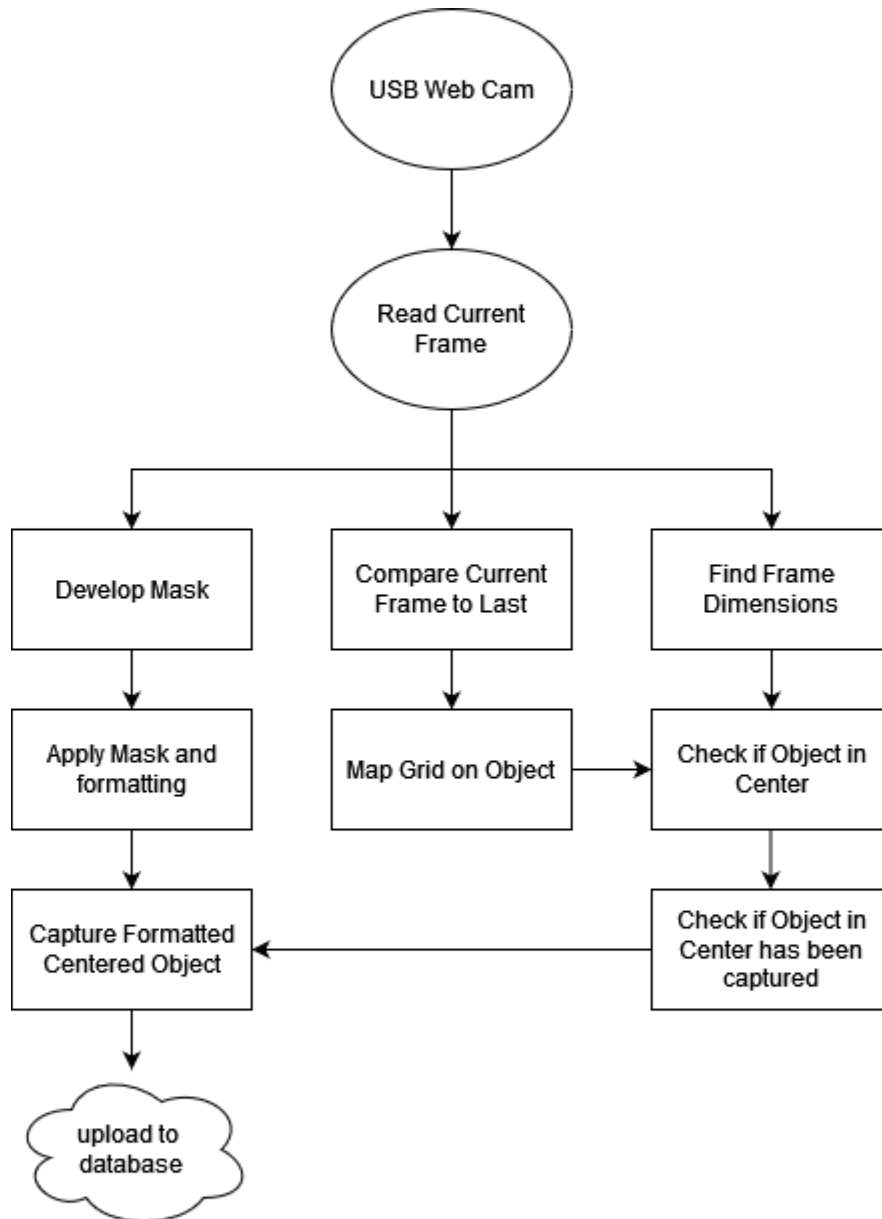


Figure 2 : High-level system architecture of the comveyor belt software.

3.2.2. 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 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.

3.3. Image Classification Model

3.3.1. 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.

4. Module and Interface Description

4.1. Mobile App

4.1.1. Select Image Module

Description

The Select Image Module is responsible for picking images from the camera or device's gallery using ImagePicker, and then checking if an image has been picked successfully. Then the image file is passed to ImageCropper, the image cropper successfully crops the Image, and the image's colors are converted to grayscale using the img.Image package. Finally, the module returns an image file. If at any time during this module image picking or cropping fails, the system executes the resetScreen function, which returns the application to its initial state.

UML Diagram

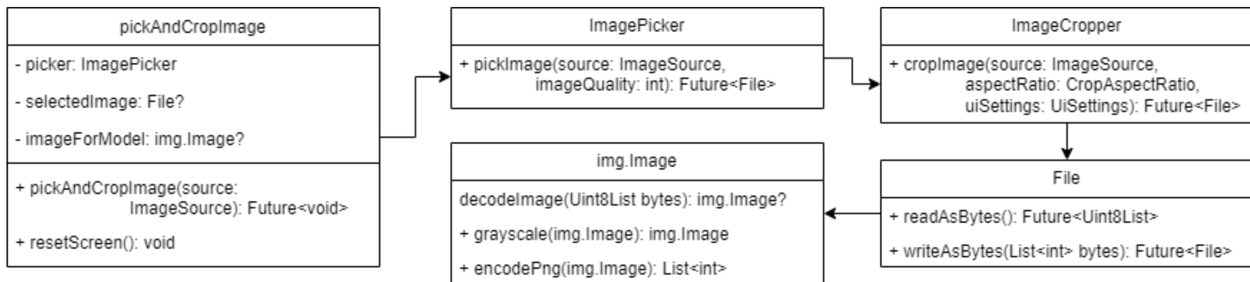


Figure 3: Select Image module UML class diagram.

