Spring 2019

LSST Classifier

David R. Carbonari
carbonar19@up.edu

Maggie Ryan
ryanmar19@up.edu

Branden N. Vennes
vennes19@up.edu

Mikayla K. Whiteaker
whiteake19@up.edu

LSST Classifier Final Report

David Carbonari, Maggie Ryan, Branden Vennes, Mikayla Whiteaker
Advisors: Dr. Dvorak, Viet Phan, Dr. Katherine Kornei

Introduction

Project Description

The Large Synoptic Survey Telescope (LSST) is currently under construction in Chile. Financial support for the LSST comes from the National Science Foundation, the Department of Energy, and private funding raised by the LSST Corporation.[1] It is scheduled to begin full operations in 2023. Unlike most telescopes, which only focus on one portion of the sky with extreme detail, the LSST will conduct a deep survey of a large portion of the sky. The 10-year survey of the sky planned for the LSST will deliver 200 petabytes of images and data products. This data will be used to achieve several science goals including searching for various indicators of near earth asteroids, looking for items in the deep solar system, discovering the structure of the Milky Way, and the study of dark matter and dark energy.[2]

While the telescope is still being constructed, the software architecture that will support operations has already been developed. This project made use of the LSST Data Science Pipeline, through which LSST data is accessed and manipulated. In addition, mock data has already been made by the LSST team and released to test prototype analytical software.

This project used the LSST Data and software to build an Artificial Intelligence (AI) to classify sources in LSST test images. The LSST Data science pipeline labels each light source in the image as either a star or extend source (not a star). Our neural network AI was developed to predict one of these two classifications. A web application (app) was also constructed with the intention to use crowdsourcing to create additional training sets for the neural network in the future. The app displays a LSST image and asks users to classify a marked source within that image. No astronomy expertise will be required to use the app as every user will be trained to classify data via an introductory tutorial. Using crowdsourcing and elements of gamification we can engage a diverse group of users with the LSST science goals.

[1] See Appendix A

[2] Telescope, Large Synoptic Survey. "About LSST." LSST. Accessed September 27, 2018. https://www.lsst.org/about.

Implementation Overview

The project has been implemented in four parts, using the LSST pipeline, AWS storage and API, a Convolution Neural Network, and finally an Angular web app. The LSST pipeline environment is used in conjunction with a Python programming language script, which makes use of LSST libraries, to calibrate and upload LSST images and sources (light sources within images) data to Amazon Web Services (AWS). LSST images are stored in AWS S3 Buckets and the source data is stored in a DynamoDB table. In order to access this data from the web app an AWS API was created. The Convolution Neural Network ingested LSST images and learns to label the sources in the images. Finally, the Angular web app was built to display LSST images and train users to label sources in the images.

The greatest challenge faced was a lack of expertise in astronomy that had to be overcome. A great deal of research had to be done throughout the project. We were surprised by the amount of continued work that had to go into research in astronomy related topics. In order to understand how to use the LSST pipeline the team had to learn basic astronomy and optics jargon. To better understand how to use the LSST data, the team had to do extended research on Flexible Image Transport System (FITS) file formatting and how telescope architecture influences image data. Dr. Katherine Kornei advised the team and help fill in gaps in astronomy expertise, specifically in concepts of photometry and telescope architecture.[3] The team also faced challenges in correctly formatting data for the neural network and successfully learning to classify stars.

There were three big milestones that the team achieved over the course of the project. Firstly, extracting correctly calibrated FITS files and source data from the pipeline. This was a critical step in achieving the functional requirements of the project. The second, was training the neural network on actual LSST data. While there were many challenges faced in getting the neural network to learn to identify sources in LSST data, being able to training with it was a huge achievement and a culmination of months of research. Finally, the third biggest milestone was having LSST images displayed in our front-end web app.

[3] Dr. Katherine Kornei is an astrophysicist who is the Program and Exhibit Developer at the Oregon Museum of Science and Industry and works as the Space Science Content Expert for the NASA-funded Space and Earth Informal Science Education project. Dr. Kornei met with the LSST Classifier team in a consultant role.

Assessment

Assessment of Functional Requirements

An objective assessment of the success of the project can be measured by comparing the outcomes to the functional requirements.[4]

LSST Pipeline Requirements

Install Science Pipelines: The LSST pipeline and supporting software was successfully installed on the team's lab computer.

