Broad Institute

CellProfiler Analyst

- Yahiya Hussain, Yahiya.Hussain001@umb.edu
- Emma Kelminson, Emma.Kelminson001@umb.edu
- Bella Baidak, Bella.Baidak001@umb.edu
- Jing Yuan, Jing.Yuan001@umb.edu
- Emmanuel Adeniyi, Emmanuel.Adeniyi001@umb.edu
- Alex Abel, Alexander.Abel001@umb.edu
- Corey O'Connor, Corey.OConnor001@umb.edu
- Thouis Jones, Thouis@broadinstitute.org

Project Proposal

- **v1.0 02/26/2021**
- **v2.0 03/26/2021**

1 Introduction

1.1 Purpose

Modern Cellular-Biology is tasked with the challenge of analyzing tons of samples with many different parameters in both automated and manual experiments. This means a lot of data must be poured through to classify the cells by phenotype before any in depth analysis of their phenotype properties can occur. CellProfiler Analyst for the desktop is a tool meant to automate this task with classical machine learning. However its codebase is complex and decades old making it difficult and time-consuming to install for non-programmer users. To make this tool more accessible, a web-based app must be developed with a modern software architecture to make this tool accessible to more scientists and extendable by future developers. With this, cell-biologists could quickly label and classify cells by phenotype with minimal setup required.

1.2 Scope

The name of the product is CellProfiler Analyst for the Web (CPAW). Similarly to CellProfiler Analyst (CPA), this web-app will include an iterative user-driven training loop to train and refine the model to classify a given phenotype. The tool will include a drag and drop interface to allow the user to correct the algorithm to classify their phenotype of interest. A typical work session of a couple of hours with 20-40 iterations should be enough to result with a good classifier. The resulting TensorFlow model or training set used to generate it can be downloaded by the user for future use. The software will not classify based off of locational or well-plate data of the biological cell in the image, but will classify based off of relevant non-spatial cellular features. The software will not be responsible for extracting these features from images, but rather it will rely on the upload of feature-vectors from the CellProfiler desktop-app (CP).

1.3 Overview

Our goal is to develop a web-based app based on CellProfiler Analyst that assists scientific researchers by learning their visual phenotype classification through interactive active-learning. We will implement an intuitive drag and drop interface to allow the researcher to critique the machine learning model's predictions with ease. We will design the codebase with modularity in mind to provide the ability to later allow later work to flesh out the entire CellProfiler Analyst toolkit.

1.3.1 Existing Work

- **CellProfiler Analyst:**
  Desktop-app which Allows interactive exploration and analysis of data from high-throughput, image-based experiments.
- **Ilastik:**
  Desktop-app tool for interactive image classification, segmentation and analysis. Pixel Classification workflow offers a machine learning technique for segmenting an image based on local image features computed for each pixel.
- **CellCognition:**
  CellCognition Framework which uses image processing, computer vision and machine learning techniques for single-cell tracking and classification of cell morphologies.
- **WND-CHARM:**
  Desktop-app which is a multi-purpose image classifier that can be applied to a wide variety of image classification tasks.
- **Advanced Cell Classifier:**
  A data-analyzer desktop-app that aims to provide a very accurate analysis of cell-based high-content screens and tissue section images with minimal user interaction using advanced machine learning methods.

1.3.2 Limitations

The front-end will exploit features of all modern web browsers (i.e. Chrome, Firefox, Edge, etc.), and thus may struggle to have compatibility with old web browsers (i.e. Internet Explorer) for the variations in each browser's ECMAScript implementation. Graphics hardware may also be a bottleneck, as well as WebGL's limitations for performing the various machine learning operations. The complexity to access the client's file system will present challenges for storage of images and user data especially if we want to add persistent storage to this client-side only app.