Public Interface Details

1. **pickAndCropImage**
    - **Description:** This method allows the user to pick an image from their device (camera or gallery), crop the image according to their need, and convert the image to grayscale. It can also return the app to its initial state if something fails.
    - **Return Type:** Future<void>
    - **Service Provided:**
        - Pick Image

        - Crop Image
        - Convert Image to Grayscale
        - Update selectedImage and imageForModel global variables

2. **resetScreen**
    - **Description:** Clears the current state of the screen and resets it so it returns to the initial state.
    - **Return Type:** void
    - **Services Provided:**
        - Clears application state,
        - Resets the user interface to its initial state, removing the selected image

4.1.2. Classify Image Module

Description

The Classify Image Module is responsible for classifying a provided image. This module uses a TensorFlow Lite model to classify the type of sherd according to objects or patterns within the image provided to it. The classification results are stored with location data. These results are then stored in the device/cloud and then displayed in the app later.

UML Diagram

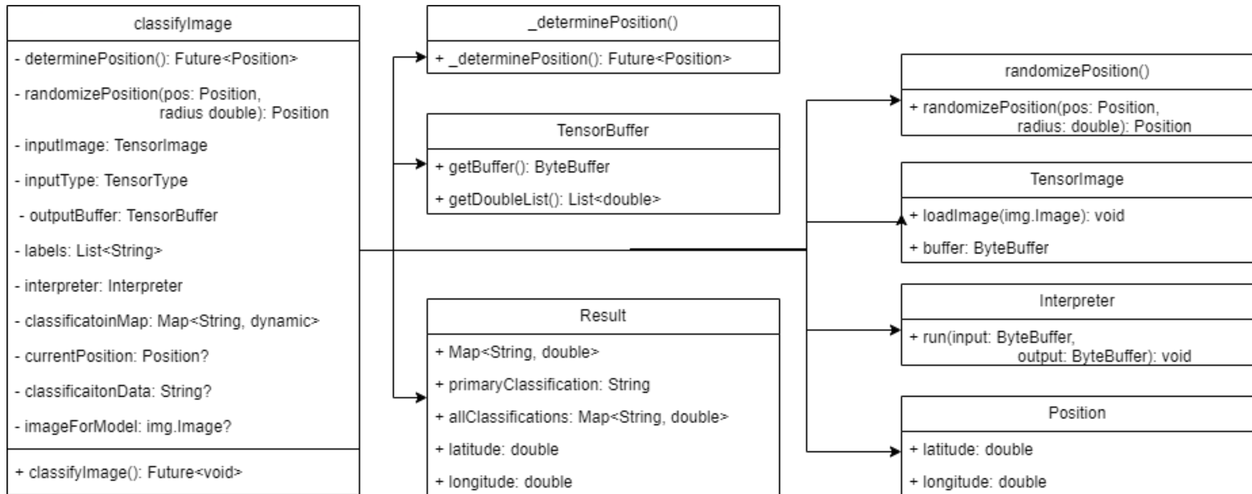


Figure 4: Classify Image Module UML class diagram.

Public Interface Details

1. **classifyImage**
    a. **Description:** This method takes the image and passes it through the Tflite model. It retrieves the classification results and location data and then updates the application state with these results.
    b. **Return Type:** Future<void>
    c. **Services Provided:**
        i. Runs the image through an image classification model for classification.
        ii. Updates classification results to the classificationResult global variable for further use.
2. **randomizePosition:**
    a. **Description**: This method takes Position data from the device's GPS and then randomizes it within a provided radius.
    b. **Return Type:** Position
    c. **Services Provided:**
        i. Randomizes the location latitude and longitude within a provided radius.

4.1.3. Local Storage Module

Description

This module is responsible for saving the results obtained from the Classify Image module into local storage. It saves all the classification results, location data, and image data into a Hive Box using a NoSQL database. Hive is a NoSQL key-value-based database, and Box is a predefined collection where data is saved. It is all stored in a persistent format so it can be retrieved even when the application is offline.

UML Diagram

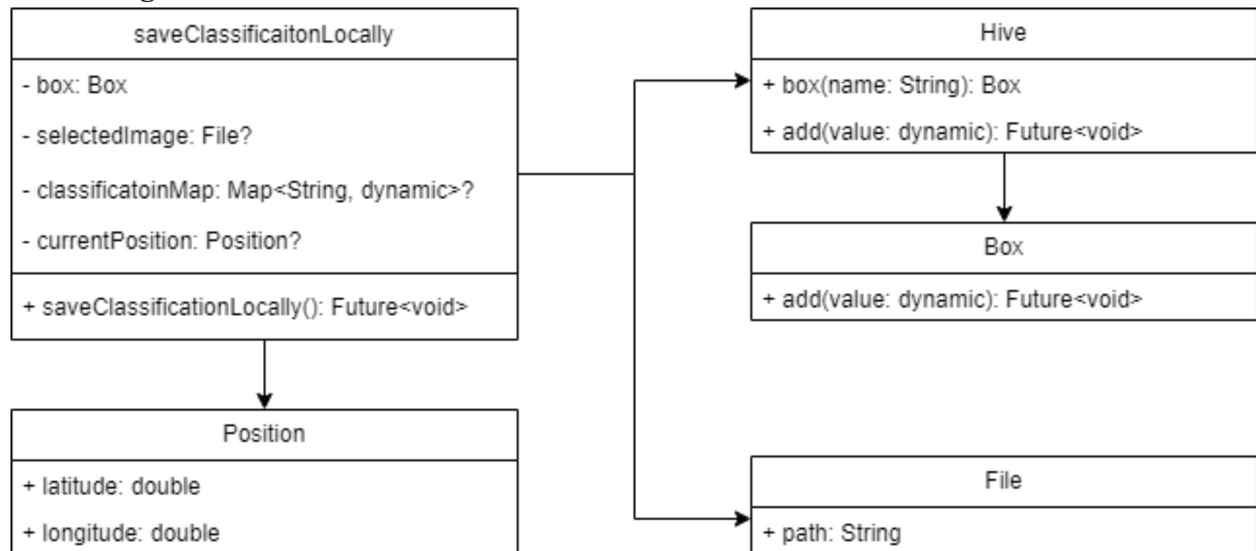


Figure 5: Local Storage module UML class diagram.