Setup Pipeline Environment: The pipeline environment was successfully setup and remained functional throughout development lifecycle.

Calibrate Data: The sampled data provided in the ci_hsc package was successfully calibrated, coadded and forced photometry applied.[5]

Process Data: A Python script was written that successfully uploaded calibrated fits files to the S3 bucket , and puts all the sources, their labels, and location, in each image into the database.[6]

Other Image Sources Requirements

Collect Hubble Space Telescope Images: A Python script was created the successfully scraped images of comets from the Hubble space telescope. However, we did not end up using these images. We pivoted to classifying stars instead of comets, we realized classifying comets would be infeasible after much research when it became apparent there is not a sufficient data set of comet imagery. Thus we only ended up using the LSST data.

Amazon Web Services (AWS) Requirements

S3 Storage Bucket: An S3 Bucket was successfully created that holds all of the FITS files needed for the project.

AWS Database: A DynamoDB table containing all our source data was successfully created.[7]

[4] See Appendix C for a detailed outline of the original functional requirements
[5] The ci_hsc package can be found here: https://github.com/lsst/ci_hsc. This github project provided by LSST contains all of the sample data used, and scripts that provided support in calibration of that sample data.
[6] An Amazon S3 bucket is a public cloud storage resource available in Amazon Web Services' (AWS) Simple Storage Service (S3), an object storage offering
[7] Amazon DynamoDB is a fully managed proprietary NoSQL database service that supports key-value and document data structures and is offered as part of the Amazon Web Services portfolio.

Web-Facing API: The AWS Gateway API was successfully created with a lambda function to get a random source and its associated information.

Pipeline Library: A library of functions were created using boto3 so that users (in the pipeline environment and in the neural network), could upload and retrieve data from both dynamoDB tables and S3 buckets.

Web Application Requirements

User sign-in: This was never fully implemented, as it was deemed as not integral to the project, so in the interest of time it was permanently put in the backlog. While a functional login form exists, it is not connected to a secure login database. This functionality would be necessary if in the future the web app was used in a crowd sourcing capacity, in that is was recording user data.

User Experience: The webpage is easy to navigate as it has few buttons and consists of one scrolling webpage. However, it did not undergo user testing due to lack of time.

Responsive Buttons: All the buttons except for the login link are fully functional.

Embedded JS9: JS9 was successfully embedded into angular and displays a FITS file with a source marked for users to label.[8]

JavaScript Server: A nodeJS express server that is built into angular apps successfully runs on the Heroku platform. Thus, users can successfully access the angular app via the Heroku address.[9] [10]

Machine Learning Requirements

Datasets: There are 6 high resolution FITS files in the S3 bucket, that combined, have 13,934 labeled sources. We were able to successfully create a large dataset of LSST sample data, 80% of which is used for training and 20% of which is used for validation.

Neural Network: A Convolutional Neural Network was successfully constructed using Tensor Flow. However, the neural network does not reliably classify stars in LSST images.

Map FITS File to Inputs: In order to prepare a FITS file for input**,** a 32x32 pixel swatch can successfully be cut out from a FITS file with a labeled source at the center. This swatch is then flattened out and formatted so that is can be successfully fed through the neural network.

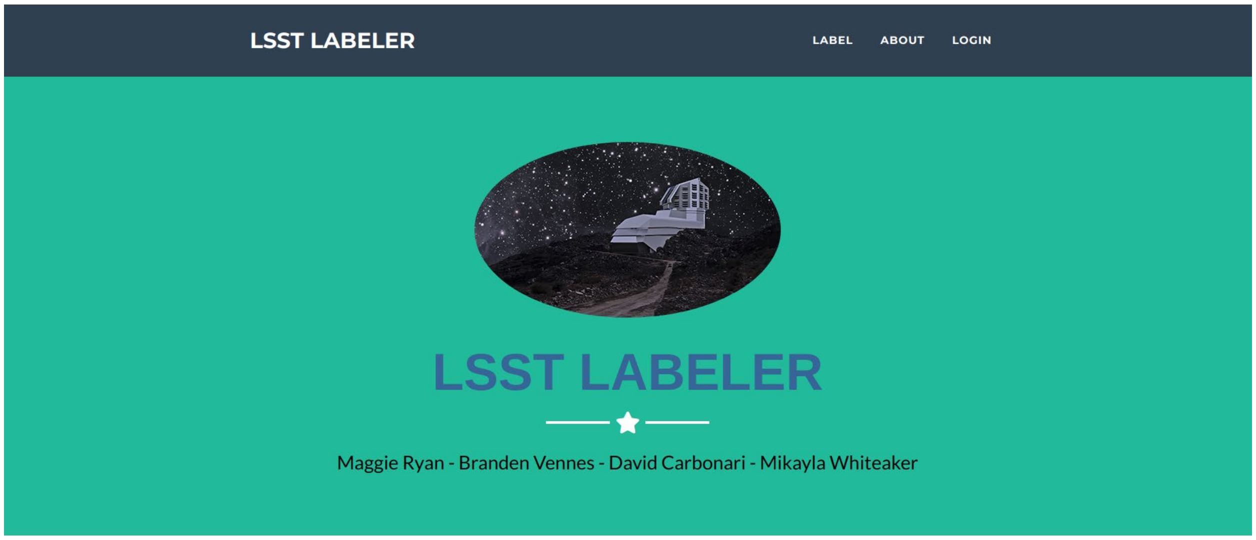
[8] JS9 is a JavaScript library that allows astronomical images to be displayed in the browser

