**Part Classifier**

**Team Classify**

Sanjay Gururam Sharma

Kevin Song

Nicholas Schaap

Nathanael Temesgen

Syed Aal-E-Ali

Anthony Yao

Andrew Berger

**March 20, 2022**

# 

# 

# 

# Table of Contents

[**Table of Contents**](#_mh3g5y2wdnos) **2**

[**1. Introduction**](#_541xhgkys6rd) **4**

[**1.1 Purpose**](#_igweqqcp8l1a) **4**

[**1.2 Scope**](#_1ylskygzeqxm) **4**

[**1.2.1 Benefits**](#_d3qi46jhozmq) **5**

[**1.2.2 Software Goals**](#_oymnw3nlvwib) **5**

[**1.3 Definitions, Acronyms and Abbreviations**](#_xckmhvrjco7s) **6**

[**1.4 References**](#_8w3yjghk5k05) **7**

[**2. General Description**](#_fhi2hqrk6fhm) **9**

[**2.1 Product Features**](#_p33ocxatd428) **9**

[**2.1.1 iOS Application**](#_j4ugt6hw8fyj) **9**

[**2.1.2 Preliminary Model of Object Identification Step**](#_bzjf6m5uq42g) **12**

[**2.2 Set-up**](#_qsq0fdjvnoo1) **14**

[**2.2.1 Software Setup**](#_r9ci914m9gxu) **14**

[**2.2.2 Hardware Setup**](#_ak9eyjq0g8i0) **14**

[**2.2.3 Environment Set Up**](#_4grawfv3n4w1) **15**

[**2.2.3.1 Camera Distance from Object**](#_ihocx8jkmmxa) **15**

[**2.3 Step By Step Usage**](#_lg93dbf51qmh) **15**

[**2.3.1 Training Step**](#_u5cp8x8fzehq) **15**

[**2.3.2 Testing Step**](#_nwj2jmjhux60) **16**

[**2.3.3 Classification Step**](#_s69td98p41df) **16**

[**2.3.4 Constraints and Limitations**](#_vgfudqkrykon) **16**

[**2.4 Ideal Mechanical System and Assumptions**](#_zabbyuuylim1) **17**

[**3. Software Development Requirements**](#_kachf256dgps) **18**

[**3.1 Risk Reduction Exercises**](#_frm1g3lygox5) **18**

[**3.1.1 Results**](#_jlb6iv9ylmmv) **18**

[**3.2 Model Training**](#_rx2us3gbcu) **19**

[**3.3 Mobile App Design**](#_5b44g81lkisa) **20**

[**3.4 Database Design**](#_gl5w7s2fj1r0) **20**

[**3.5 Reference Architecture**](#_mv8igdj187tk) **21**

[**4. Verification and Validation**](#_hxdui0qk247y) **23**

[**4.1 Verification**](#_4ocyjqt1uad2) **23**

[**4.2 Validation**](#_np5eeoy7v6h3) **23**

[**4.3 Simulation**](#_gmaqr3d7vlx2) **24**

[**4.4 Acceptance Tests**](#_vl782s83if7v) **24**

[**5. Delivery and Costs**](#_b3dpjm3h7fsy) **27**

[**5.1 Delivery Manifest**](#_1uxgjukkj1y7) **27**

[**5.2 Intellectual Property Statement**](#_2lkmppynf8ot) **27**

[**5.3 Cost Estimate**](#_w7407q76u8kz) **27**

[**5.4 Cost Estimate Divided By Task**](#_fyvr4ow1gl4b) **29**

[**5.5 Timeline**](#_jj8ekqdtwaej) **30**

[**5.6 Advertising Plan**](#_2nu9ghg3xcrn) **30**

[**6. Milestones**](#_dyc1cjj63dsy) **31**

[**6.1 User Interface Development**](#_bzpzxxkimohp) **31**

[**6.2 Coin Classification Software**](#_jltys38rhgql) **31**

[**6.3 Conveyor Belt**](#_6x8rkjwa8fzc) **31**

[**7. Press Release**](#_7rzk07qg707s) **32**

# 

# 1. Introduction

## 1.1 Purpose

***Classifier*** is an applied research project to find the potential of available vision processing tools in simple automation systems. Using these findings, our goal is to create a general purpose program that can simulate a mechanical system. This mechanical system will be able to rapidly sort small parts into different bins. These parts can be anything from coins, fasteners, to screws, or resistors. More specifically, this system is designed to identify and sort bullet casings and similar items (items distinguished by engraved/embossed text) as specified by our client. Plenty of image classification and feature extraction programs are available, but what sets ***Classifier*** apart is twofold: scale and ease of use. By scale, we mean that we are confident that we will be able to make a more general purpose image classification system that will not only be able to distinguish parts from each other, but tell the user pertinent information about the part (e.g. which state a given coin is from) at high speeds, for thousands of parts. Secondly, we want the user to be able to extend our program to their own needs. We will provide a user interface for inputting images of new parts that our program has not seen before: allow the user to specify information about that part and give confidence that our system will be able to sort them efficiently.

## 1.2 Scope

Our main design focus for this project is considering objects that contain engraved textual and graphical features. Our client is specifically concerned with classifying between various bullet casing headstamps bullet casing headstamps (see Figure below).



*Figure 1.2(a) - Example of a bullet casing headstamp*

This problem space is particularly difficult because traditional OCR algorithms for recognizing machine-printed text on documents don't tend to perform too well on engraved metallic artifacts. Due to the limitations of our environment, we will be working specifically with quarters as our testing object. We will attempt to extract distinguishing features from various quarters including year, state, etc., in order to be able to classify between different quarters and effectively provide a means for being able to sort them and any other similar type of object, such as bullet casing headstamps.

### 1.2.1 Benefits

The benefits of Classifier are derived from many different use cases which include:

1. Solving the needle in a haystack problem by providing the functionality to search through a large pile of small objects in order to find a specific object.
2. Sorting and grouping a large bucket of objects by category such as grouping coins by their value or quarters by their state.
3. Detecting objects that don’t belong within a group and providing the option for users to relabel those objects and feed them back into the Classifier or just filter unknown objects into a separate bucket.

### 1.2.2 Software Goals

The software goals for Classifier are customer-driven with a focus on meeting the client's needs for fast and accurate image detection and classification.

1. Accuracy - The first software goal of Classifier is to be able to have the accuracy to distinguish between similar objects by identifying the smallest parts within an object. The degree of accuracy we hope to achieve is to be able to recognize text on images and classify the same type of objects into separate buckets. For instance, our software will not only be able to distinguish quarters from other coins such as nickels and dimes, but also be able to categorize each quarter by state.
2. Speed - The second software goal of Classifier is to have a reasonably timed training process so that users can be given the option to classify a new object or correct a mislabeled object. Our software will be able to handle this process asynchronously so that we are concurrently classifying images, while also updating our database with the new image and label. Each new image and label will take at most 1 minute to process and then be added to our database of known images.
3. Scalability - The third software goal of Classifier is to be able to grow our training dataset and database of known images so that we can support growing amounts of data in larger image classification systems. Our software will store data on the cloud so that our users have the option of either growing or shrinking how many objects they want to be able to classify for their specific use cases. To build our model and perform image processing, we will be using a cloud instance that is more powerful than what most users would have on hand.

## 1.3 Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| API | Application Program Interface |
| RCNN | Region-based Convolutional Neural Network |
| ML | Machine Learning |
| CoreML | ML API created by Apple |
| Swift | Programming Language created by Apple |
| Template Matching | Concept where we take a template image and search for similarities to that image inside another image |
| OCR | Optical Character Recognition |
| OpenCV | Open source computer vision functions |
| Servo | Electric motor that corrects mechanism action |
| Realtime Database | Database that handles updates from multiple users who are editing code, etc. |
| Edge Detection | Technique used to find the boundaries of objects within an image |
| Image Preprocessing | Allows an image to be converted into a form that can be used for machine learning tools |
| Conveyor Belt | Device that uses two pulleys to move objects on a belt |
| Microcontroller | A small mini-sized computer that performs simp |
| Batch Size | Number of training examples used in one iteration of training a Machine Learning Model |
| Learning Rate | A tuning parameter in an optimization algorithm that determines the step size (the amount the weights are updated) at each iteration while moving toward a minimum of a loss function. |
| GUI | Graphical User Interface |

## 1.4 References

Al-Frajat, A. K. K. (2018). (thesis). *Selection of robust features for coin recognition and counterfeit coin detection*. Retrieved March 9, 2022, from <https://spectrum.library.concordia.ca/id/eprint/984864/1/Al-Frajat_PhD_S2019.pdf>.

Aslan, S., Vascon, S., & Pelillo, M. (2020). Two sides of the same coin: Improved ancient coin classification using Graph Transduction Games. *Pattern Recognition Letters*, 131, 158-165. <https://www.sciencedirect.com/science/article/pii/S0167865519303708?casa_token=5Ckw2_0AJ7sAAAAA:z6k-dympDiUeoUcdkKPKUFuSWKJ8TGbRvkScsS7ierysZ980_Uwwf_-GNTkDP9Edrxn7yxz8Pg>

Gupta, Aitik. “Object Detection in 2020 -  from Rcnns to Yolov4.” *Medium*, Code Heroku, 28 June 2020,<https://medium.com/code-heroku/object-detection-in-2020-from-rcnns-to-yolov4-fd549b182b78>.

Van Der Maaten, L., & Postma, E. (2006). *Towards automatic coin classification*. na. <https://lvdmaaten.github.io/publications/papers/EVA_2006.pdf>

Zelic, F. (2022, March 8). *[tutorial] tesseract OCR in python with Pytesseract &amp; OpenCV*. AI &amp; Machine Learning Blog. Retrieved March 12, 2022, from <https://nanonets.com/blog/ocr-with-tesseract/>

*Estimation based on function points* - University of Kansas. (n.d.). Retrieved March 16, 2022, from <https://people.eecs.ku.edu/~hossein/Teaching/Sp18/811/Papers/fpa-cocomo.pdf>

# 2. General Description

## 2.1 Product Features

Amongst other mechanisms we’ll be using to demonstrate our software, our main product is going to be a preloaded raspberry pi and camera setup. Plugging this into a screen would open up a graphical user interface (GUI) application. We will also provide an IOS application that controls the raspberry pi in the case that a user does not have the proper environment to interact with the GUI. The GUI and IOS app will allow the user to create as many labels as they wish for categorizing an item into different buckets, and then take multiple pictures of that item to train our algorithm into being able to recognize it from a gallery of images. After the model has been trained, a user may take a picture of an item and the Raspberry Pi will be able to properly classify the item and place it in a specified bin with its corresponding label. If the item isn’t recognized (hasn’t been trained, unrecognizable due to rusting/damage, etc.) that picture will then be placed in another bin labeled ‘unrecognizable’. The user can either re-train the model to help improve its accuracy by taking a picture of that faulty item or will be able to discover that there’s something off about a particular item (e.g. doesn’t match other trained pictures with that label therefore it’s not that item). Because our system is going to initially be solely driven by a Raspberry Pi, all of its features will lie in our GUI application – which is specified in the following subsection [2.1.1].