Public Interface Details

1. **saveClassificationLocally**
    a. **Description:** This method saves the classification results obtained from the previous module. The module gets data in a Map<String, dynamic> format and it saves all the data locally in the device using Hive.
    b. **Return Type:** Future<void>
    c. **Services Provided:**
        i. Stores the classification results, which include the predicted label and confidence score, location data, and image file path.
        ii. Saves data in a persistent format, ensuring results are retained even when the app is closed, or the device goes offline.

4.1.4. Local Storage Query Module

Description

The Local Storage Query Module is the module responsible for querying all the saved classification data from local storage. The data is stored in Hive Box, which retrieves data using a hardcoded box name. The module retrieves all the data saved using the Local Storage Module. The results obtained from this module update the global variable localClassifications, which can then be used to either display to the user or be uploaded to the cloud.

UML Diagram

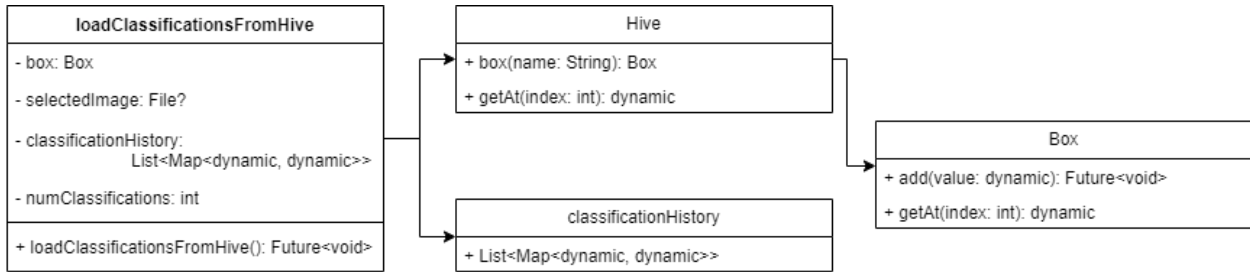


Figure 6: Local Storage Query module UML class diagram.

Public Interface Details

1. **loadClassificationsFromHive**
    a. **Description:** This method loads all the classifications that are on the Hive classifications Box, and then updates the localClassifications global variable.
    b. **Return Type: Future<Void>**
    c. **Services Provided:**
        i. Ability to load classifications from the classifications Hive Box.
        ii. Load the classification into the localClassification variable in Map<String, dynamic> format.

4.1.5. Authentication Module

Description

The Authentication Module provides user login, logout, and sign-up functionality in the application. This module uses Firebase Authentication for authentication and then uses Firebase Firestore to save all the user details including the user's name, email address, and role. The module is also responsible for notifying the global scope of the application if a user logs in or logs out. To notify the application about the state of the user authentication, we use the Provider package.

UML Diagram

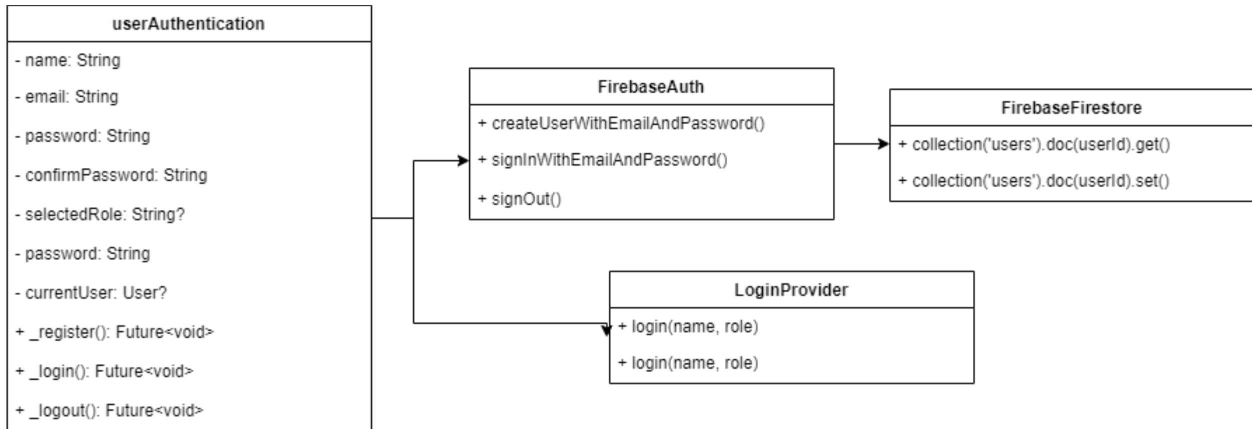


Figure 7: Authentication module UML class diagram.

Private Interface Details

1. **_register**
   a. **Description:** This method signs up a user by creating their account, it takes email, password, and role to create a user's account.
   b. **Return Type:** Future<void>
   c. **Services Provided:**
      i. Create a user's account using their email address and password.
      ii. Check email address, and confirm whether password, name, and role are valid or not
      iii. Returns error in case of error during creation of a user account
2. **_login**
   a. **Description:** This method logs in a user using their email and password.
   b. **Return Type:** Future<void>
   c. **Services Provided:**
      i. Login user using their email address and password
      ii. Checks email address and password are valid or not
      iii. Returns error in case of error during login.
3. **_logout**
   a. **Description:** Logs out user and removes user information from the current state of the app.
   b. **Return Type: Future <void>**
   c. **Services Provided:**
      i. Log out the user
      ii. Clears the user data from the current state of the app

4.1.6. Cloud Storage Module

Description

This module is responsible for saving the results obtained from the Local Storage Query module into the cloud database. This module is only accessible if the authentication module provides a user's information to the app. This module uses Firebase Firestore to store all the classification data and images of the classified sherds.