1.4 Definitions

ECMAScript -> JavaScript specification that web browsers are tasked with implementing.
CP -> CellProfiler
CPA -> CellProfiler Analyst
CPAW -> CellProfiler Analyst for the Web

2 Requirements

- R1 - The software should be entirely web-based and client-side.
- R2 - The software must utilize an easy drag and drop user interface to train and correct a image feature classifier, similar to the existing software.
- R3 - The software must utilize CellProfiler output.
- R4 - The software must allow the user to download a finished model's weights or its training set.
- R5 - The software must be reactive to user-input and be easy to use

3 Specifications

- S1 - The software will run code entirely client-side on a website in the browser, and should run in most modern web browsers (ECMA 6), regardless of the underlying operating system being used.
- S2 - The user dragging and dropping cells between 3 boxes: unclassified, positive classification, and negative classification should communicate to the software the mistakes its feature classifier has made and to train and correct them for the next user iteration.
- S3 - The user must upload a folder containing data obtained from the CellProfiler desktop app: A dataset of images, a properties file, a SQL file of column names, a csv of image locating data, a csv of cell object features, and an optional starter text file of the training set of cells and labels.
- S4 - The software will allow the user to download a TensorFlow model.json file of the trained machine learning feature classifier specification and its weights and/or allow the user to download a text file of the training set used to construct the classifier.
- S5 - The software employs the React framework to organize UI/UX components along with various handlers that deploy asynchronous functions and promises to keep UI reactive. Through Web Workers, the process of handling the data and communicating with the user is sped up as well. Also, the handler maintains communications with all components and necessary modules to ensure a fast, easy experience for the user;

4 Design

The web application UI for CPAW will be implemented using the React.js UI framework, a library for creating websites with dynamic user interfaces using JavaScript (ECMA 6). The UI components will be supplemented by the React-based UI component libraries of Bootstrap and Material-UI. Clean transitions of the current software state in response to user input commands and various processes of the software will be managed by the MobX library, a simple and scalable state management system. The different options for machine learning feature classifiers will be implemented and trained using TensorFlow.js, which is a library for training machine learning models in the browser using WebGL GPU shaders. The CellProfiler input folder data will mainly be parsed by Papaparse, the robust and simple csv parsing tool, which will use web-workers to speed up execution where possible. The unit tests and integration tests on the source code will be implemented using jest and run automatically upon GitHub push changes using GitHub Actions. The site will be deployed to GitHub Pages for the time being, but it may be moved to a more private Broad Institute server.

4.1 Use Cases

Actors: Researchers at the Broad Institute and other users of the open-source software especially researchers studying microscopic biological cell features with florescent markings with high-throughput experiments.

Use Case: Train a machine learning algorithm by manually classifying cells by phenotype as positive or negative to come up with a classification algorithm to classify any number of cells at high-throughput.

Pre-Conditions: The actor is required to have the data outputs from the CellProfiler desktop-app. The user needs knowledge of the biological cell phenotype they want to train the algorithm to classify for. The actor is also required have to have an internet connection and a modern and up-to-date web-browser to use the web-app.