### 2.1.1 Application

* Camera

Our application will control a raspberry pi camera to take pictures of items to train a model that will classify items into the appropriate bins

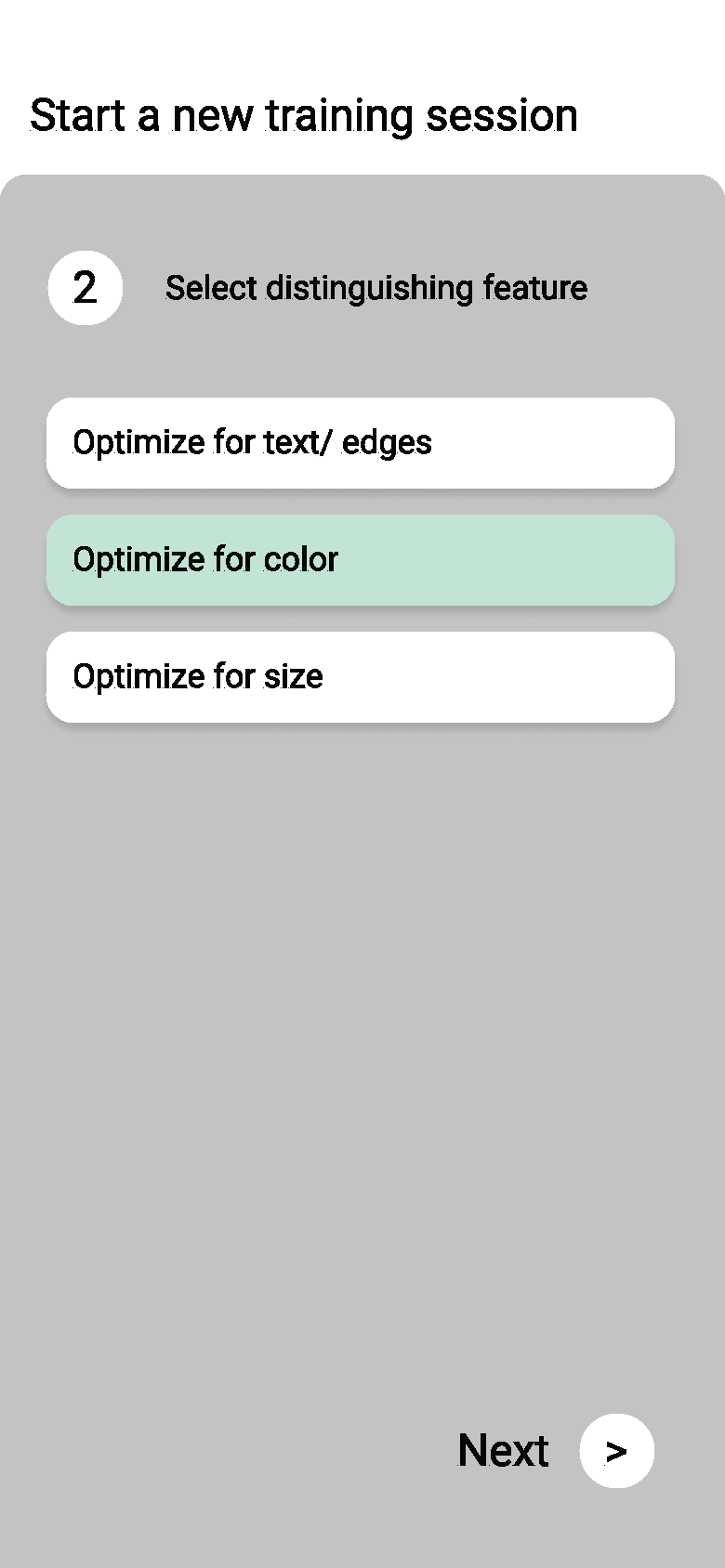
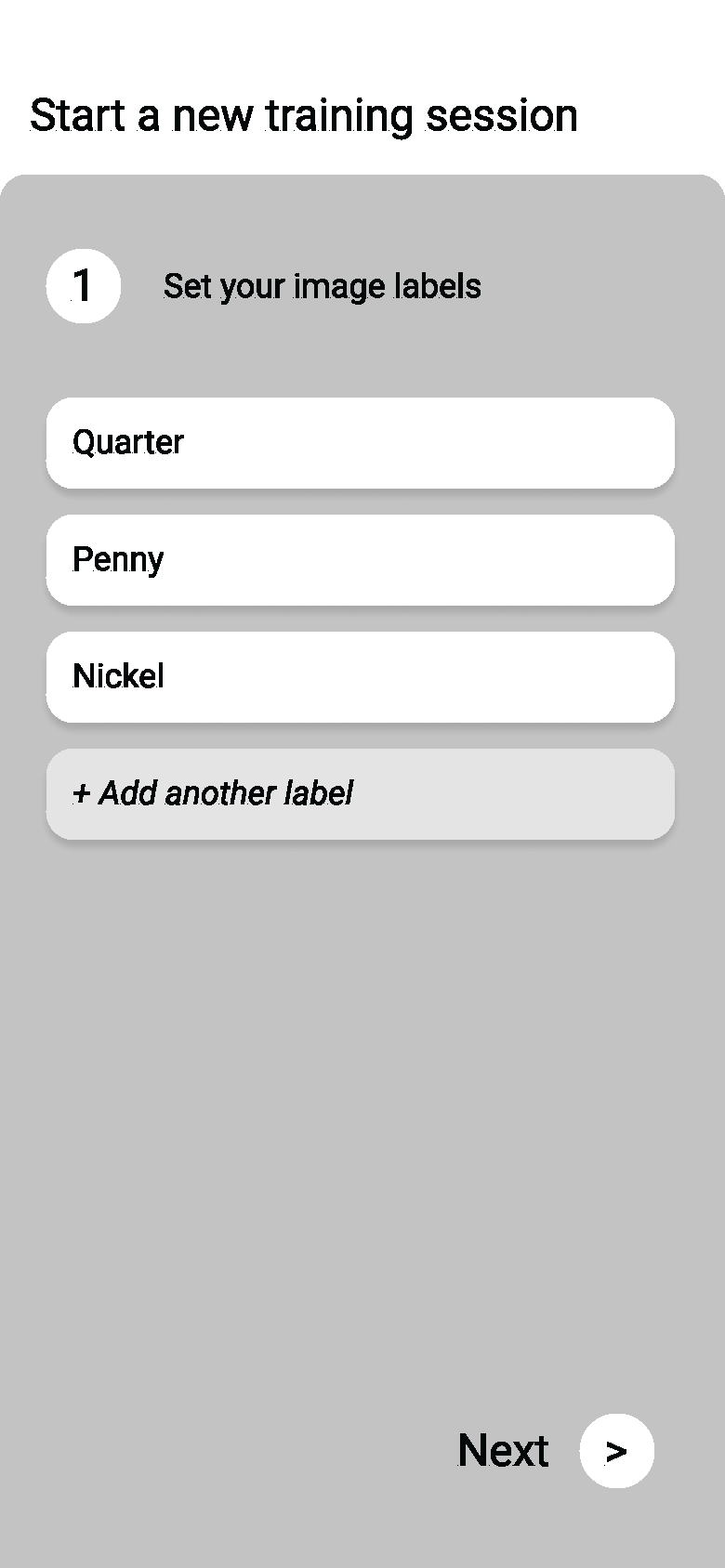
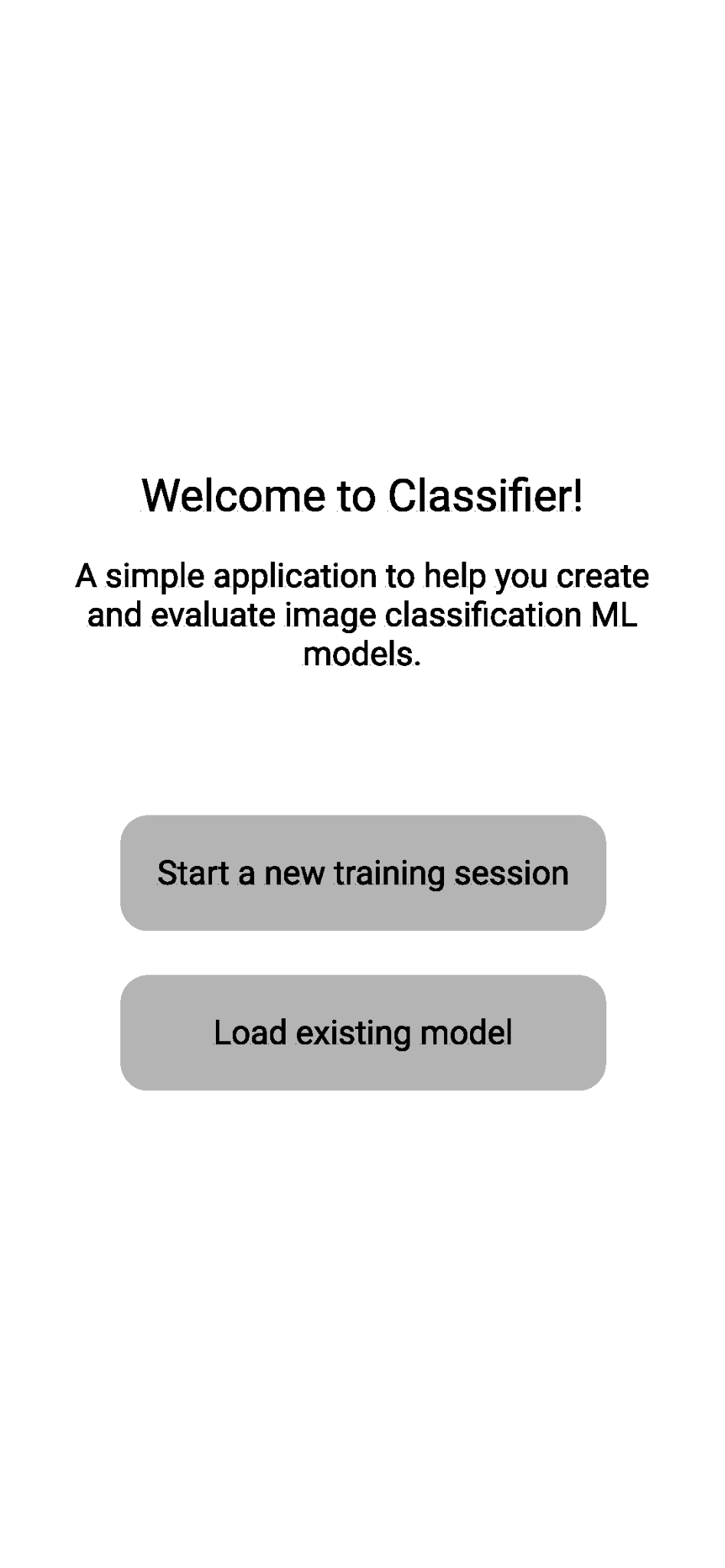
* ‘Train/Classify’ Modes

Depending on what mode it’s in by use of a picker, the images that are being captured by our application will be used to either train the algorithm or classify the items in those images into a specified bin. If on ‘Train’ mode, a text box will be displayed where the user can type in a label that they want to associate imminent pictures of an item to.

