Part Classifier

Team Classify

Sanjay Gururam Sharma

Kevin Song

Nicholas Schaap

Nathanael Temesgen

Syed Aal-E-Ali

Anthony Yao

Andrew Berger

March 13, 2022

Table of Contents

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 be plugged into a mechanical system via an API. 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 (see Figure below).

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

This problem space is particularly difficult and new because traditional OCR algorithms for recognizing machine-printed text on documents don't tend to perform too well on these applications. Due to the limitations of our environment, we will be working specifically with quarters. 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. Another area of focus that is derived from the client's needs is to create scalable software and easy-to-use APIs that can be extended for other use cases. For instance, we hope to create a flexible API that can be extended from simple conveyor belt setups to more complicated image classification systems.

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 fast 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.

4. Ease of Use - The final software goal of Classifier is to abstract the software component of our model with a simple to use API so that users who aren't familiar with programming can still operate the training and labeling of new data, as well as understand the results returned from our model.

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

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 & OpenCV*. AI & Machine Learning Blog. Retrieved March 12, 2022, from https://nanonets.com/blog/ocr-with-tesseract/

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 an iOS application. This app will allow the user to create as many labels as they seem fit and then take multiple pictures of an item to train our algorithm into recognizing that item from a gallery of images. After our model has been trained, the user would then be allowed to take pictures of any item, and if the user has trained our algorithm into recognizing that same item, our application will be able to properly classify the item and place it in a specified bin that has its corresponding label. If the item isn't recognized (hasn't been trained, unrecognizable due to rusting/damage, etc.) that picture would 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 would 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 this solely runs on an iPhone, all of its features will lie in our application – which is specified in the following subsection [2.1.1].

2.1.1 iOS Application

- Camera

  Having access to the user's camera, our application will use the iPhone's camera to take pictures of items to either train our algorithm or classify the item into a bin.

- '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, a list of all of the images that our algorithm classified to be of type <label> will be displayed. There'll also be an 'unrecognizable' bin where all of the images that our algorithm couldn't classify will be stored in.

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 would 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. The program will be available to download via the Apple AppStore. 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. We will provide documentation for an API that any microcontroller will interface with. This microcontroller 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, camera/phone setup. Bin setup with servo and motors.

The minimum hardware requirements to run Swift 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 should be adjusted to accommodate consistent imaging. It is important to avoid oddly shaped shadows or bright flashes interfering with the imaging area. The iPhone 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 would suffice). Lastly, open up the Part Classifier Application.

2.2.3.1 Camera Distance from Object

The distance between the iPhone 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 iPhone 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. This process 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 should 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 would 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. In addition, because this project is being developed in Swift, it is necessary that users MUST have an iPhone in order to use this application off of the App Store.

2.4 Ideal Mechanical System and Assumptions

Ideally the entire sorting system would include a hopper, a dropper, a separator, a conveyor, a camera, a sorter. The hopper would 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 would 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 would 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 would allow us to simply use a timed camera where there is an occasional case of having no object in the frame. Our program would 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 should be sent to the sorter that would 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.

3. Software Development Requirements

3.1 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.
    a. Tesseract-OCR engine for reading text
    b. Template Matching to supplement model predictions
    c. SIFT algorithm for extracting key points from an image
2. Models/ libraries we will utilize:
    a. CoreML - Apple's machine learning library specifically designed to operate on Apple devices such as the Iphone, Mac and Ipad.
    b. Mask RCNNs, a modern computer vision model applied to smaller regions with an image for more accurate results.

3.2 Mobile App Design

Our app will be the center of our user-facing experience. The mobile app portion of our system will leverage the iPhone camera to capture images of objects and process the images by feeding them into our algorithm. The app will also be synced with our realtime

database such that the state is publicly accessible by various other realtime systems that want to make use of the classification results in real time.