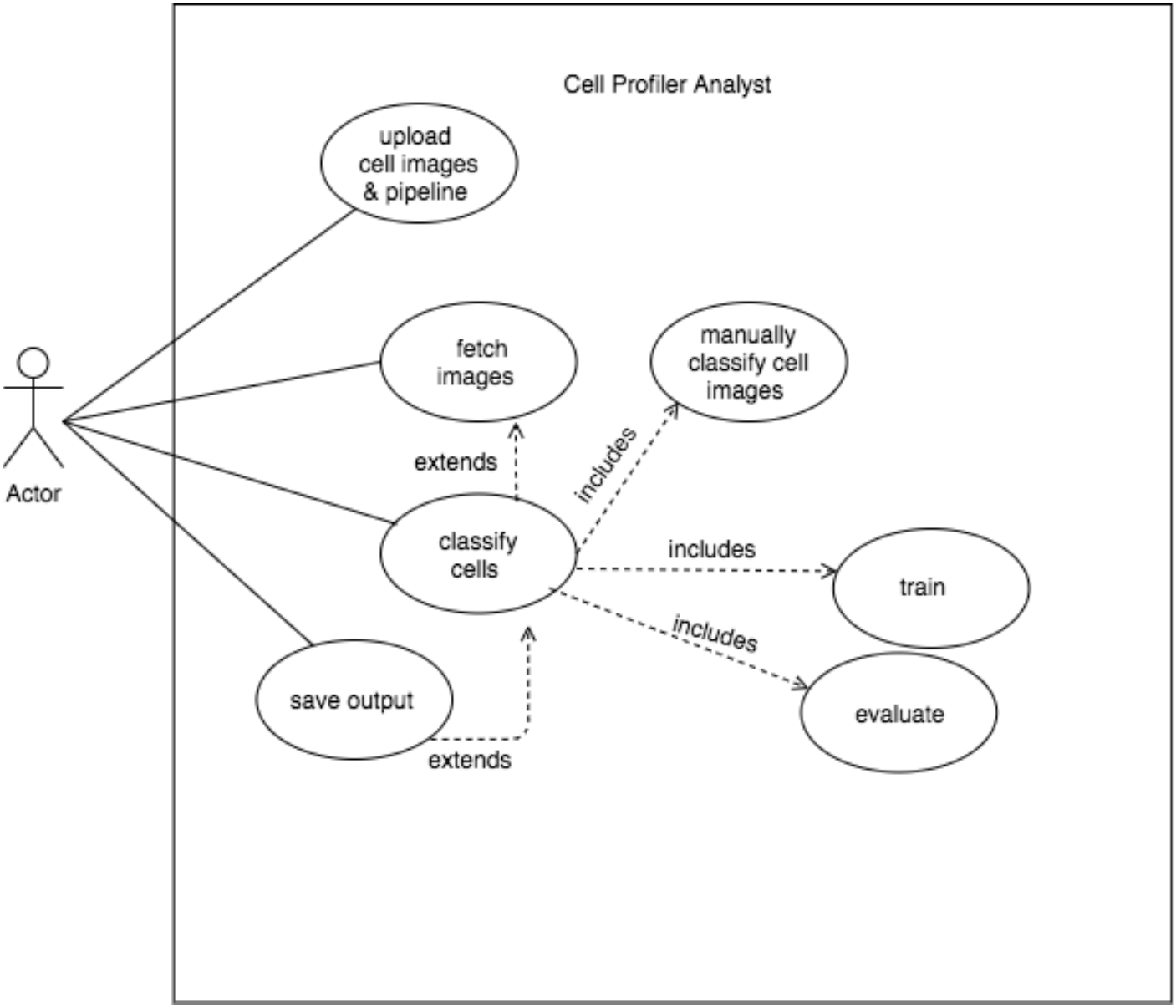

Figure 1: Use Case Diagram

After uploading the dataset, the average user of CellProfiler Analyst for the Web will have the ability to fetch cells of a certain phenotype,and to correctly classify cells to train the algorithm. After about 30-50 iterations (a couple hours of work) the user will be able to save the output as the parameters of the classifier its the training set.