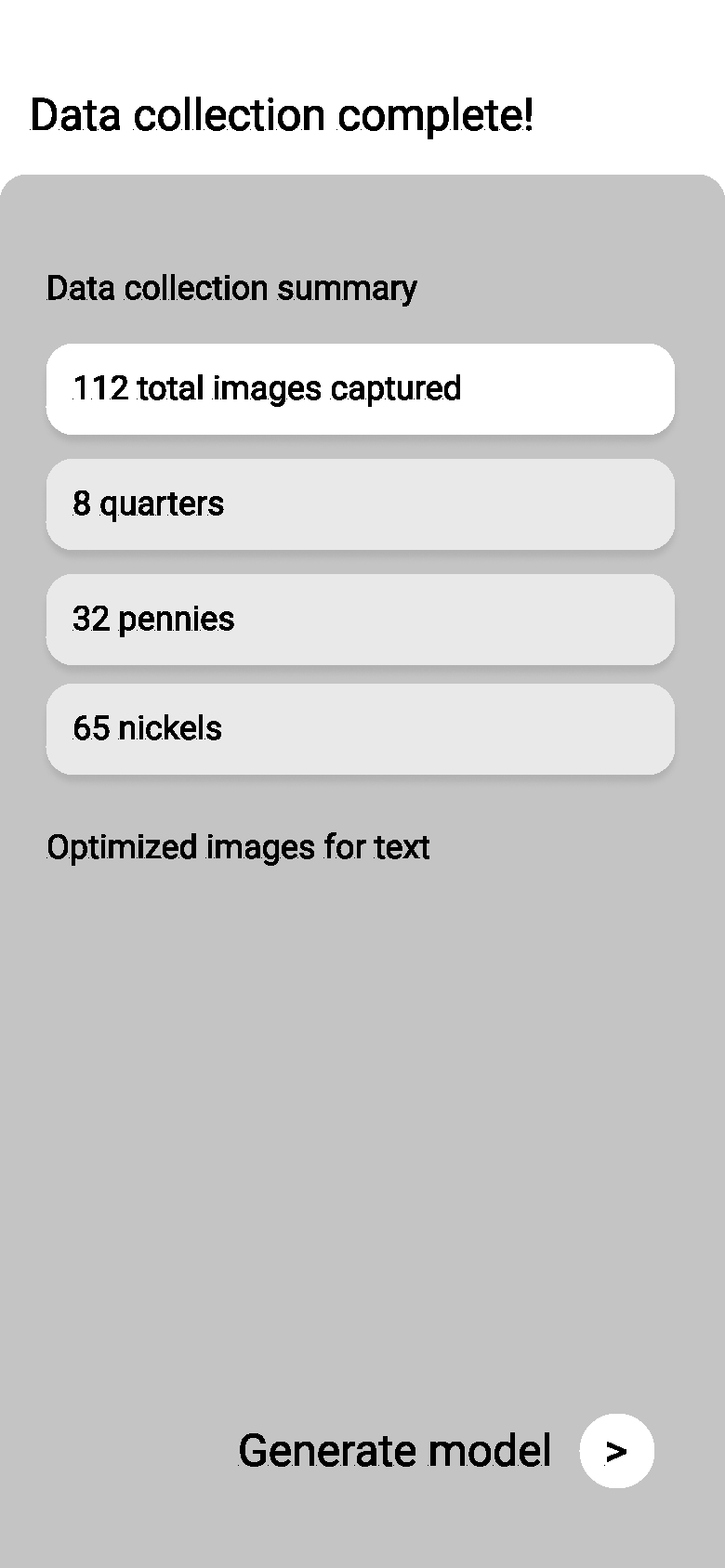
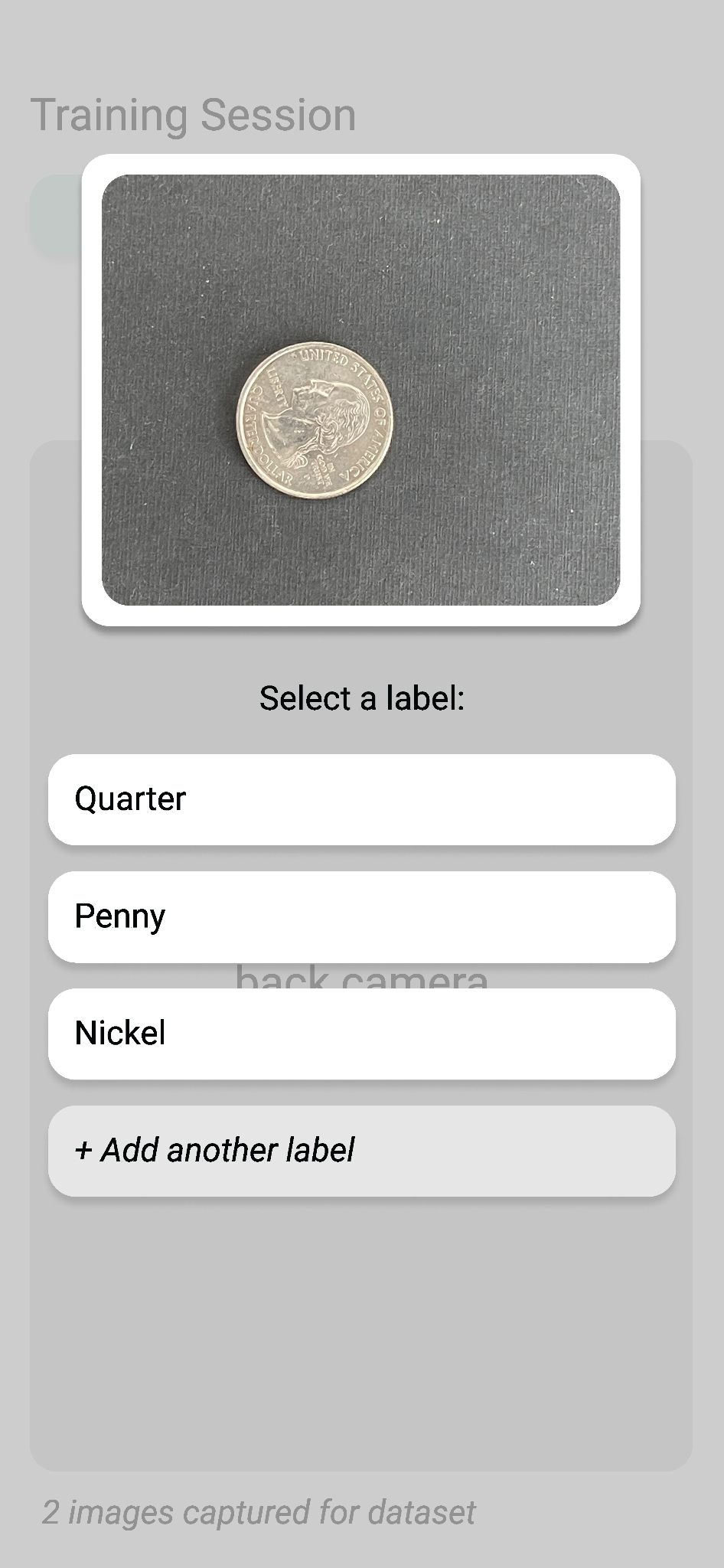
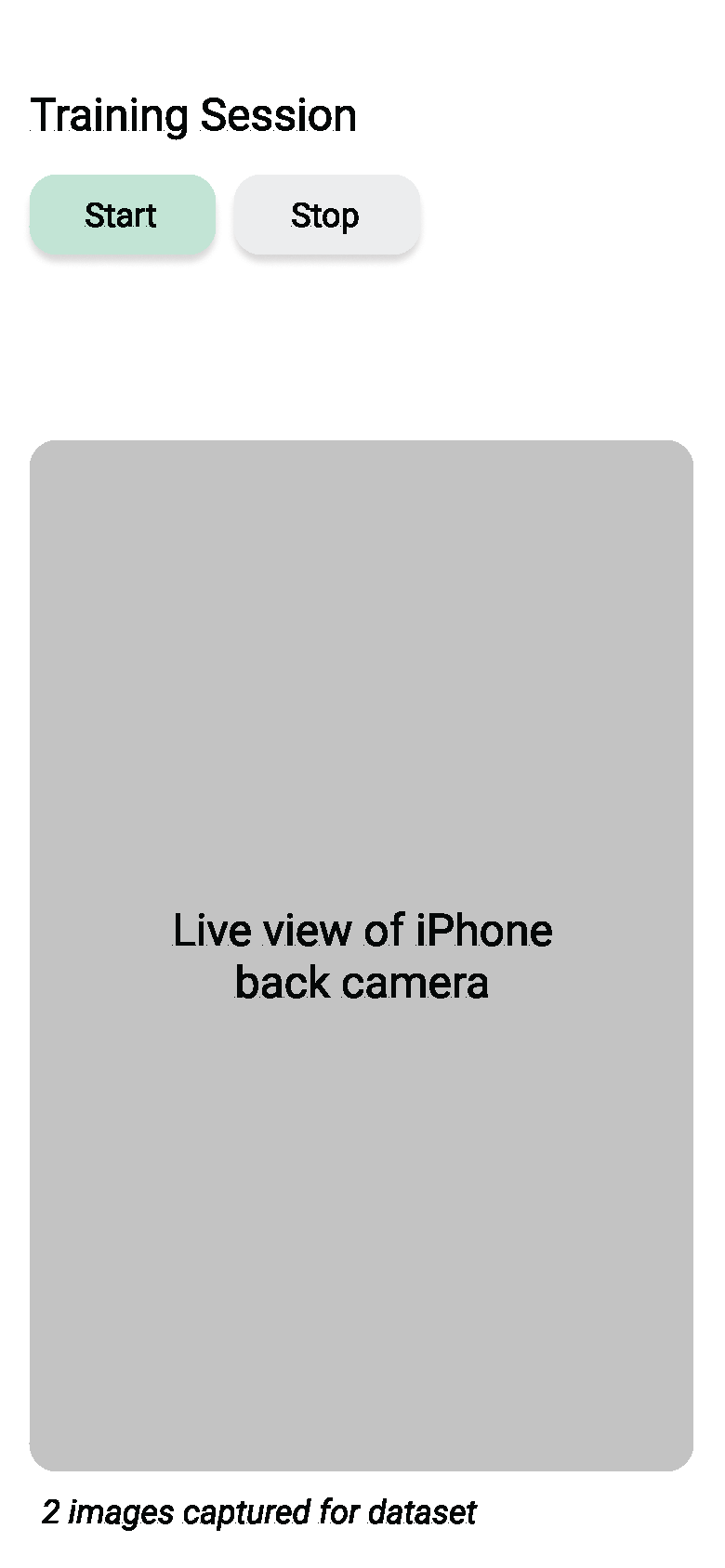
* ‘Bins’ with Classified Images

There’ll be a bin for each of the images the user had specified during ‘Train’ mode. When a bin with a label is tapped on, that data will then be associated with the image and stored in our database to be used later for training the ML model.