UML Diagram

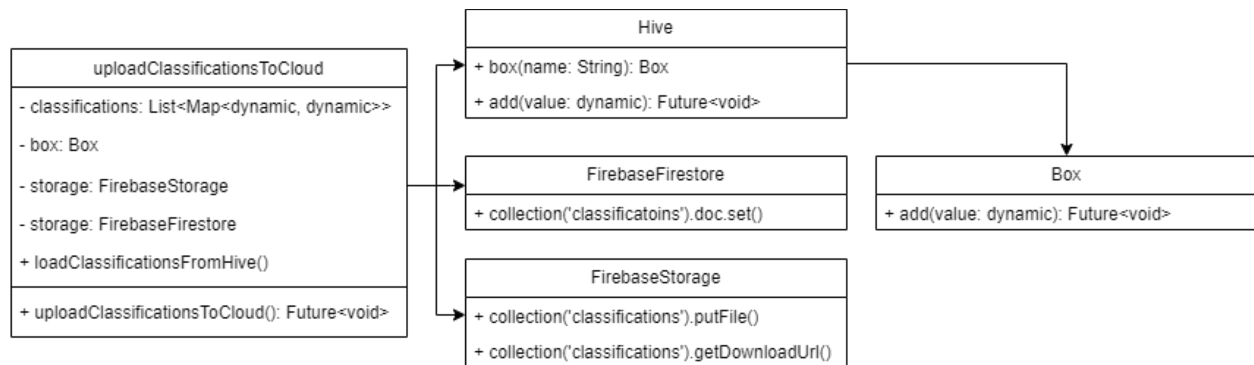


Figure 8: Cloud Storage module UML class diagram

Public Interface Details

1. **uploadClassificationsToCloud**
   a. **Description:** This method uploads all key-value classification data in Map<String, Dynamic> data type to the Firebase Firestore database, and then uploads the image of the sherd to Firebase Firestore. Once the image is uploaded, it then retrieves the link for the image file and saves it in the database for the respective sherd.
   b. **Return Type: Future<void>**
   c. **Services Provided:**
      i. Upload classification results to the Firestore database
      ii. Upload an image file associated with the classification data of a sherd

4.1.7. Cloud Storage Query Module

Description

The Cloud Storage query module is responsible for querying the classification data uploaded into the Firebase Firestore database. This module looks for all the classification data uploaded to the database by the logged-in user. The query discards data uploaded by other users and gets the queries from the logged-in user.

UML Diagram

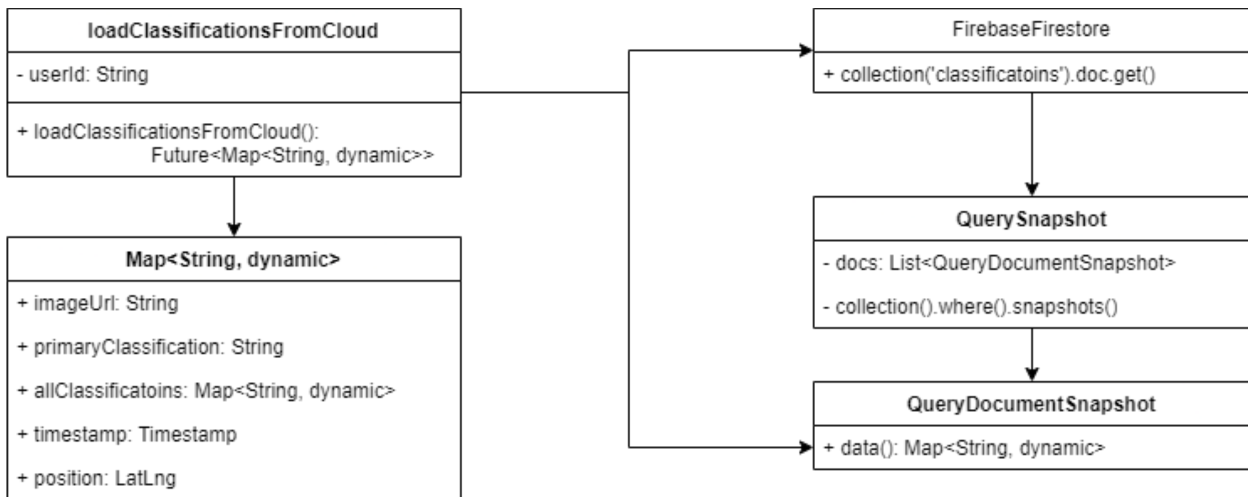


Figure 9: Local Storage Query module UML class diagram

Public Interface Details

1. **loadClassificationsFromCloud**
   a. **Description:** This interface loads all the classifications uploaded to the cloud database by the logged-in user.
   b. **Return Type:** Future<Map<String, dynamic>>
   c. **Services Provided:**
      i. Ability to load classification data from Firebase Firestore
      ii. Only load classification data uploaded by the logged-in user
      iii. Format data in dart-compatible data type i.e., Map<String, dynamic>

4.1.8. Display Query Module

Description

The Display Query Module is responsible for displaying the query from either local storage or cloud database and displaying it to the user in the app.