[9] Angular is a platform that makes it easy to build applications with the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

[10] Heroku is a cloud platform that lets companies build, deliver, monitor and scale apps

Finished Product

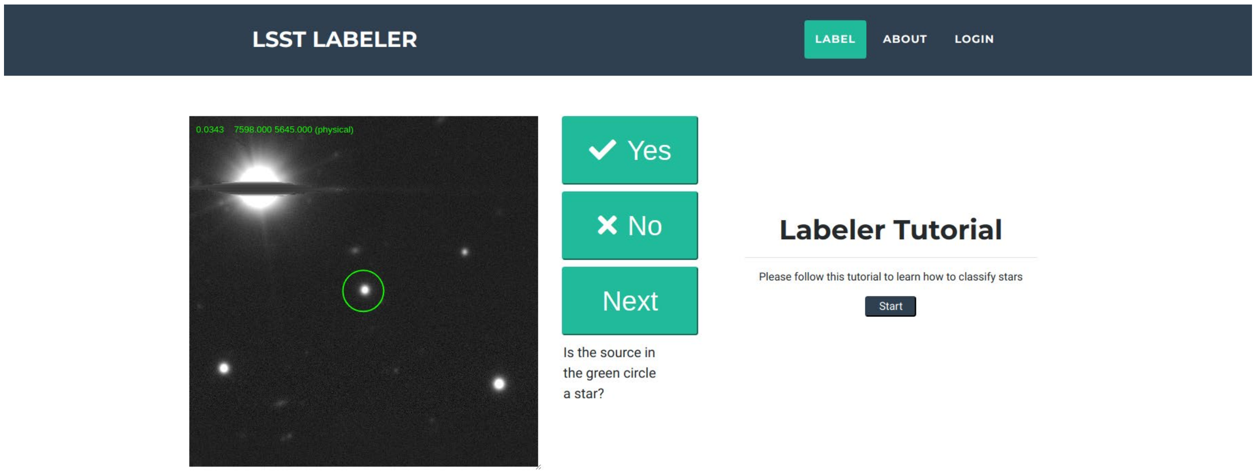
Angular Web App



LSST LABELER

Maggie Ryan - Branden Vennes - David Carbonari - Mikayla Whiteaker

Figure 1: Landing section of the site