**Training Mode Wireframes**

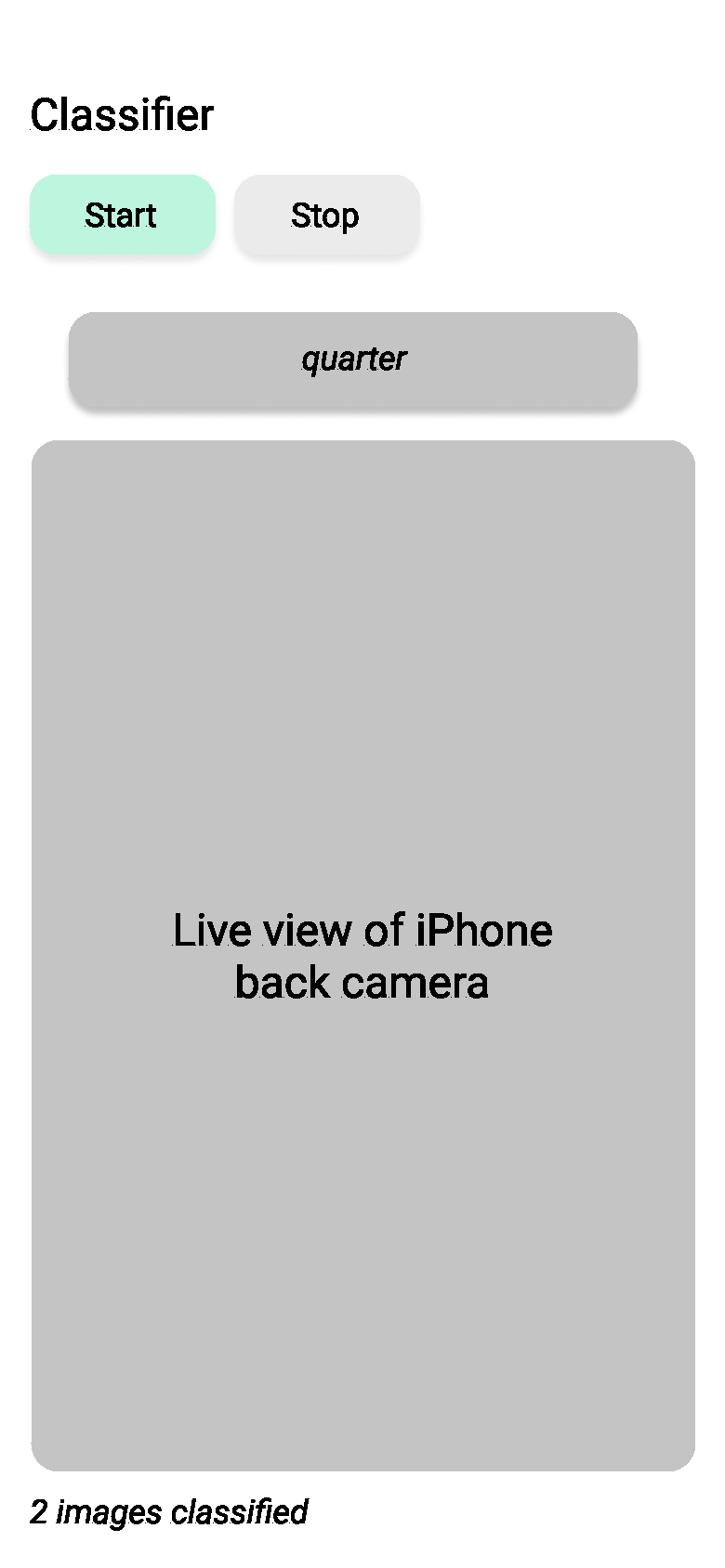


The first step to starting a new session requires prompting the user to start a new training session to establish a new model or loading an existing model. This model will likely be stored on our remote database. When starting a new training session we will prompt the user to establish their labels for various objects they wish to classify objects on. Then, we will ask them to choose an optimization. The optimization will affect how images are pre-processed. For example, for text/images our pre-processed image will likely involve some amount of edge amplification through edge detection schemes, whereas for color we will ensure that color is preserved in the pre-processing of our captured images.



Once the labels are set the user will have to click 'Start' to initialize the live view back camera and send a signal to the realtime database that the user has started capturing images such that any real time system can synchronize with the user's progress. In addition, if the user is not using an iPhone, the “Start” button on the GUI can then be used to send a signal to the Raspberry Pi. At this point we will begin sensing whether an object has entered the scene and capture an image when that object is near the center of the view. We also have the option at this point to have the user manually tap when to capture a new image. After an image has been captured, the user will be prompted to select a label for the new image. After the user is satisfied with the number of images captured for the dataset they will be presented with a data collection summary and the action to generate the model using the new training data just captured.

**Classify Mode Wireframe**

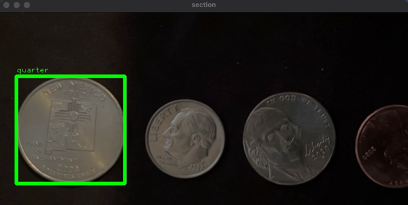
****

When a user loads an existing model, they will not be prompted to assign labels to captured images. Instead we will display the classification result given to us by the pre-trained model based off of the image captured. The 'Start' and 'Stop' buttons here will start and stop the camera respectively and the state is maintained in real time by our database such that any mechanical system can tell when the user is actively capturing images and actuate accordingly.

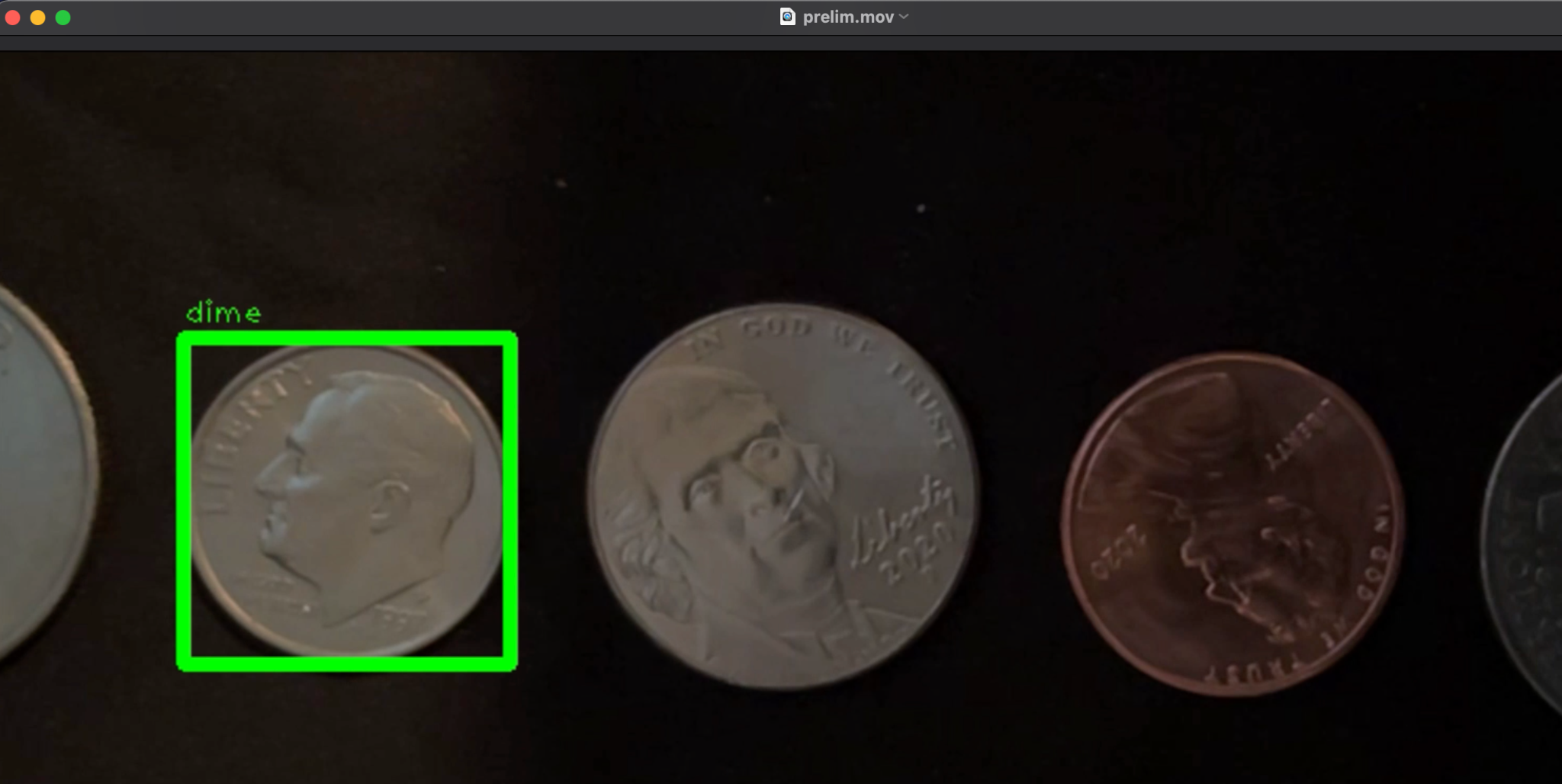
### 2.1.2 Preliminary Model of Object Identification Step

This exercise of labeling coins passing through a camera gives an idea of how the final product will be able to classify objects. The figures below are frames taken from a video of the coins being slowly slid across a table as if they were on a moving conveyor belt. In each frame, we can see the left-most coin get identified. While here we are only placing the label above the coin, the system that the app is integrated with will be able to translate this output into a bin selection using motor control.

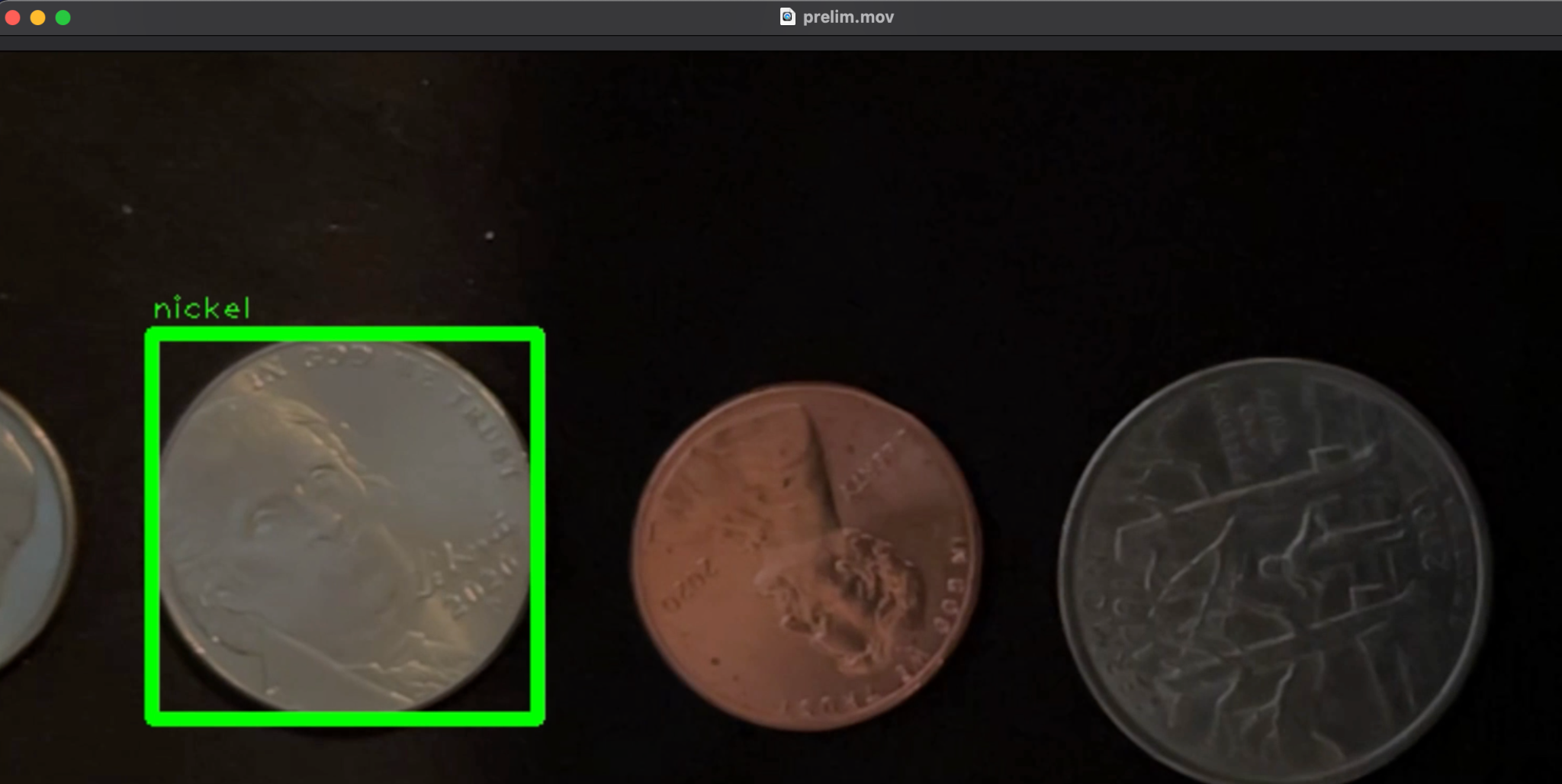
**Conveyor Belt Simulation:** The coins below are moving in the leftward direction and are being identified one by one.