UML Diagram

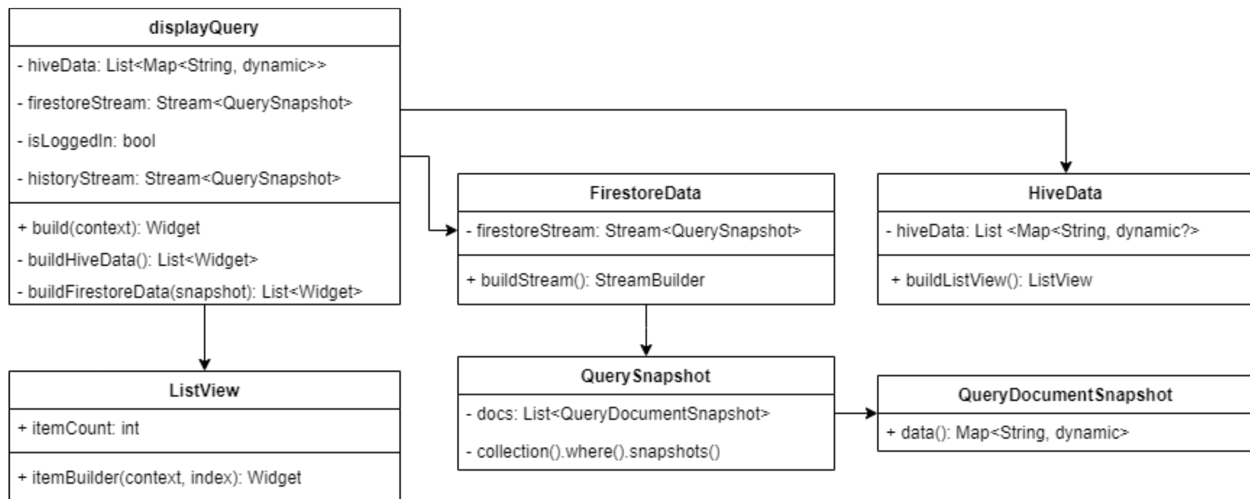


Figure 10: Display Query module UML class diagram

Public Interface Details

1. **buildHiveData**
    a. **Description:** This method is responsible for building a list of widgets based on data stored in the local storage, a Hive Box.
    b. **Return Type:** List<Widget>
    c. **Services Provided:**
        i. Loop around all the individual documents from the Hive Box and create a widget for each document.
2. **buildFireStoreData**
    a. **Description:** This method is responsible for building a list of widgets based on data stored in Firebase Firestore database.
    b. **Return Type:** List<Widget>
    c. **Services Provided:**
        i. Loop around all the individual documents from the Firebase Firestore and create a widget for each document.

4.2. Conveyor Belt

4.2.1. Formatter Module

Description

The Formatter Module is responsible for properly formatting the image of the sherd into the format our image classifier has been trained on. Specifically, our image classifier requires a black-and-white image of the sherd with the background removed and set to white. This module starts by getting the video capture of a camera it assumes is connected and ready. It then reads

the current frame of the feed which it hands over to MaskHandler. MaskHandler changes the frame from BGR to HSV and then finds the mask using a prespecified set of color ranges to filter out using getMask(). Once the mask is obtained and returned, FormatterHandler takes the mask and applies it to its current frame, then applies a grayscale to this result. Once it is finished prepping the altered frame, it is sent to class ContourFinder, which uses its method findThresh() to get the threshold of our grayscale masked frame. It then uses its method findContours() to determine all the contours detected in the frame. It returns these contours to FormatterHandler, which it uses in findObj() to identify an object and remove it from its background.

UML Diagram

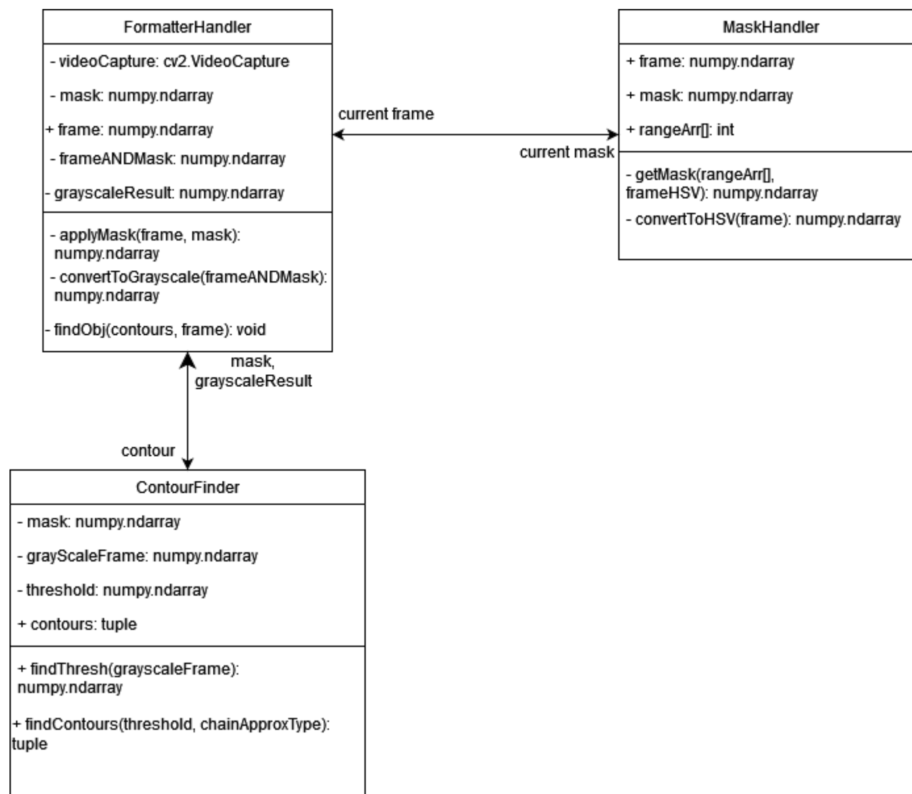


Figure 11: Image Formatter module UML class diagram

4.2.2. Motion Detection Module

Description

The Motion Detection Module handles tracking the sherd’s motion as it passes through the conveyor belt. We need to track the sherd’s motion because we want to be able to take consistent photos with it centered in view. It is also necessary to differentiate between sherds in view at the same time for image capture. Our module starts with MotionTrackPrep which reads our raw input from videoCapture as frames. In this module movement is found by comparing differences