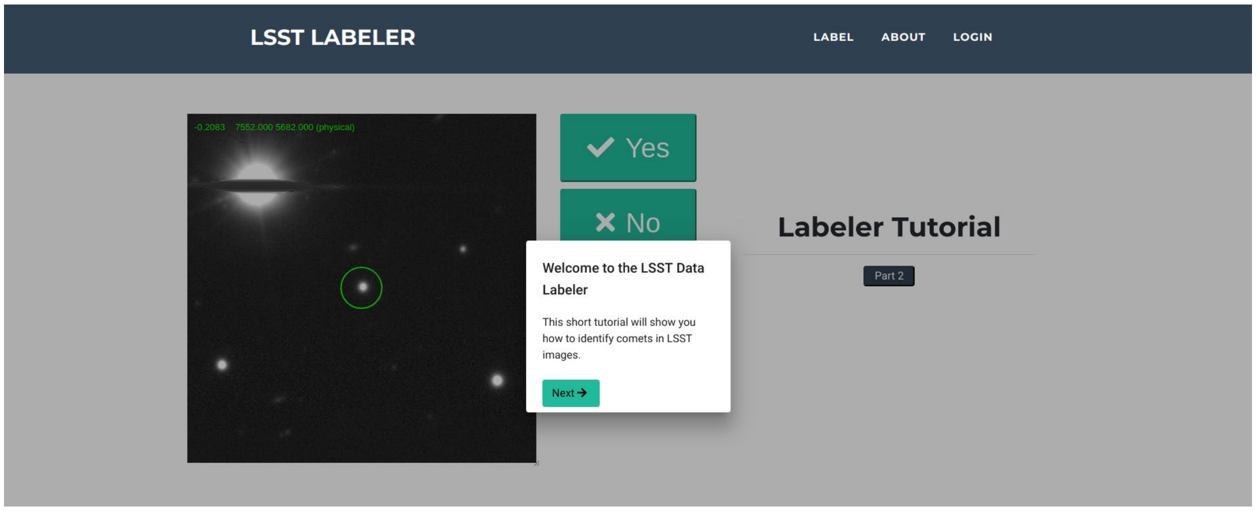


Is the source in the green circle a star?

Labeler Tutorial

Please follow this tutorial to learn how to classify stars

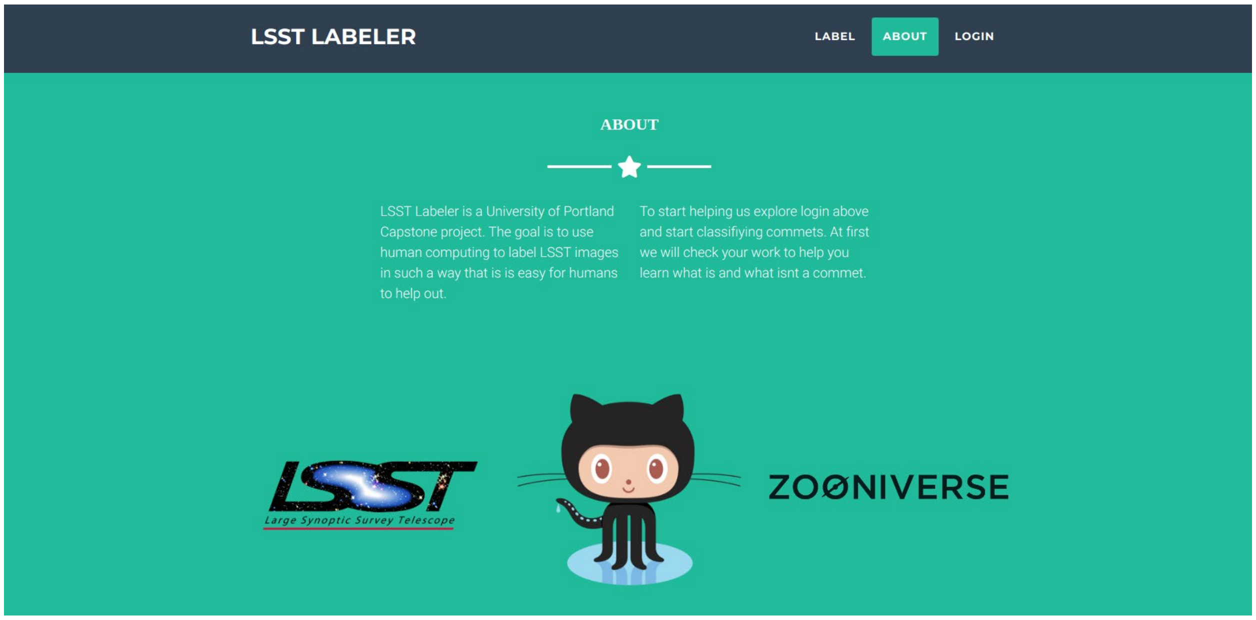
Figure 2: The labeling section of the site with the image is displayed in js9 and a source marked with a green circle



Welcome to the LSST Data Labeler

This short tutorial will show you how to identify comets in LSST images.