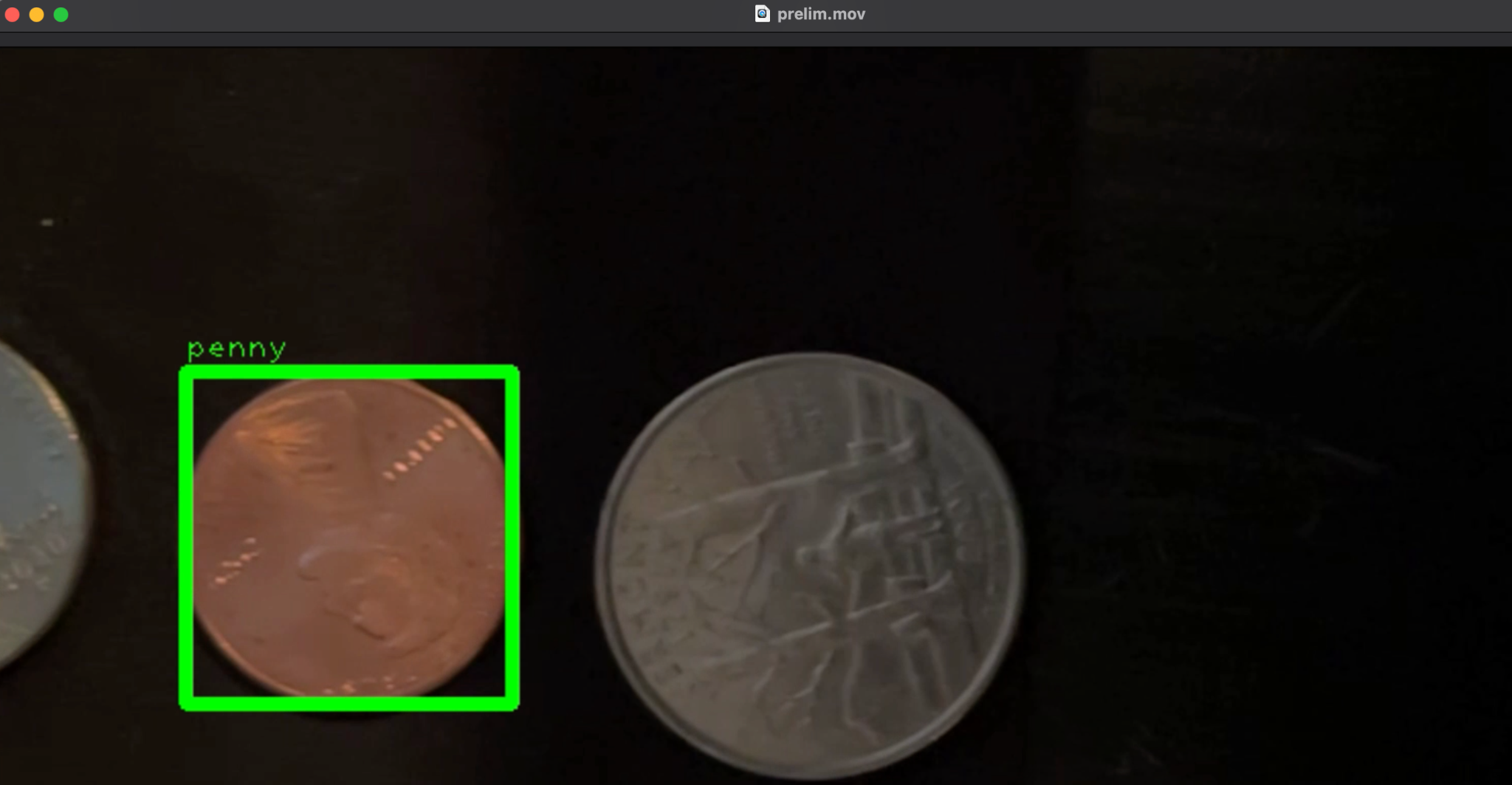
*Figure 2.1.2(a)*



*Figure 2.1.2(b)*



*Figure 2.1.2(c)*



*Figure 2.1.2(d)*

Recorded video of conveyor belt simulation is located in the SVN repository for Team Classify.

## 2.2 Set-up

Our algorithm’s accuracy is almost completely dependent on how the user sets everything up. This includes the software, hardware, and the environment.

### 2.2.1 Software Setup

The setup of our software will be quite simple. If the user would like to control everything on their iPhone, the program will be available to download via the Apple App Store. For demo purposes, we will be offloading the app onto an iPhone via a source in XCode without having to purchase the $99.00 Apple Developer license until we are ready to ship it publicly. If the user decides to go the alternative route and use the Raspberry Pi version of the software, a copy of the software will be available for download on a Github repository which will be provided. The Raspberry Pi will be responsible for controlling the mechanical system and receiving signals from our program: telling the microcontroller at what rate to move the conveyor belt and when to take a picture. It also controls the servo that sorts the part into their respective bin and communicates the results of the program’s classification results back to our app.

### 2.2.2 Hardware Setup

* Conveyor belt
* Bin Setup
  + Servo
  + Motors
* Mac Setup
  + MobileNet runs on macOS 10.13 (High Sierra)+ which currently only runs on the following Mac models:
    - MacBook (Late 2009 or newer)
    - MacBook Pro (Mid 2010 or newer)
    - MacBook Air (Late 2010 or newer)
    - Mac mini (Mid 2010 or newer)
    - iMac (Late 2009 or newer)
    - Mac Pro (Mid 2010 or newer)

*If going the Raspberry Pi route:*

* Raspberry Pi = https://tinyurl.com/39mzkhf2
* SD Card = https://tinyurl.com/k6j5vcrf
* Camera compatible with Raspberry Pi = https://tinyurl.com/3r49sy99
* MicroUSB Power Adapter = https://tinyurl.com/2p98jay2

*If going the iPhone route:*

* iPhone Setup
  + iOS 14.5+ (iPhone SE or newer)
* The minimum hardware requirements we require to program with Swift and XCode are:
  + Intel i5 CPU
  + 4 GB RAM
  + 128 GB Disk Storage
  + MacOS 10.9 or later

### 2.2.3 Environment Set Up

The environment setup is the first step to getting the best results. Lighting conditions will be adjusted to accommodate consistent imaging. It is important to avoid oddly shaped shadows or bright flashes interfering with the imaging area. The camera must be secured at a height that allows for a clear and well-focused view of the object. Additional hardware requirements include a conveyor system capable of bringing items to the camera and a sorting system capable of sorting items based on signals from the system. (Note that for testing, a system capable of mimicking the mechanical system will suffice). Lastly, open up the Part Classifier Application.

### 2.2.3.1 Camera Distance from Object

The distance between the camera and the “conveyor belt” is pivotal in determining how accurate our model will be. Because of the amount of fine detail we’ll need to extract from each item (e.g. year in which a coin is made), we need to have things such as resolution, lighting, and background all controlled prior to running items through our product to be classified. The exact distance between the camera and the belt may vary depending on the scale of the items that we’re classifying, the room in which the belt will be placed, etc. so a proper setup must be conducted prior to running our application.

## 2.3 Step By Step Usage

To effectively use this application, the user must follow a step-by-step process to properly train, test, and classify their images.

### 2.3.1 Training Step

Once the software, hardware, and the environment have been set up, the user may start labeling the objects and training their model. You may start and monitor this process through the iOS application or a desktop monitor. The training of images requires some supervision and knowledge about the objects. There are two modes through which you can label items through the system.

1. Objects are passed through one at a time and then the user will have to enter in the label for that object
2. The user enters in a label and then roughly a dozen objects of a single class are passed through one at a time. Each object will then be given that label. The user will only have to tell the system when they want to begin labeling a new class of objects.

### 2.3.2 Testing Step

With the training step complete, it is important to perform the testing step which improves the model. This process also requires supervision and knowledge about the objects. For testing, the user may take about half a dozen items from each class and pass them through the camera. After an object has passed through the camera, the belt motion will pause and the predicted label will show up on the screen. If the model was correct then press continue. If however, the model predicted incorrectly, press the “Relabel” button and enter in the correct label.

### 2.3.3 Classification Step

After the training and testing processes have been completed, the user may enter into “Classify Mode”. Classify mode does not require constant supervision and is designed to run through hundreds of items. The only requirement of this step is that objects pass under the camera one at a time. Each object will pass through the camera and then the motor attached to the array of bins will move the predicted bin into the centermost position, for the object to cleanly fall into it. If the user is unsatisfied with the sorting results, images of the items that were sorted will be stored and can then be labeled and thus improve the sorting accuracy of subsequent classifications.

### 2.3.4 Constraints and Limitations

This sorting system is optimized for small circular objects so results may vary when trying to sort through larger or differently shaped objects. The camera is taking pictures which will then be transferred to the app, which will then be uploaded to an S3 bucket for analysis and classification. Therefore, the image quality of the camera will have a huge impact on the identification and classification of the object, hereby having a huge say on the accuracy rate of the model.

## 2.4 Ideal Mechanical System and Assumptions

Ideally the entire sorting system will include a hopper, a dropper, a separator, a conveyor, a camera, a sorter. The hopper will be for the user to put all the items he wants to be sorted in. A dropper to release items in a controlled fashion onto a conveyor. A separator that will separate dropped items through the use of vibrations. A conveyer to bring items to the camera. After the camera takes an image, the image will be processed and the sorter will then bring the items to the desired bin. The mechanical system can be created to mimic a fully controlled environment. Variables such as lighting can be controlled and other variables such as dropping time and conveyor speed are known. With these known variables, it is possible to calculate when exactly an object is under the camera though we can’t assume it will be centered or if an object was properly dispensed onto the conveyor. This however will allow us to simply use a timed camera where there is an occasional case of having no object in the frame. Our program will take these images in either training or sorting mode. Training mode does not require the other mechanical parts to do anything but sorting mode will require the sorter to move based on the program output. If the program determines an object to be Type X, a signal will be sent to the sorter that will allow it to sort it into bin X. Note that we are not creating an entire mechanical system but rather this is the system our software will be compatible with through the development of a comprehensive API.