4.2 Architecture

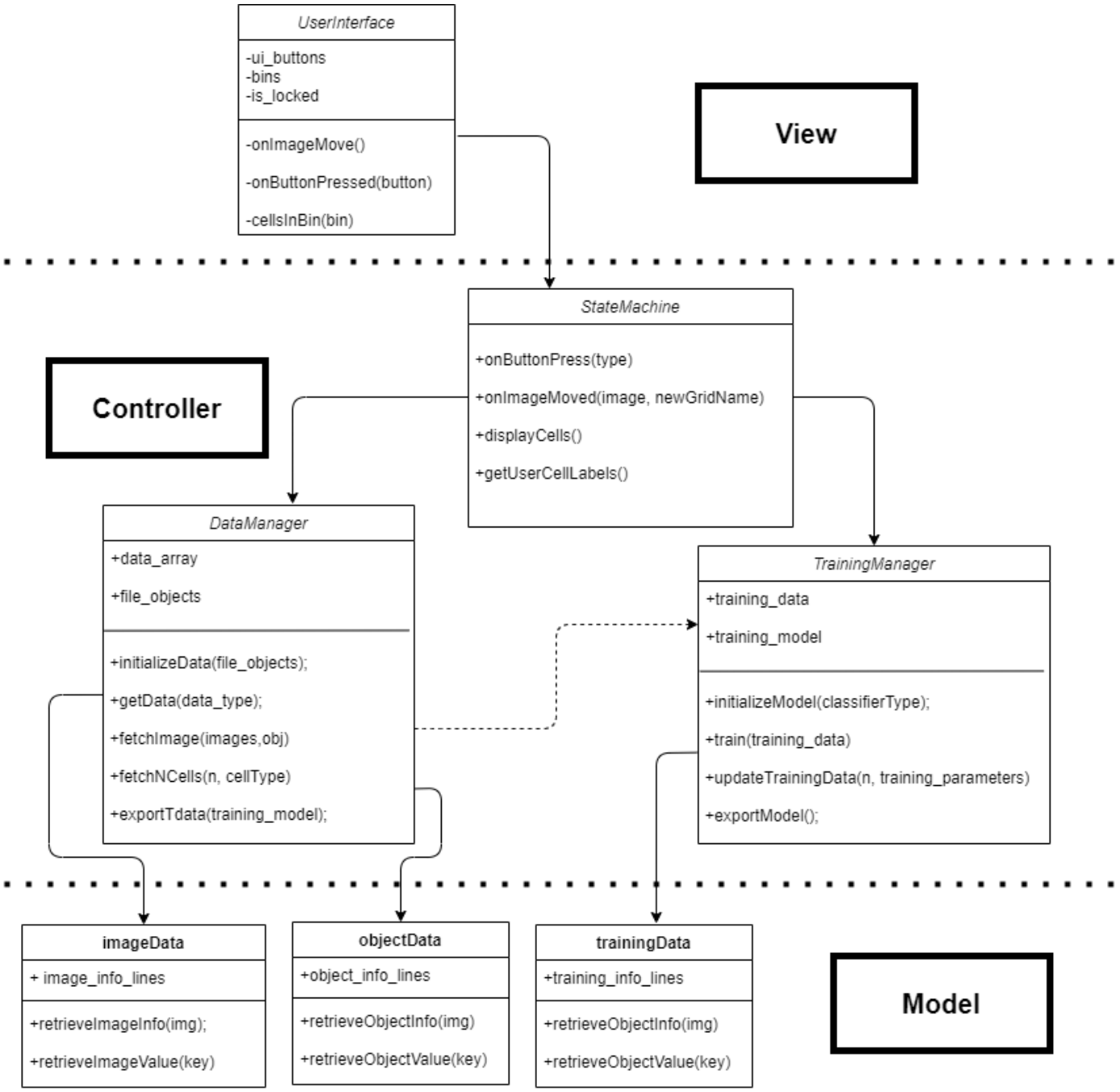

Figure 2: Class Diagram

The source code of CellProfiler Analyst for the Web is organized by the Model, View, Controller architecture. The UserInterface class is tasked with handling user input and UI layout. The UI sends calls based on user actions to the StateMachine to cleanly handle interactions in the program's core. The StateMachine handles communication between the UI, DataManager, and TrainingManager to ensure proper execution. TrainingManager manages the training data and the use of the classifier. DataManager handles the image and cell feature data and any interactions needed to communicate this data to the rest of the program.