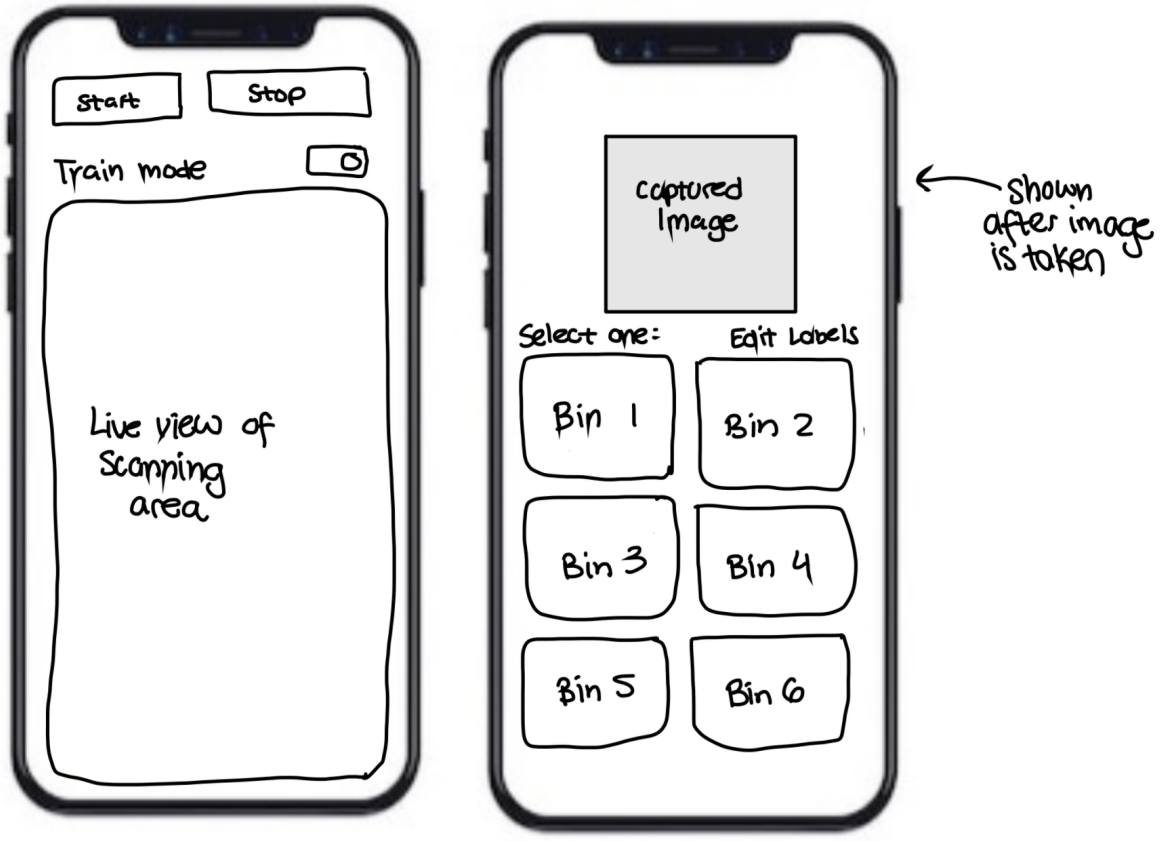

Figure 3.2(a)

We will use Swift/ Swift UI to build the UI of our app and connect to on-device hardware.

3.3 Database Design

To connect/sync all the various components of our end-user system (iPhone app, 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
    - 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. Would require the addition of another database counterpart to store historical data.
    - Pro: Comes prepackaged with Swift SDKs

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 80% 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 an iPhone onto a table where the camera is facing down and have a sheet of coins in a single-file line motion under the phone where pictures will be taken. 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 phone so that an item will be displayed to the phone, classified, then move on to the next item that's on the belt.

5. Delivery and Costs

5.1 Delivery Manifest

- [ ] Our iPhone App will run with minimal bugs, errors, latency, or crashes.
- [ ] API with documentation that anyone with a mechanical system will be able to interface with via a microcontroller.
    - [ ] This API will communicate information about the results of our model run on a given object as well as settings for how fast the mechanical system should be moving.
- [ ] Classify mode will correctly identify and extract textual information for bullet case head engravings and a variety of U.S. minted coins >= 80%.
- [ ] 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 or session ids of the API. 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

Task = hours * people assigned

- [ ] Collecting data and images to be tested and labeling
    - [ ] 5 hours * 3 people = 15
- [ ] Running CoreML or RCNN on the data
    - [ ] 15 hours * 1 person = 15
- [ ] Developing a circular text identifying program

- ☐ 10 hours * 3 people = 30
- ☐ Developing a template matching program
  - ☐ 10 hours * 3 people = 30
- ☐ Creating a user interface
  - ☐ 5 hours * 2 people = 10
- ☐ Setting up a simulated mechanical system
  - ☐ 2 hours *2 people = 4
- ☐ Testing the model
  - ☐ 3 hours * 1 person = 3
- ☐ Optimizing environmental variables such as lighting and colors
  - ☐ 3 hours * 1 person = 3

Total man hours : = # of people on team * total hours they spent =
15+15+30+30+10+4+3+3 = 110 total man hours

5.4 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

5.5 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.

5.6 Milestones

5.6.1 User Interface Development

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

5.6.2 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.

5.6.3 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. 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.

5.7 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.

The team behind 'Part Classifier' is made up of software engineering students with strong backgrounds in machine learning, computer vision and application development. 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.

We plan to release an iPhone application along with an object classification and model training API that is suitable for use in a variety of larger conveyor-belt systems. If this sounds like something you would use or have any questions about the product, feel free to reach out to our team!

Contact us at: saaleali@umd.edu