# 

# 3. Software Development Requirements

## 3.1 Risk Reduction Exercises

We came to the decision to use CoreML as opposed to other Computer Vision/Machine Learning Libraries through a series of experiments. At the outset of the project we split ourselves into three different teams striving to use three different technologies to teach a computer to distinguish between images of pennies, quarters, nickels, and dimes. First we’ll describe the two that did not work out. We attempted to utilize TensorFlow’s object detection API for this base problem statement. After training, two separate pre-trained models on two different datasets of coins (150 vs. 550 images), as well as playing around with a whole host of different input options (distinguishing between heads and tails for our image labels, batch-size, learning rate, step size, total number of steps, image preprocessing settings) we were unable to consistently get it to draw a bounding box around the correct coin in our test set. TensorFlow in general also takes quite a long time to train a given model with even more compute than a mid-tier laptop (5-6 hours each time); we used a Ryzen 3700X processor, 16GB of RAM, and a GeForce RTX 2060 Super graphics card with 8GB VRAM for our training. Our other alternative, ImageAI, was not giving us accurate results with the same dataset mentioned above, nor did we have sufficient hardware to have it run in a reasonable amount of time. Specifically, it took 10 hours of training overnight to process the dataset with 550 images using ImageAI, and only 5 training models were produced with the fifth one achieving only 27% testing accuracy. In addition, there is a lot of setup involved such as formatting images with unique labels and installing outdated dependencies that could only be set up in Linux and Unix based environments. CoreML was by far the fastest to train, the easiest to set up, and provided the most accurate results using one of their pre-trained MobileNet models. We also found it convenient to use an iPhone, Swift, and the CoreML library because of how commonplace iPhones are compared to more traditional camera setups as well as how sophisticated the cameras in these devices have become. For example, we will be able to take advantage of the plenty of extra features these devices offer such as lidar sensors, gyroscopes, etc.

## 

## 3.1.1 Results

CoreML was by far the fastest to train, the easiest to set up, and provided the most accurate results using one of their pre-trained MobileNet models. Because CoreML is Apple’s machine learning framework and is available to use on XCode which is used to develop software for macOS, iOS, iPadOS, watchOS, and tvOS, we are restricted to using a MacOS instance to communicate with our model.

Based on these findings we will have a Raspberry Pi controlling the camera and conveyer-belt setup which will send pictures to a Mac AWS instance to train our model and perform inference on new images for sorting. The output of the classification and feature extraction will be shown via a GUI on a monitor connected to the Raspberry Pi, but we will also make an iPhone application available for when no desktop application is available, making it much easier to shoot photos and analyze them (all done within one machine)..

## 3.2 Model Training

As an applied research project we are planning an ever-evolving, iterative approach to the design of our core ML model. From our early risk assessment exercises we learned a great deal about model training that we can take advantage of throughout our design/ development process. Specifically, we learned that pre-processing of images sent into the model will have a significant effect on how well the model performs as well as how many images it takes to train the model to optimal precision. While we are likely to make use of machine learning as our primary means for identifying and classifying features in the images we take, we plan to also take advantage of various image processing techniques to amplify certain features such as text and graphics and edges that will make it easier for our model to distinguish between different images. This includes, but is not limited to, performing edge detection, and thresholding. These initial operations will allow us to focus on the more important characteristics of an object. Our plan is to explore various options throughout our development process as a part of a two pronged approach to classifying objects:

1. Using image processing techniques to recognize text/engravings and general features associated with an object.
   1. Tesseract-OCR engine for reading text
   2. Template Matching to supplement model predictions
   3. SIFT algorithm for extracting key points from an image
2. Models/ libraries we will utilize:
   1. CoreML - Apple's machine learning library specifically designed to operate on Apple devices such as the iPhone, Mac and iPad.
   2. Mask RCNNs, a modern computer vision model applied to smaller regions with an image for more accurate results.

## 3.3 Mobile App / GUI Design

Our iPhone app / Desktop GUI will be the center of our user-facing experience. The iPhone app / Desktop GUI portion of our system will display the results outputted from our MacOS AWS instance fed by images from the camera connected to the Raspberry Pi. The app will be synced with this instance in such a way that the state is publicly accessible by various other realtime systems that want to make use of the classification results in real time (Desktop GUI application for the Raspberry Pi).We will use Swift/ Swift UI to build the UI of our app and Python to develop our desktop app for the Raspberry Pi.

## 3.4 Database Design

To connect/sync all the various components of our end-user system (iPhone app/Raspberry Pi API, Desktop GUI, user-implemented mechanical system, and model training history/ data) we plan to leverage a real-time Pub/Sub message platform and database. The advantage of using a real-time database with Pub/Sub functionality to record the state of the app is that we can make the state publically accessible to any generic mechanical system such that it may listen and subscribe to changes in the data. In this way it is possible to send generic signals such that a mechanical system may actuate accordingly. While creating a functional mechanical conveyor belt/ sorting system is out of scope for our project we still want to build-in the ability for users to plug-in their own systems. In our case, it is important that we consider low-latency services that are specifically configured to accept subscribing connections such that the database can push data to any client after it is updated. For our specific use-case, we might want to leverage a fully-managed, severless realtime database from a top cloud provider. These sorts of services tend to be inexpensive and extremely scalable. They do a lot of the heavy lifting for us in terms of server management and database configuration which are not the primary goals of our project. For our own development and testing purposes they also tend to offer generous free tiers which makes them a great starting point.

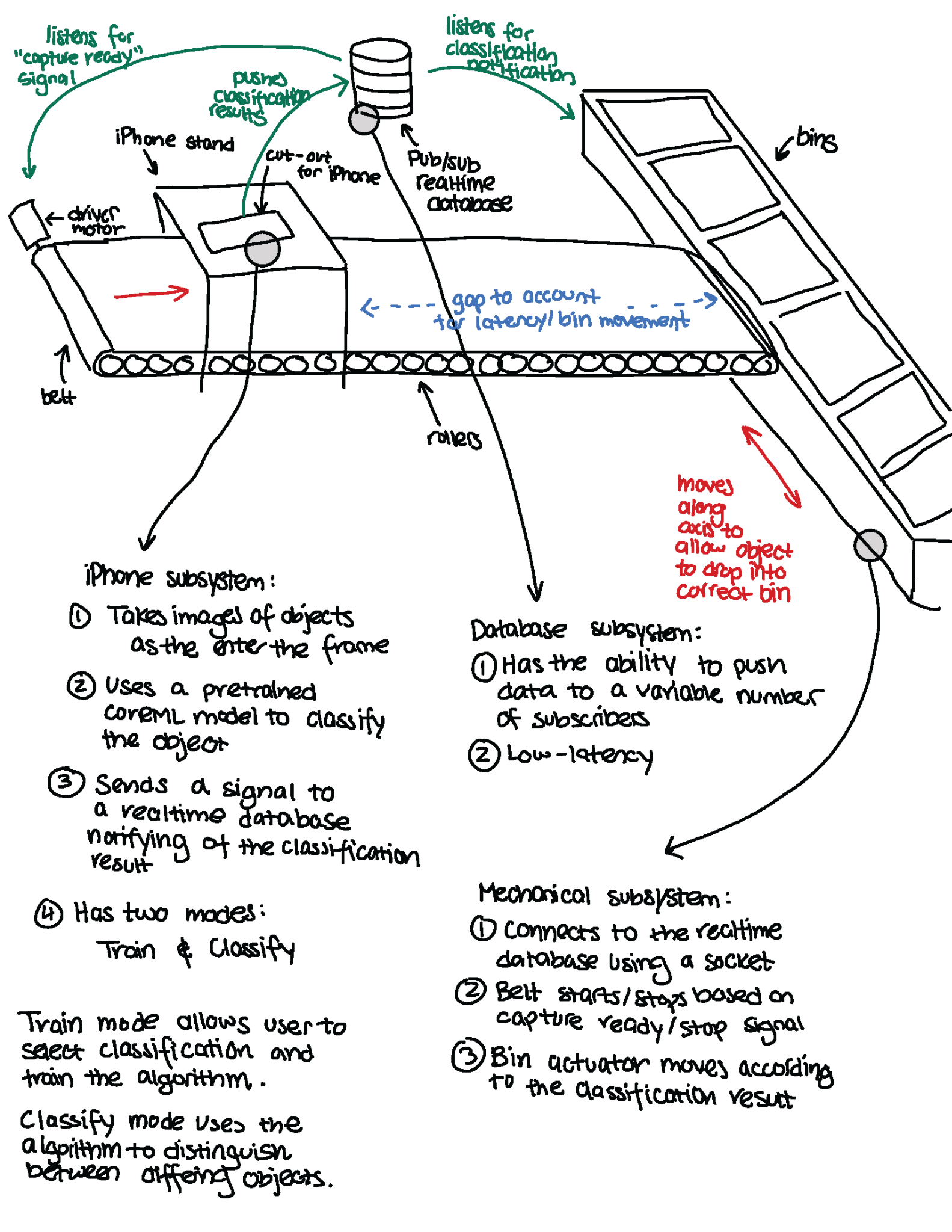
**Various Options:**

* AWS Amplify (https://aws.amazon.com/amplify/)
  + Pro: Combines both a realtime database and Pub/Sub system
  + Pro: Has a well-maintained Swift SDK
* Google Firebase/ Firestore (https://cloud.google.com/firestore) - NoSQL document based serverless database by Google that is specifically designed for real time applications
  + Risk: Client is strongly opposed to interacting with Google services
* Supabase (https://supabase.com/) - open-source Firebase alternative with a Postgres backend
  + Risk: Does not currently have an established Swift SDK
* Ably (https://ably.com/) - Realtime pub/sub service
  + Risk: Ably is not itself a database. will require the addition of another database counterpart to store historical data.
  + Pro: Comes prepackaged with Swift SDKs

**Our Recommendation:**

We recommend moving forward with AWS Amplify. Amplify provides us with most frictionless service moving forward since it provides an all-in-one environment to interact with Swift, store data (Amplify Datastore), store images and model files (AWS S3), and provide realtime pub/sub data synchronization that we can make publicly available for mechanical systems that want to interface with our application (Amplify Pub/Sub).

## 3.5 Reference Architecture



\*The iPhone in this design outline will be replaced with a camera and raspberry pi module.

# 4. Verification and Validation

## 4.1 Verification

The verification of our program is intended to ensure that the results returned by our software are valid. It is important in order to warrant the fact that software API works properly. In this case, it is known that the customer will be testing this with the head of bullet casings, which contains circular text. Therefore, we want to use a dataset that is similar to that of a circular object with text written on it, such as a coin database (database similar to head of bullet casing). Because it is preferred to use U.S. coins as all developers here are from the U.S., and there is a lack of U.S. coin datasets online, the developers created their own dataset of coin images that they are running, which includes a variety of coins in different lighting environments, different backgrounds, different centering of the coin, and varying picture quality. This can be run multiple times on multiple portions of the dataset to determine how many data points are required. In order to verify that the coin identification works properly, the developers also have xml files that correspond to the labels of each image to compare to which can calculate a prediction accuracy.

## 4.2 Validation

Validation of results is very important for any system that relies on image classification, object detection, and extraction of features in a given object. There is no point in sorting objects if the assumptions that we use to sort or the information we return to the user about certain features are wrong. Validation of our results will take a few different forms. First and foremost we will have set aside a database of images containing ground truth information associated with a part that we have trained on so CoreML will be able to discern with a percentage of confidence whether its guess is correct or not. Anything below a certain threshold will be set to the side and not be sorted. This is actually a good thing because for new parts that the system has not been trained on, the user will be able to go back into train mode of our app and provide more information and images that describe the ground truth for that given part. We will also be constantly validating the models that we use throughout the development process. We expect greater than 95% accuracy for both classification and information extraction for parts that our system will have already been trained with, and greater than 50% accuracy for new objects that the user has only provided a small amount of training data for. This will increase over time as our program will take parts for which it has classified with a low confidence percentage and input them back into the system: prompting the user for more information, images, or running some preprocessing steps before inference to get the most out of our data that we can.