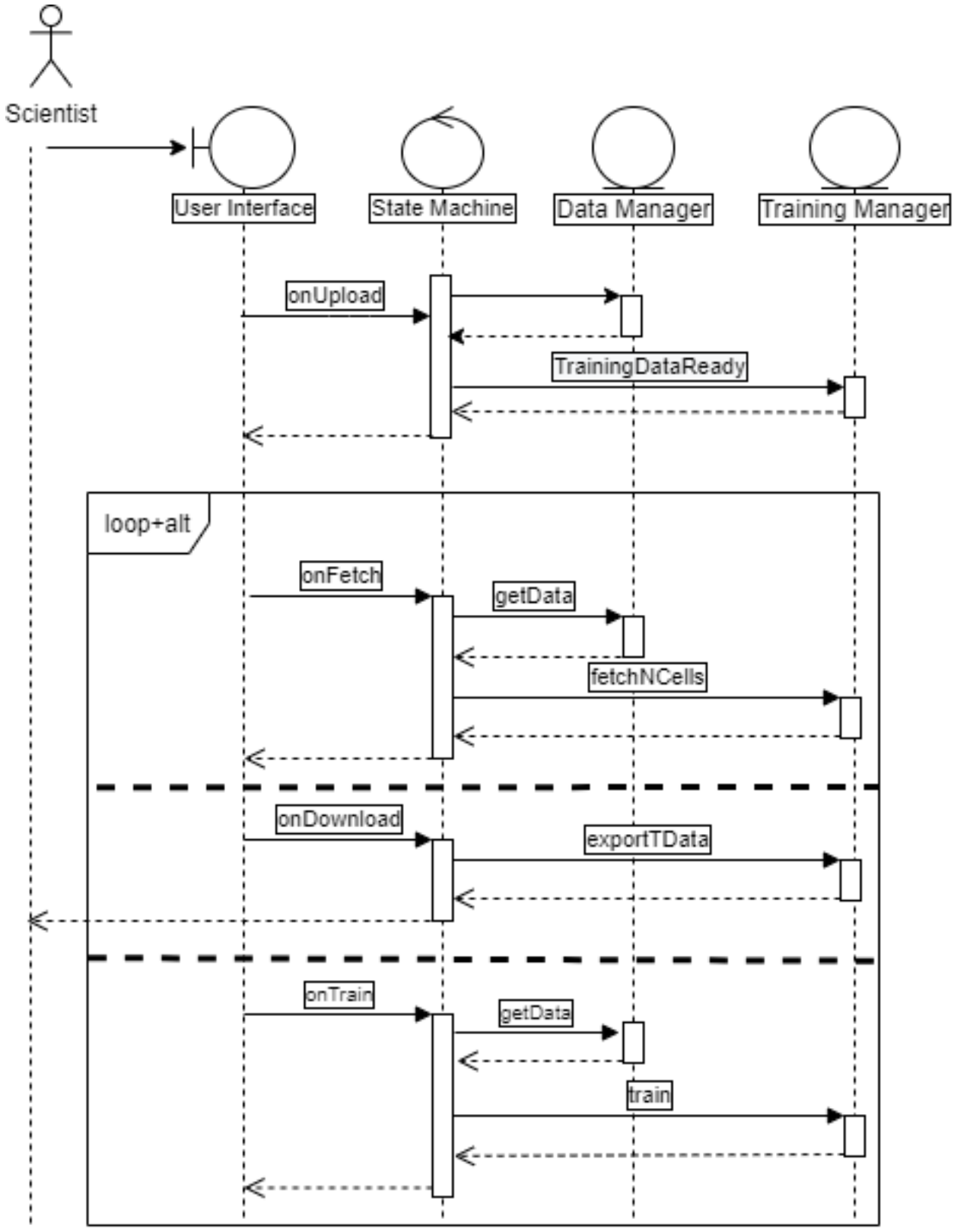

Figure 3: Sequence Diagram

The user first uploads files and the UI calls the State Machine to initialize the data and classifier trainer. After data is initialized the other buttons are enabled and the user can now participate in an iteration loop to train the classifer. Each iteration allows the user to choose between different actions: to fetch the desired cells to present to the user for critique, train the model based off of those critiques, and to download the resulting model specification or training set.

4.4 Technologies and Implementation Details

The web application UI will be implemented use React.js with an object-oriented design using JSX components. We will augment the project with a wide array of JavaScript libraries for extending UI features of React such as Material-UI and Bootstrap. HTML5 base components such as the file input tag, the download tag, and the drag-and-drop div properties will be used to add greater ease in interaction by the user. The state machine management for the logic will be implemented using Mobx-react, which is a simple and a scalable state management library with bindings for React.js. There will be an array of classical machine learning models to choose from to construct the classifier including: Logistic Regression, Random Forest, Support Vector Machine, K-Neighbors, and Fast Gentle Boosting. Each classifier will be implemented using TensorFlow.js, which stores data in an optimized tensor format and implements machine learning training and fitting in the browser using WebGL shaders. The entire web-app will be client-side and will be hosted on GitHub Pages and possibly moved to a private Broad Institute server. The sourcecode will be stored, unit-tested, and deployed from Github using the Github Actions and Github Secrets services.