between frames, MotionTrackPrep helps with this by recording these before/after frames. It then sends them to FramePrep, which will convert the image to grayscale, blur it using a Gaussian blur, get its threshold, and then dilate it. When comparing frames, even with no motion in between, it is common to find many insignificant differences caused by lighting, noise, camera shake, etc... .We must take this into account, which is why FramePrep performs all its steps to reduce noise and simplify the image. With the difference between the images prepped properly, we can find its contours; we will hand these to MotionTrack which will evaluate the contours to see if they suggest an object is present. If we have found an object, we map it with a rectangle. From here we simply continue the process, updating our frames each time we get a new one and check the difference for movement.

UML Diagram

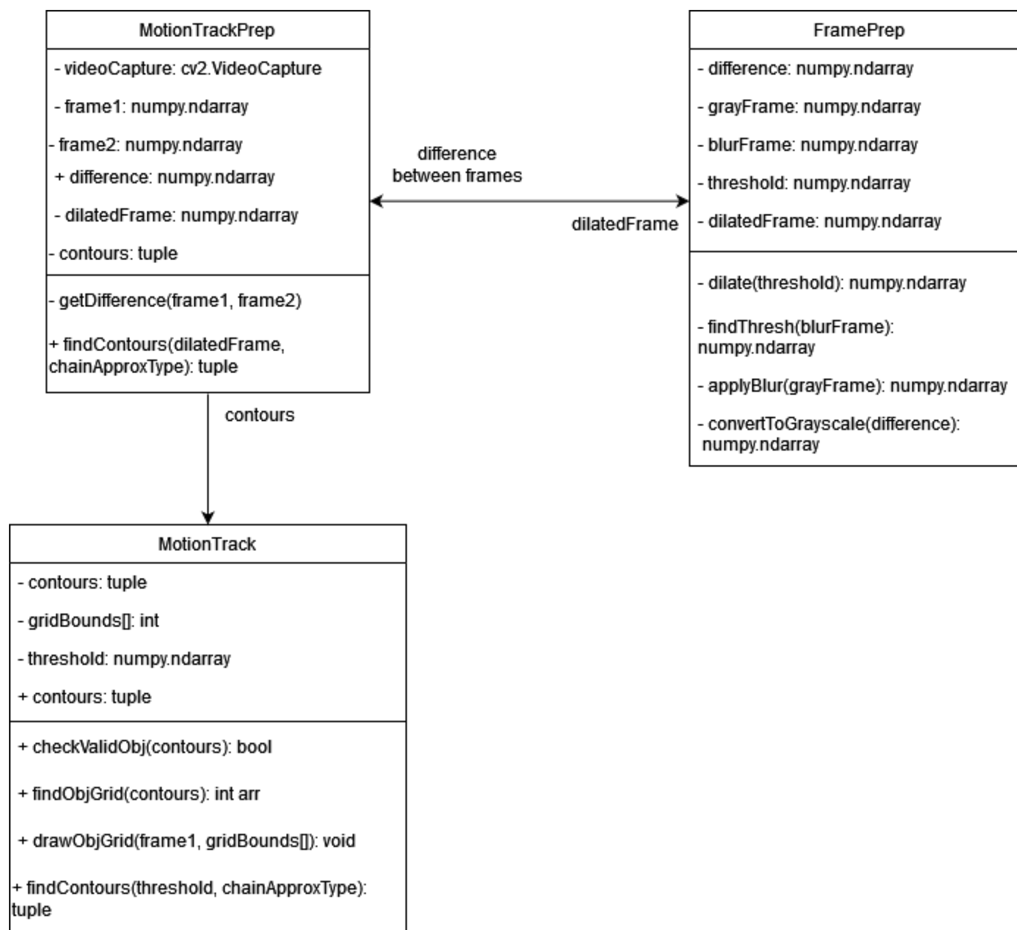


Figure 12: Motion Detection module UML class diagram.

4.2.3. Image Capture Module

Description

The Image Capture Module is responsible for taking the photos of our sherd that our image classifier will use. While the Formatter Module handles almost all formatting, there is one part it cannot handle. Our image classifier has been trained with sherds centered in their image, which is what this module will handle. This module starts by reading in the mapped grid around our object from MotionTrack. This gives us a constant stream of coordinates telling us how our object is moving, we will compare this data against what we have calculated to be the center of the screen. The precalculated center (found using frame width and height) is then expanded into a larger area with centerTolerance, an integer that will allow us to tweak how vague or precise our definition of the center is. We loop until we have found motion present in the center (checked by boolean flag objCenter), upon finding motion we need to check newObjCenter, a bool. If true, we take a photo of the frame and set the flag to false. When set to false we will know that if we detect motion in the center, it's still the same object that has yet to leave. When we see our object exit the center, we reset the flag. This gives us the ability to not only take centered photos of sherds but also ensure we do not take duplicates.