## 4.3 Simulation

In the attempts to validate that our product works, we’ll run a simulation that takes coins from different states and made in different years and see if our software is able to pick up those fine details. After that, we will mount a camera onto a table where the camera is facing down and have a sheet of coins displayed in a single-file line motion under the phone where pictures will be taken. If our application is on ‘Train’ mode, we’ll command our Raspberry Pi to take a picture once a coin is motioned directly under it and repeat that process for each coin on the sheet. If in ‘Classify’ mode, we’ll do a similar process as in ‘Train’ mode, but instead of moving the sheet to the next coin right after taking the picture, we’ll follow the display prompted by the Raspberry Pi classifying the coin and physically move the coin to its proper bin. Once the image has been classified, we’ll move the sheet so that the next item is in the picture frame. This will simulate a conveyor belt that’ll have items in a single file line and will motion coins under the camera so that an item will be displayed, classified, then move on to the next item that’s on the belt. As a result, each object will be classified one at a time.

## 4.4 Acceptance Tests

Our client plans to use the software to sort through bullet casings. Bullet casings are small circular objects and their key identifying feature is circular text along the edge. Since we do not have access to bullet casings, a similar object will be used for testing. Quarters will take on the role of our test object since, like bullet casings, they are circular and have circular text along their edge. Additionally, they are similar in size to a bullet casing. Using quarters to test our system will give similar results to using bullet casings.

We will not build the entire mechanical system and so must simulate the conveyer belt and sorting system. For each quarter, we will place it under the camera and the sorting system will identify the object via by displaying a label that appears on the application.

Our acceptance tests will fall into two categories. First will be the user interface and the second will be related to our data management and machine learning pipeline.

The baseline standard by which we pass or fail our tests will be derived based on whether our software can be deployed in the wild. This means that we will measure success through whether the user can deploy the ML and use our software without it breaking. For example, we will ensure that the user is able to use our software with no dependencies breaking, that all errors are checked and handled, and that any input data is being properly read, stored and controlled. All of these requirements need to be consistently met in order for us to pass our system check and feel confident in deploying our application. Additionally, we will monitor the validation performance of our ML algorithm to ensure the software component of our system is working properly, with a specified requirement on classification accuracy described later in this section.

It is difficult to quantify user interface, so we will rely on feedback from test users and the clients along the way to make sure we are able to meet our outlined requirements. Field tests with non-tech individuals will be conducted when the final product is ready to check whether the application is easy to use. We will also begin testing user experience when the application framework has been developed and ask for feedback on how to improve the application, which may be useful information for updating the baseline system requirements described above.

System Requirement Questions:

1. Were the required software dependencies installed before deployment?
2. Did the GUI application display successfully?
3. Did you encounter any error messages? If so, what were they?
4. Did the software crash at any point? If so, when? (i.e. during deployment, while running, etc.)
5. Were images successfully uploaded to the ML?
6. Did the GUI classify your images and return their label in the expected format?

An example of how this process may work in a deployment environment is detailed below for an preprocessed coin classification task and a new casings sort task.

1. For pre-existing items and their corresponding labels that have already been trained through our Classifier model, we won’t need to retrain our model, so we can simply capture new images sequentially and classify them, with images that cannot be recognized and placed into an unknown category. Specifically, our algorithm will categorize all items that have a low confidence score below 80% as unrecognized. These unrecognized images will not initially be included in our measure of classification accuracy because there are many factors that may influence the ability of our algorithm to classify them. Some of these factors include system errors like mechanical breakdowns and images that are taken on the wrong side such as state quarters being placed tails side up when trying to categorize what state they belong to. Our solution is to give the user the option to either disregard improperly taken images, update new unclassified images with the correct label, or correct a label that was misclassified. Additionally, users can also check all the images that have been classified and mark whether they have been labeled correctly. Only misclassified images will be taken into account when calculating classification accuracy.
2. For classifying new items, the user will need to supply an image of the item and a label specifying what bucket to classify the item into. Preferably, multiple images can be taken for a single item at different angles and sides. For example, casings are mostly flat, so they each have a unique front and back side. It is crucial that an image is taken on the side of the casing that has text because text is a crucial factor in differentiating casings by year, model, and type. Assuming that the front side of a casing has text, then that means all the images taken of the back side of the casing must be either categorized under an unknown label or simply disregarded because no text was detected. This also will impact image classification success rate, so for each image we will just take the best matching label for an image and output that result. For example with casings, our Classifier will use image processing on a single image and augment data by rotating the image and filtering it to a consistent resolution and scale, to increase the chances of finding an optimal match. However, this means that it is the user’s responsibility to take an image of the appropriate side of the casing with text on it when trying to classify a new label, because our model will train to recognize the image the user takes, and trying categorize two identical items with different labels will interfere with the accuracy of our algorithm. Therefore, it is crucial the user does not mislabel items during the training process, and also important for the user to only label items to the extent that it can be uniquely identified by some distinguishing factor like text when training a large group of similar items. Finally, the capture of trained images will be the same as in example 1.

We calculate classification accuracy as a function of how many items were not misclassified / all items that should have been classified correctly. While the ultimate goal is to reach 100% classification accuracy and correctly place every item into its corresponding labeled bucket, it is unrealistic given the nature of classification problems and variety of factors that can influence a ML model like image quality, sample size, bucket size, how specific labels need to be classified, similarities between items, and so on. Therefore, we will allocate a buffer of 20% when it comes to classifying items to account for the fact that our algorithm will always be improving in ‘real’ world systems and consider a classification accuracy >80% as a success. This buffer was chosen based on estimates from testing simulated environments with simple classification tasks like differentiating between different types of coins, and may increase or decrease for more compute intensive and detailed classification tasks like differentiating between casings. It is also important to note that the accuracy of our model will vary with the amount of input data given, so there needs to be at least 10 items classified per label in order to assess validation accuracy. We also intend to account for the case of overfitting of training data by implementing a cap of 500 images that can be labeled under a single bucket, which allows us to be more consistent in gauging the expected accuracy of our models given that we want to classify items quickly with small testing sample sizes.

# 5. Delivery and Costs

## 5.1 Delivery Manifest

* Our iPhone App will run with minimal bugs, errors, latency, or crashes while effectively controlling Raspberry Pi.
* GUI implementation that’ll prompt the user on controlling Raspberry Pi.
* Raspberry Pi functionality to take and upload images to run model classifying those images, providing displays that’ll prompt a conveyor belt.
* Classify mode will correctly identify and extract textual information for bullet case head engravings and a variety of U.S. minted coins >= 95%.
* Train mode will allow the user to input pictures and contextual information regarding new objects for which our model has not yet been trained on.
  + Using this mode will render a significant increase in accuracy when using Classify mode on newly trained parts.

## 5.2 Intellectual Property Statement

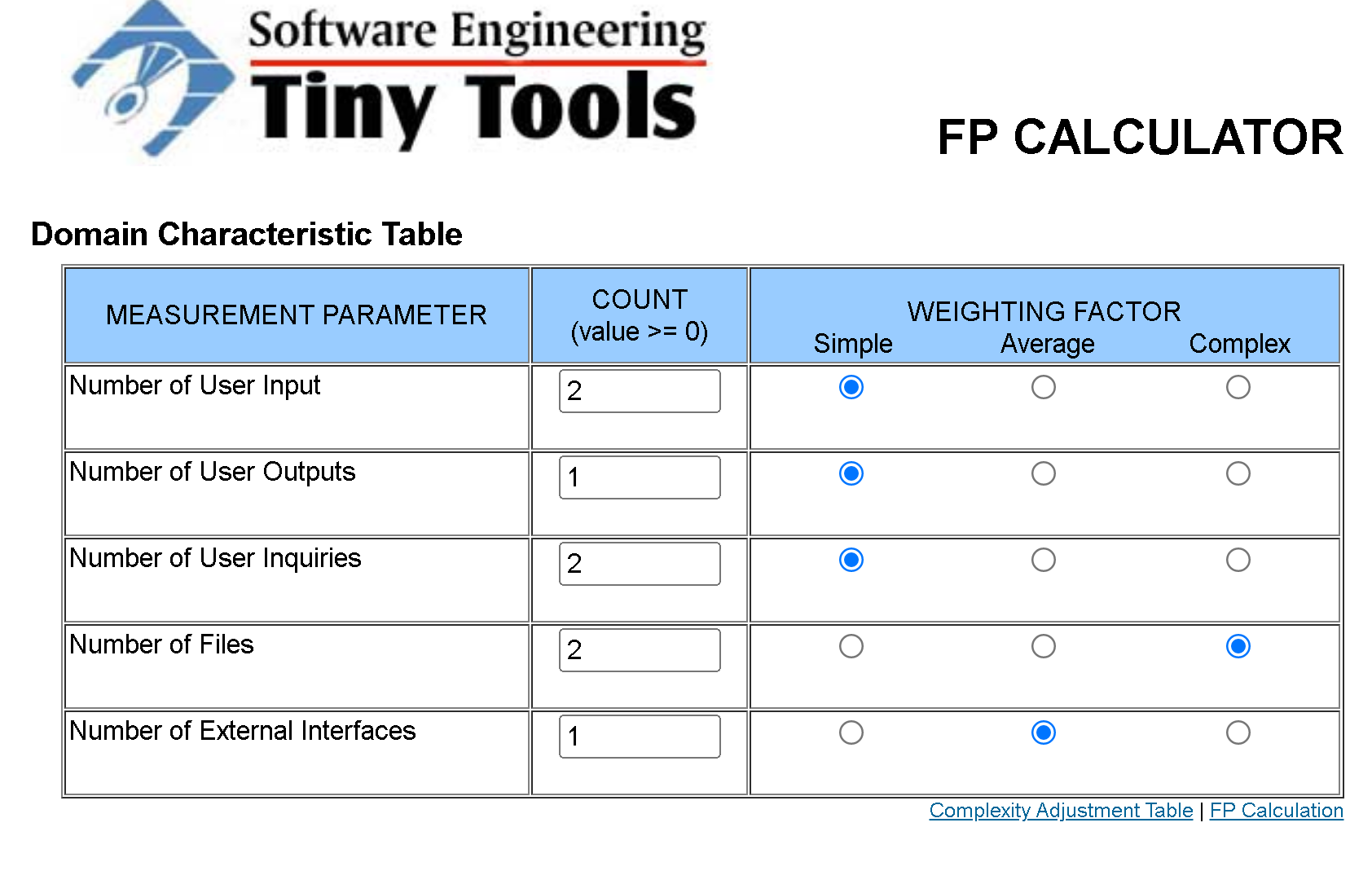
No data about the customers using this application will be stored including all forms of user ids. In addition, any data including proprietary information that the customer uploads into our databases for training our Classifier will be deleted as soon as the customer terminates our program. Therefore, customers can rely upon being completely anonymous while using our product. Finally, it is the responsibility of the customer to save their progress before terminating our program if they don’t wish to lose their custom trained models or labeled databases.