4.5 User Interface

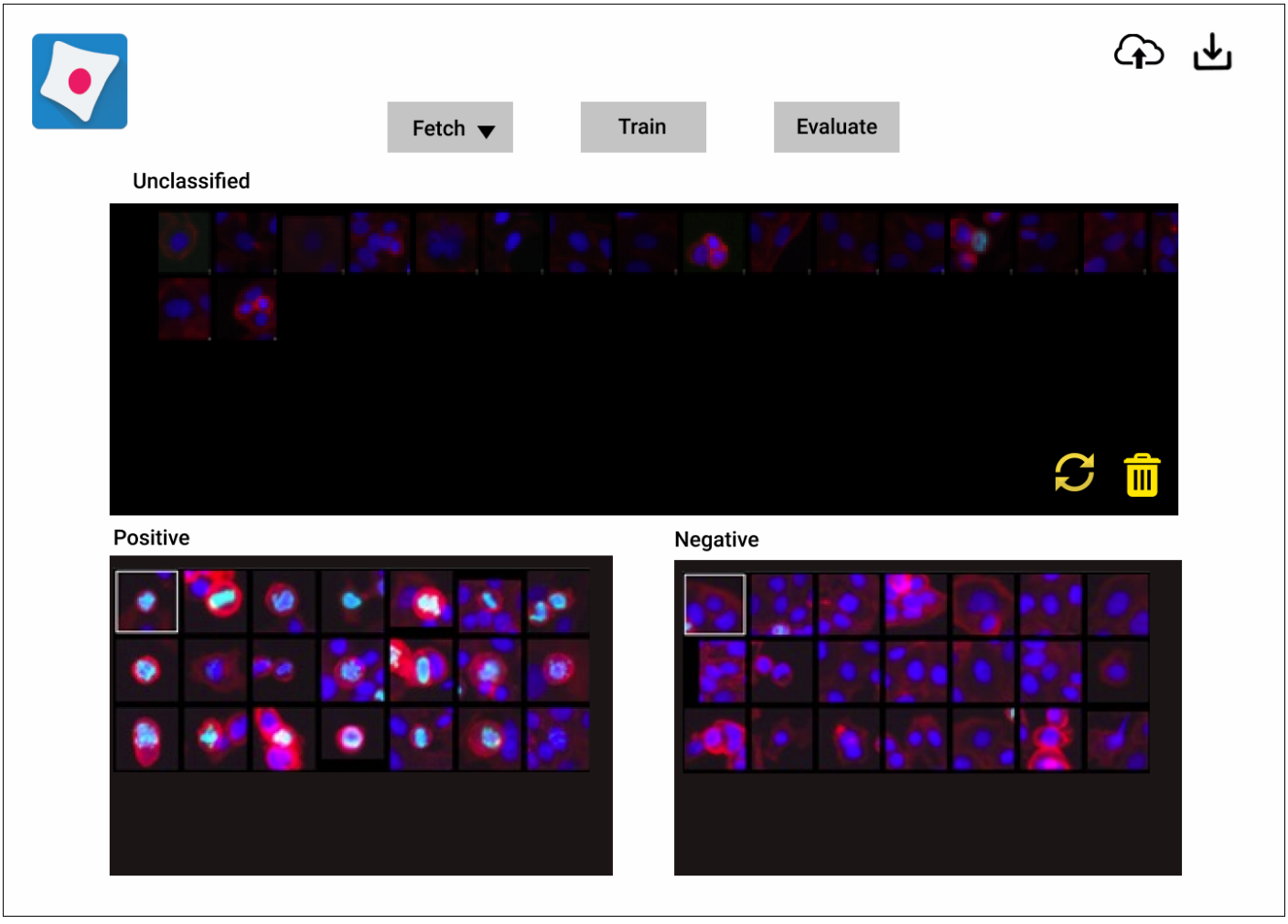

Figure 4: UI Wireframe

The CellProfiler Analyst for the Web has a similar UI to the original CellProfiler Analyst. It includes the ability for the user to fetch a fixed number of cell images that it identifies as positive, negative, or random classification. The user can then detect mistakenly classified images and drag them into the correct bin to train the model. After some iterations, the user can download the machine learning model JSON spec and weights or download the training set.