Figure 3: Tutorial modal in the labeling section of the site

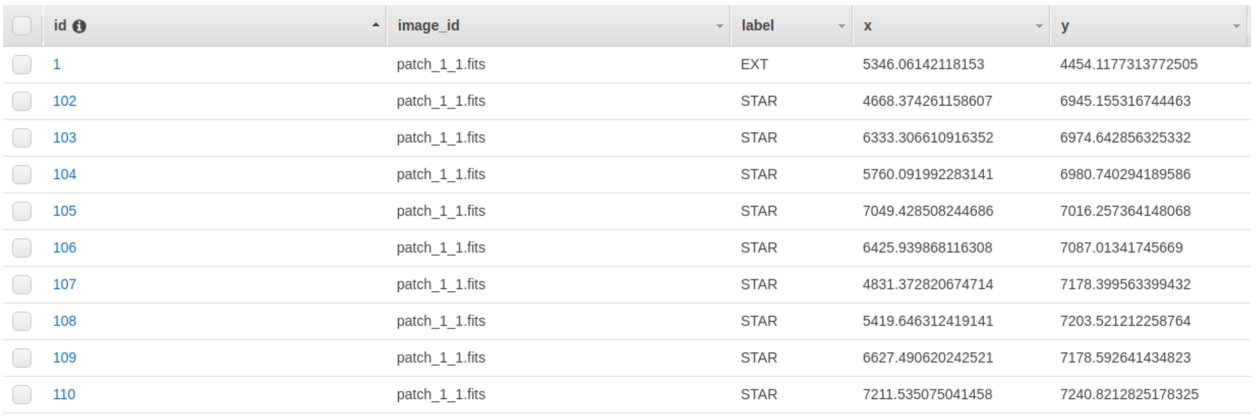


LSST Labeler is a University of Portland Capstone project. The goal is to use human computing to label LSST images in such a way that is is easy for humans to help out.

To start helping us explore login above and start classifiying commets. At first we will check your work to help you learn what is and what isnt a commet.

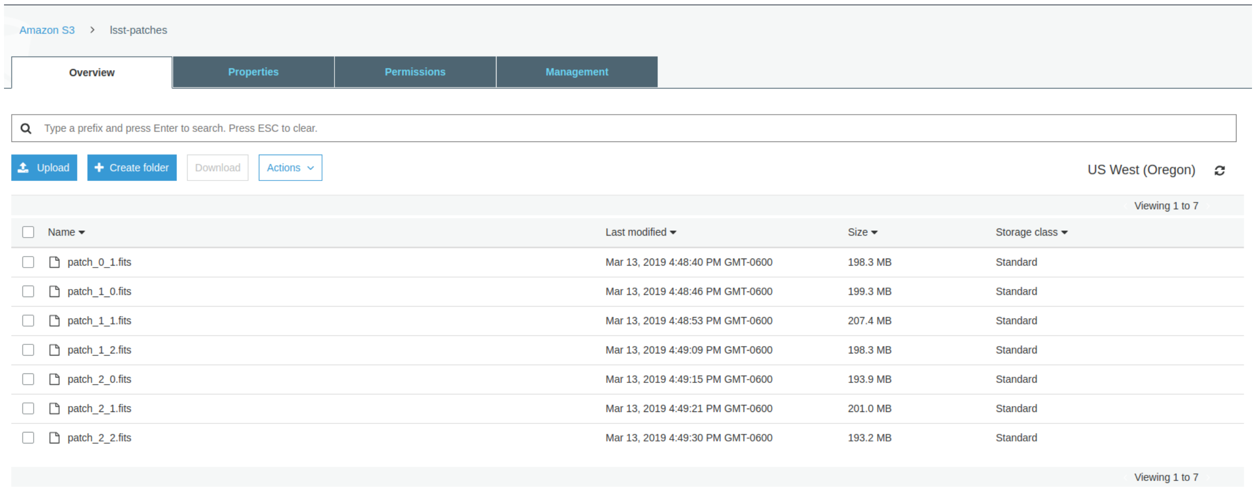
Figure 4: The “About” section of the web site.