UML Diagram

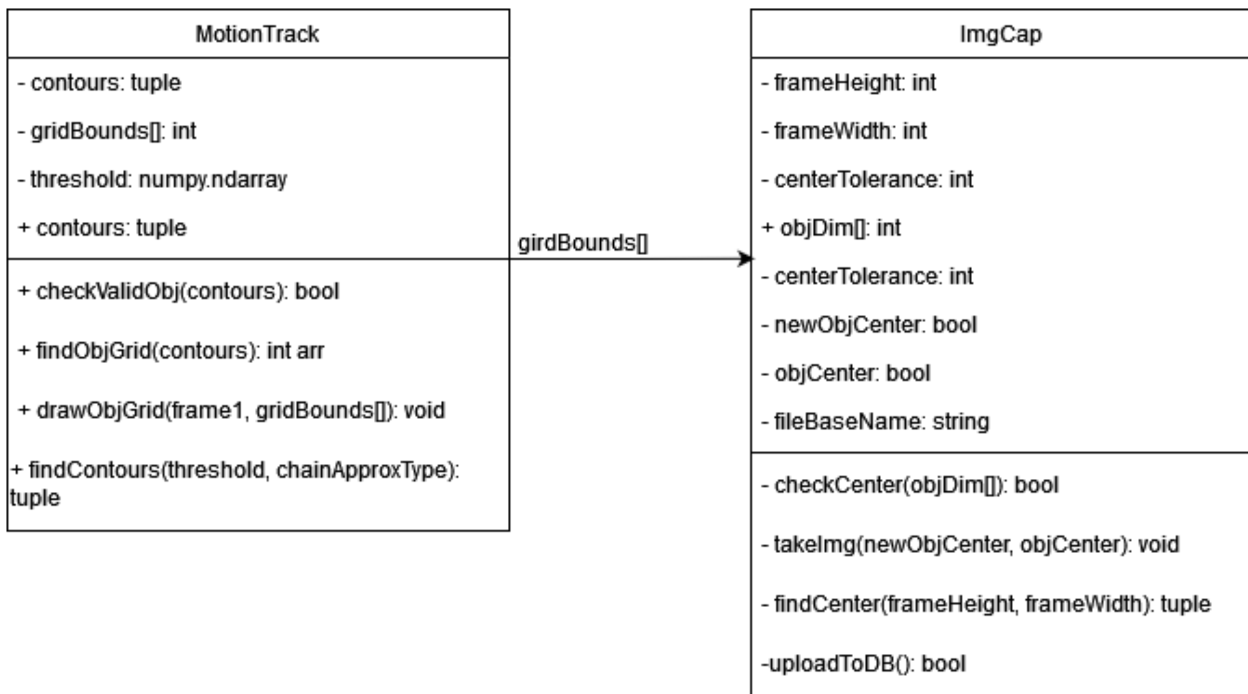


Figure 13 : Image Capture module UML class diagram.

4.3. Image Classification Model

4.3.1. Training Module

Description

Due to the nature of researching multiple vastly different deep learning architectures, the only commonality between them is training. This module is comprised of multiple parts that all lead into the next starting with the loading of training data. Training data is loaded from directories and preprocessed to be the proper format and size depending on the architecture currently training. Once preprocessed, the training data is then augmented to reduce overfitting. These augmentations include random zooms, rotations, or flips. After the training data is ready, the deep learning model trains on the training data. All the statistics from training are stored until the end, where they are graphed.