4.3 Workflows

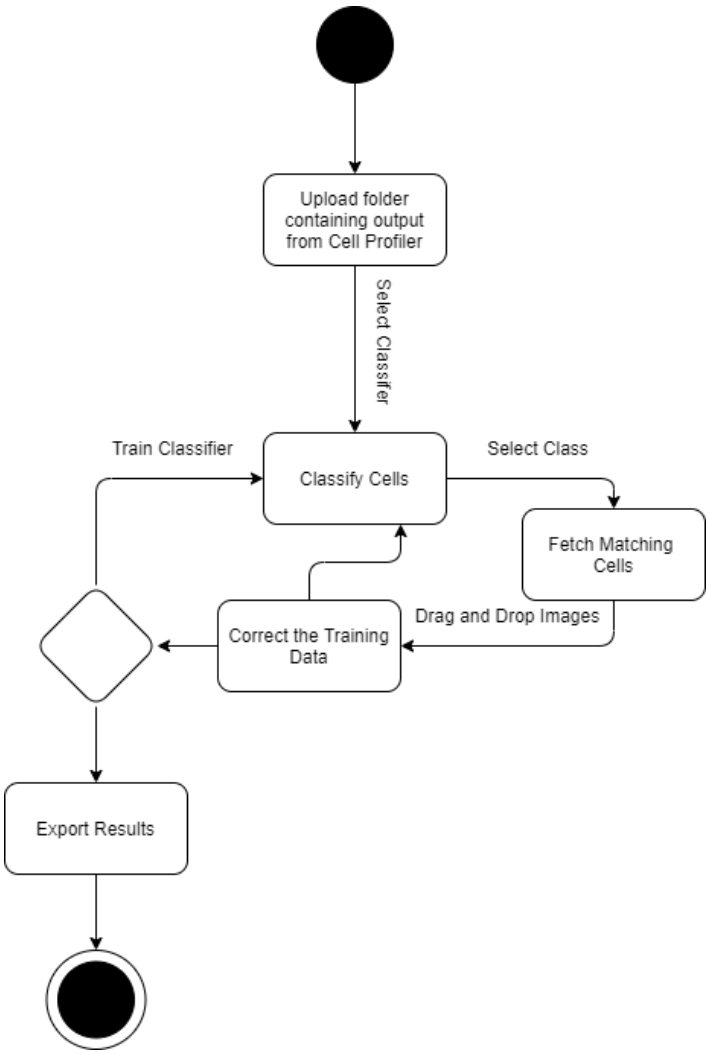

Figure 3: Workflow Diagram

CPAW allows the user to upload a dataset outputted by CP possibly with a starting classifier training set. Upon the upload the user can select a classifier to train. The user can then iteratively train the classifier by fetching images of the predicted phenotype and telling the software where it has made a mistake with drag-and-drop actions. At the end the user can export the classifier specification or its training set.

User Interface Features:

- The user can use the upload button on the upper right corner to upload the outputs of CP including the cell images they want to train the classifier to classify.
- The user selects from the fetch drop down menu the type of cell images they want to upload; positive, negative, or random and it fetches the relevant cells.
- The user then drags cell images from the unclassified bin and drops them in either the positive or negative class bins to train the algorithm.
- When the user runs out of images in the unclassified bin they can press the refresh icon in the bottom right corner of the unclassified bin to get more images of the selected type (positive, negative, or random)
- If the rest of the unclassified images are unable to be classified or are redundant to do so, they can be deleted by clicking the trash bin button in bottom right corner unclassified bin
- After manually dragging and classifying 30-40 cell images, the user can press the 'train' button to train the classifier
- The user can evaluate the accuracy of the training model by selecting the 'evaluate' button and viewing an accuracy summary
- When the user approves the accuracy of the model they can save the classification output by selecting the save icon in the top right corner of the browser

5 Timeframe and Milestones

- **Proposal Rough Draft: 2/26**
- Strip Down the core features of CellProfiler Analyst (CPA) and analyze as a Jupyter Notebook: 3/5
- Implement a crude UI for core features of CPA in React.js: 3/12
- Reimplement different features of CPA in isolation: 3/19
- **Proposal Final Draft: 3/26**
- Cleanly polish the Minimum Viable Product of CPA for the Web: 4/2
- : Optimize performance and user experience features with promises and web-workers 4/9
- (stretch goal) Implement CellProfiler with a modern machine learning algorithm for image segmentation: 4/23
- Implement UX feedback from user studies with actual potential users of the product: 4/30
- **Project Presentations: 5/3**
- **Project Documentation: 5/21**

6 Supplemental Documents and Notes