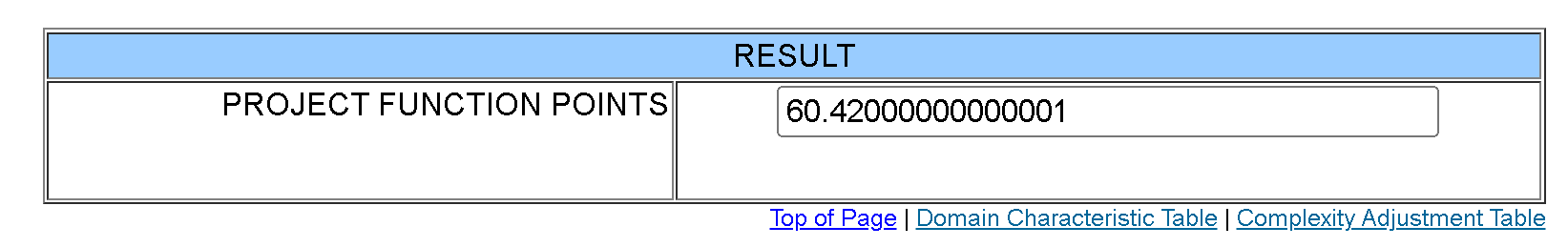
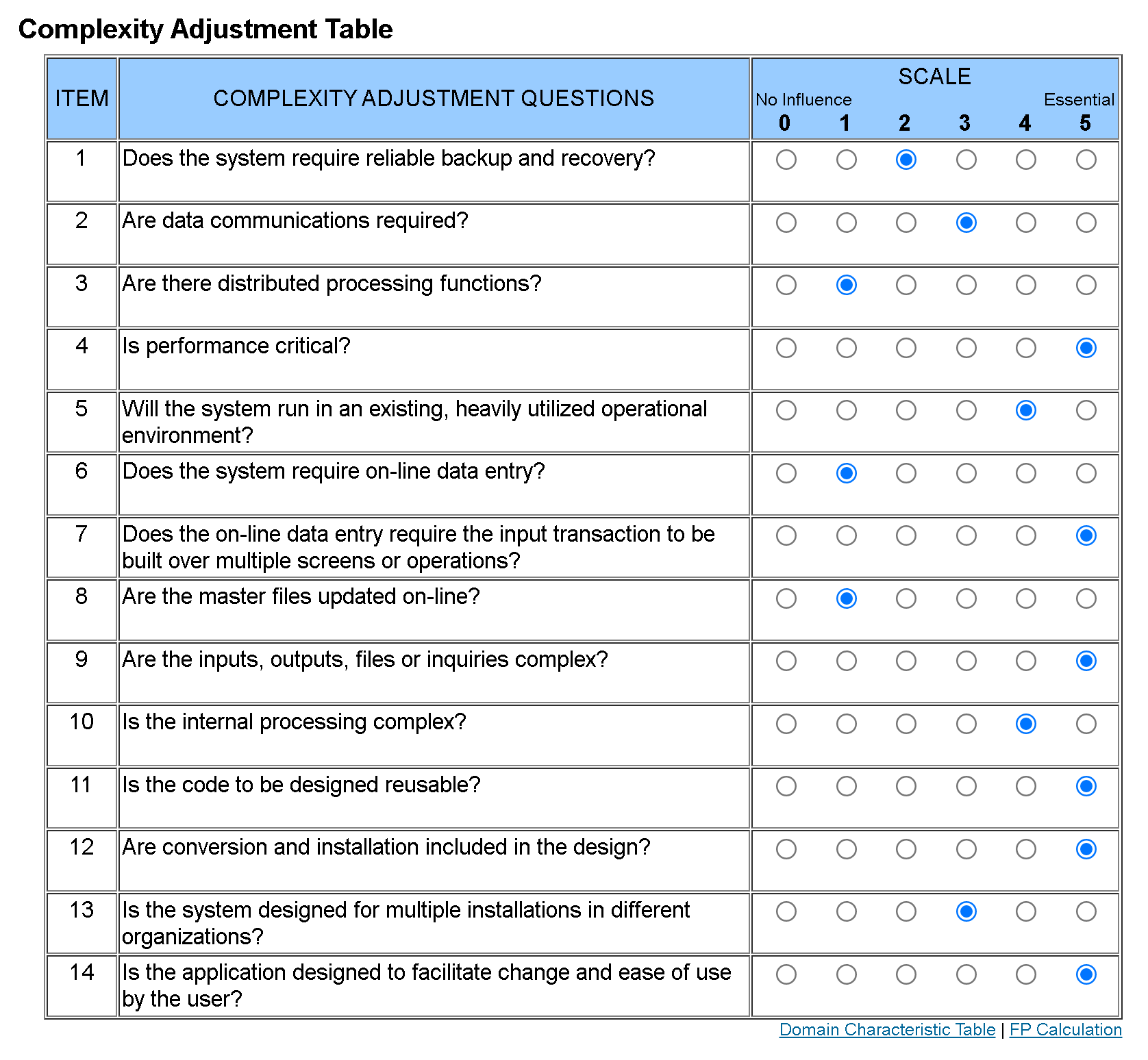
## 5.3 Cost Estimate

For estimating the cost of our project in the following section, we used function point analysis to estimate the number of project function points our final product will require. In order to use the function points for estimating man hours, we then calculated how many lines of code our project will likely need based on the function points we found and the type of project we have.

**Step 1**: We used a function point calculator to estimate the total function points of our

project.





This resulted in an Unadjusted Function Points estimate of 60.42. Next, we reasoned that our Classifier will mostly use configurations and train on top of existing models so we won’t need too much coding, so we estimated a minimum of 600 lines of code for our final product based on our initial function point estimate of around 60. After getting an estimate of the lines of code for our project, we were able to use the COCOMO Model referenced from the University of Kansas to compute a basic estimate of person-months during the construction phase of our project. We did this by assuming that our project is an organic type since our project is being done on a small scale. By the assumptions made above, we had all the variables needed to complete the organic COCOMO model formula, and found that we need around 3.2 \* (0.6 ^ 1.05) = 1.87 person-months to complete our project. Since there are roughly 20 working days in a month, we can convert person-months to person-days by multiplying 1.87 by 20 which equals 37.43 person-days. Since there are 8 hours in a work day, we need around 37.43 \* 8 = 299.5 person-hours or man hours to complete our project. Since we have 7 people working on our team, the work divided equally will be 299.5 / 7 ≈ **43 hours per person**.

For financial cost, we are planning to buy a raspberry pi and camera which will be **$100 bundled.**

## 5.4 Cost Estimate Divided By Task

We further divided our project into 10 main tasks, and categorized them under 3 difficulties: easy, medium, and hard. On average, tasks will take 43 hours to complete, but the actual expected hours will vary based on the difficulty level assigned. Easy tasks are expected to be completed faster compared to other tasks, need only a single person to complete and are estimated to take 15 hours to complete. Medium tasks will require a few people to complete, and take the estimated 43 hours. Hard tasks are expected to be a lot tougher and require most of the group working together to complete, so we estimated they take around 71 hours to complete. We reached these estimates by categorizing 3 tasks as easy and 3 tasks as hard, so their hours will average out at 43 hours, since (15 + 71) / 2 = 43.

**Tasks**

1. Developing the iOS app with SwiftUI (hard)
2. Developing macOS app with SwiftUI (hard)
3. Developing GUI (medium)
4. Setting up Raspberry Pi (easy)
5. Setting up AWS S3 (easy)
6. Setting up AWS Amplify (Pub/Sub, Datastore, Lambda Functions) (easy)
7. Writing/optimizing an image pre-processing script in the AWS Lambda (medium)
8. Testing the application with various datasets (medium)
9. Creating a simulated mechanical system (hard)
10. Testing the simulator (medium)

## 5.5 Timeline

|  |  |
| --- | --- |
| **DEADLINE** | **TASK** |
| Mar 18, 2022 | Proposal |
| April 1,, 2022 | Coin Classifier Model |
| April 8 2022 | Mobile Application |
| April 15, 2022 | API For Conveyor Belt |
| April 22, 2022 | CDR |
| May 8, 2022 | Acceptance Testing |
| May 10, 2022 | Deployment |

## 5.6 Advertising Plan

Classifier offers an API that will automate the process of recognizing the parts of an object within an image. It is specifically designed for users who need to integrate a flexible API with the goal of sorting, filtering, or searching from a group of parts. It will allow Professor Purtilo to test a conveyor belt system with a bucket of spare parts and categorize the various types of nuts and bolts. In addition, the professor can utilize the API to filter for unrecognized spare parts by automating the process of classifying unrecognized objects.

Our plan for advertisement integration is to present a quick overview of our final product on a webpage by hosting it from a virtual machine. Our advertisement will have overview pages describing the functionality of our product, with descriptions of how the automation of classifying parts of an object can be extended from a hardware system using conveyor belts to larger more sophisticated setups for sorting parts, and a page with images showing the results generated from our image classification models. Within the overview page there will be navigation links to a setup walkthrough, and an information page. The setup walkthrough link will provide instructions on how a user can train their own image classification models and update the existing model with images of the objects they want to be able to classify. Additionally, there will be instructions attached on how to install the needed dependencies for developer testing and there will be links provided for all relevant documentation and download links. The page with images will emphasize the various benefits of using Classifier and illustrate examples of how users could use the Classifier to solve specific small object classification problems with results that are output from our trials.

# 6. Milestones

## 6.1 User Interface Development

Create an interface that will allow the user to switch between the different classification modes, i.e. sorting, image detection, labeling new images, filtering by image, etc.

## 6.2 Coin Classification Software

To serve as a basis for image classification models, we tested various coin classification software models on classifying different US coins (dollar, quarter, nickel, dime, and penny). As a result, we are narrowing down the performance ability of different ML solutions and evaluating the tradebacks between the different approaches to building an accurate coin classification model that will ultimately be advanced enough to read text off from coins so that quarters can be classified by the state or year they were made in.

## 6.3 Conveyor Belt

To figure out the most optimal resolution we need each image of a part to be in order for ML to accurately classify, we need to get a simulated conveyor belt system running with a camera attached to it so we could get probable issues like lighting and positioning out of the way early. To mimic the conveyor belt, we’ll have a display controlled by a Raspberry Pi to display commands prompting the belt (‘move to next item on belt,’ ‘place item in bin X,’ etc.) The purpose of this conveyor belt is to position each object one at a time in order for the camera to capture the object and analyze the image. Additionally, if we are able to make the conveyer function similar to a green screen, we could potentially isolate an item through colors, or preprocess the images through a standard filter.

# 

# 7. Press Release

Part Classifier: A Classification System for Users to Sort Through Their Own Set of Small Objects

Team Classify is designing a method to allow users to sort their own set of small objects using machine learning models and various computer vision techniques. This application has been designed with the consumer in mind and opens up access for non-tech people to reap the benefits of today’s advanced machine learning classifiers.

We plan to release a software package for a Raspberry PI + Camera setup as well as an iOS monitoring application. Our program will have the ability to classify new objects and train models. This product aims to provide users with a simple user interface with built-in classification modes and features to create unique and personalized models for different user goals.

The final product will bring about significant change for those with large collections of small items that still carry value, use or important information. The following are some examples of objects that this model is optimized for: circuit components, coins, bullet casings, trading cards, batteries, Lego’s, etc.

Small items are often difficult to keep track of. Through the use of this application, non-tech users will finally be able to organize their shelves, find valuable needles in their haystacks or extract vital information out of small objects in an efficient manner. The secondary market for collectors’ items is becoming an increasingly larger market, this model will allow you to sift through a large bucket of unknown objects to specifically find items you specify.

If this sounds like something you will use or if you have any questions about the product, feel free to reach out to our team!

Contact us at: saaleali@umd.edu