UML Diagram

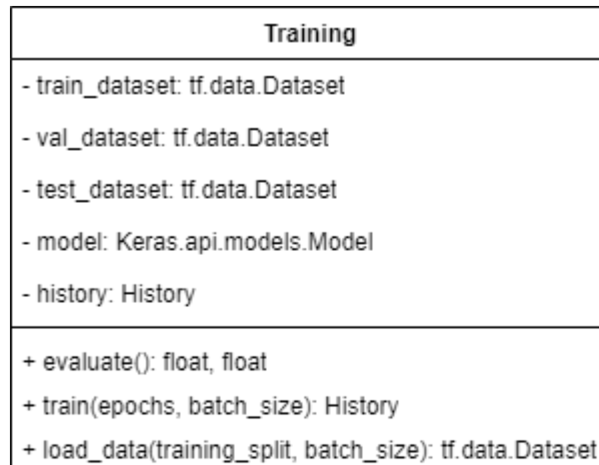


Figure 14: Training module UML class diagram

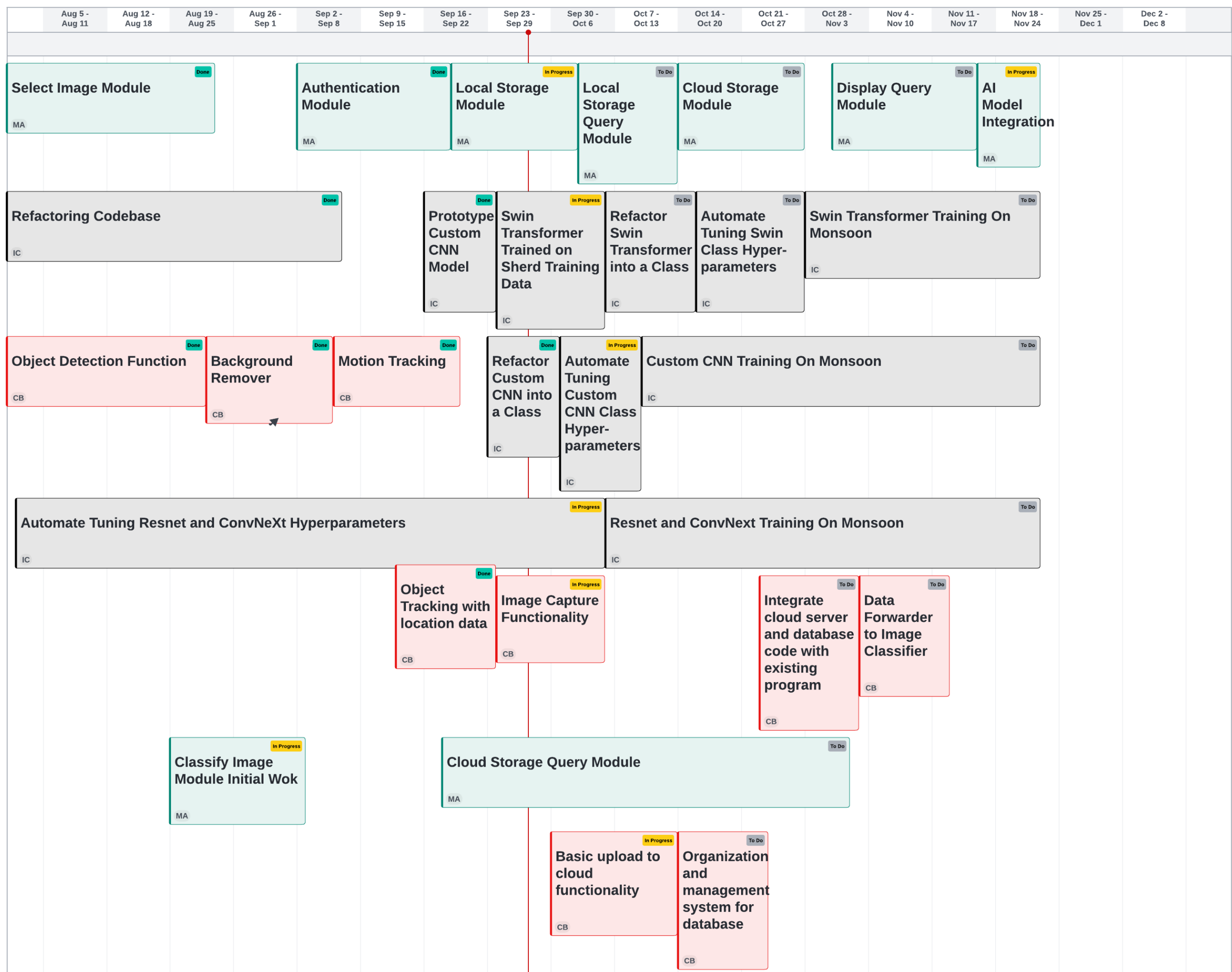
Public Interface Details

1. **evaluate**
    a. **Description:** This method is responsible for evaluating the trained model on the test dataset.
    b. **Return Type:** float, float
    c. **Services Provided:**
        i. Evaluate the trained model on the test dataset and return the test accuracy and test validation.
2. **train**
    a. **Description:** This method is responsible for training the model on the training dataset while validating it on the validation dataset.
    b. **Return Type:** History
    c. **Services Provided:**
        i. Trains the model for the specified number of epochs using the given batch size.
3. **load_data**

a. **Description:** This method is responsible for loading the training, validation, and test datasets.
b. **Return Type:** tf.data.Dataset
c. **Services Provided:**
   i. Loads training data into separate datasets based on the training split.

5. Implementation Plan

Team CRAFT's timeline plans for a project finish time in late November. The following chart demonstrates team CRAFT's implementation plan for the coming semester. Tasks related to the development of the CNN are gray, tasks related to the mobile application are green, and tasks related to the development of the conveyor belt are red.



Work on this project will be split between team members, allowing tasks for each piece of the project to be completed in parallel. Through the initial phases of application and conveyor

belt development, a temporary CNN model will be implemented. The final model will replace the temporary one during the last stage of development.

Team CRAFT plans to tackle the project in tiny modules, these modules most of the times directly correspond to the high-level architecture of the project itself. Details of the allocated development time for each module is listed below, categorized into 3 phases:

Early Stages (August to September)

- **Select Image Module:** This is likely the initial step, starting the project.
- **Authentication Module:** This might be implemented early on to secure access.
- **Local and Cloud Storage:** These would be set up concurrently to accommodate local data storage and caching needs.
- **Display Query Module:** This could be developed alongside storage to visualize data.
- **AI Model Integration:** This might involve initial exploration and selection of models.
- **Refactoring Codebase:** This could be a continuous process throughout the project to optimize code.

Mid-Stage (September to October)

- **Prototype CNN and Swin Transformer:** These models would be developed and tested.
- **Automate Training:** This would involve creating scripts to streamline the training process.
- **Training on Monsoon:** This suggests a specific focus on monsoon-related data for training.

Later Stages (October to December)

- **Formatter Module:** This might be developed to ensure data consistency.
- **Motion Detection Module:** This could be implemented for specific tasks.
- **Custom CNN Training:** This would involve tailoring a CNN to specific requirements.
- **ResNet and ConvNext Training:** These models would be trained and compared.
- **Classify Image Module:** This would be developed to apply the trained models.
- **Cloud Storage Archive:** This might be implemented towards the end to store trained models and results.
- **Image Capture Module:** This could be integrated for real-time or on-demand image acquisition.

- **Model Integration:** Final integration of the chosen model into the system.

The project commenced in late August, with the majority of development taking place from September to early November. We aim to complete the validation and testing phases by the third week of November, followed by handing off the final product to the sponsor in early December.

6. Conclusion

The use of artificial intelligence in archeology will save time, money, and resources. With the ability to classify and catalog sherds quickly, archeologists can return them to archeological sites faster. Our team aims to develop a streamlined process for gathering training data, utilizing this data to train a deep learning model, and finally placing the model within a mobile application to ease the burden on archeologists classifying sherds.

In conclusion, the following project components will be completed by team CRAFT:

- A refined CNN model
  - Improved accuracy compared to the original model
- A mobile application
  - Allows users to upload or take a photo of a sherd
  - Stores classification information
- A conveyor belt system
  - Detects sherds and saves taken images classification images

Artificial intelligence will provide an accurate, consistent baseline for archeologists when classifying sherds. By creating a mobile application and a conveyor belt system, team CRAFT will create avenues for both hobbyists and professionals to utilize the CNN model. We are very excited to see where this project ends up and cannot wait for archeologists to benefit from our work!