AWS Resources



id	image_id	label	x	y
1	patch_1_1.fits	EXT	5346.06142118153	4454.1177313772505
102	patch_1_1.fits	STAR	4668.374261158607	6945.155316744463
103	patch_1_1.fits	STAR	6333.306610916352	6974.642856325332
104	patch_1_1.fits	STAR	5760.091992283141	6980.740294189586
105	patch_1_1.fits	STAR	7049.428508244686	7016.257364148068
106	patch_1_1.fits	STAR	6425.939868116308	7087.01341745669
107	patch_1_1.fits	STAR	4831.372820674714	7178.399563399432
108	patch_1_1.fits	STAR	5419.646312419141	7203.521212258764
109	patch_1_1.fits	STAR	6627.490620242521	7178.592641434823
110	patch_1_1.fits	STAR	7211.535075041458	7240.8212825178325

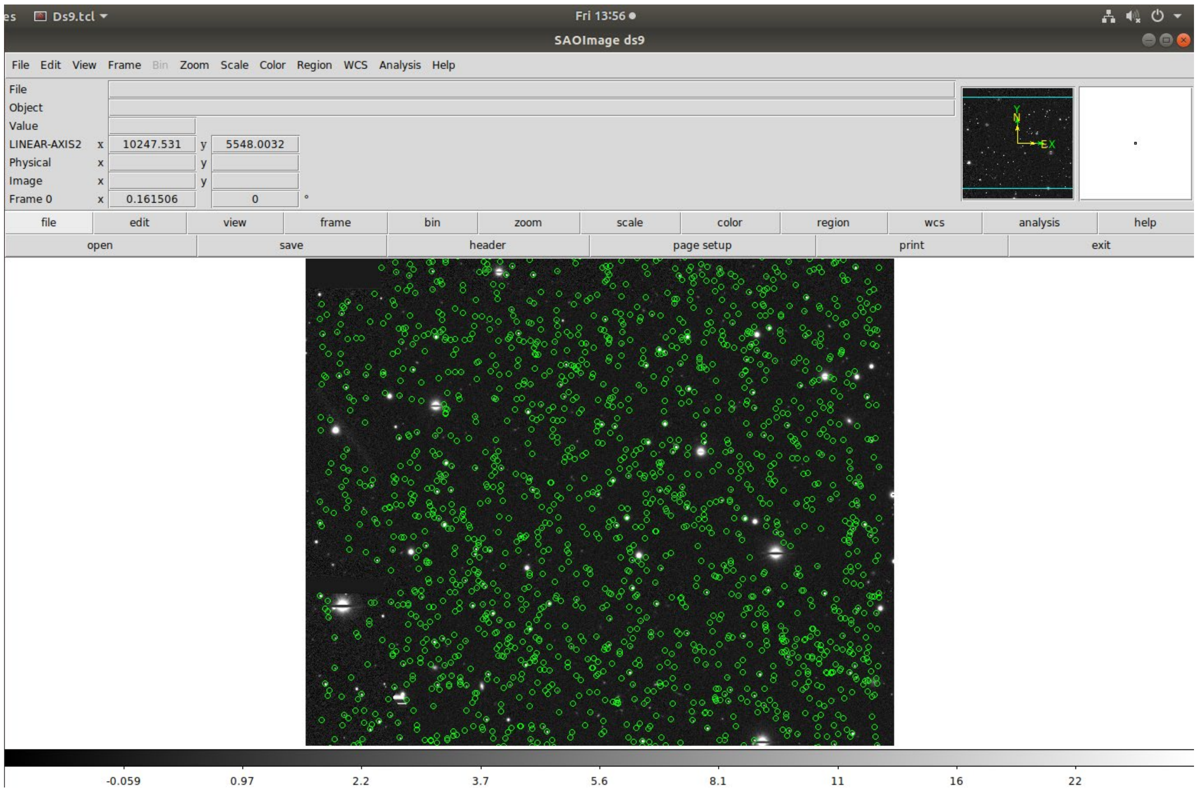
Figure 5: A section of the final source table in DynamoDB



Name	Last modified	Size	Storage class
patch_0_1.fits	Mar 13, 2019 4:48:40 PM GMT-0600	198.3 MB	Standard
patch_1_0.fits	Mar 13, 2019 4:48:46 PM GMT-0600	199.3 MB	Standard
patch_1_1.fits	Mar 13, 2019 4:48:53 PM GMT-0600	207.4 MB	Standard
patch_1_2.fits	Mar 13, 2019 4:49:09 PM GMT-0600	198.3 MB	Standard
patch_2_0.fits	Mar 13, 2019 4:49:15 PM GMT-0600	193.9 MB	Standard
patch_2_1.fits	Mar 13, 2019 4:49:21 PM GMT-0600	201.0 MB	Standard
patch_2_2.fits	Mar 13, 2019 4:49:30 PM GMT-0600	193.2 MB	Standard

Figure 6: The final S3 storage bucket containing LSST FITS files

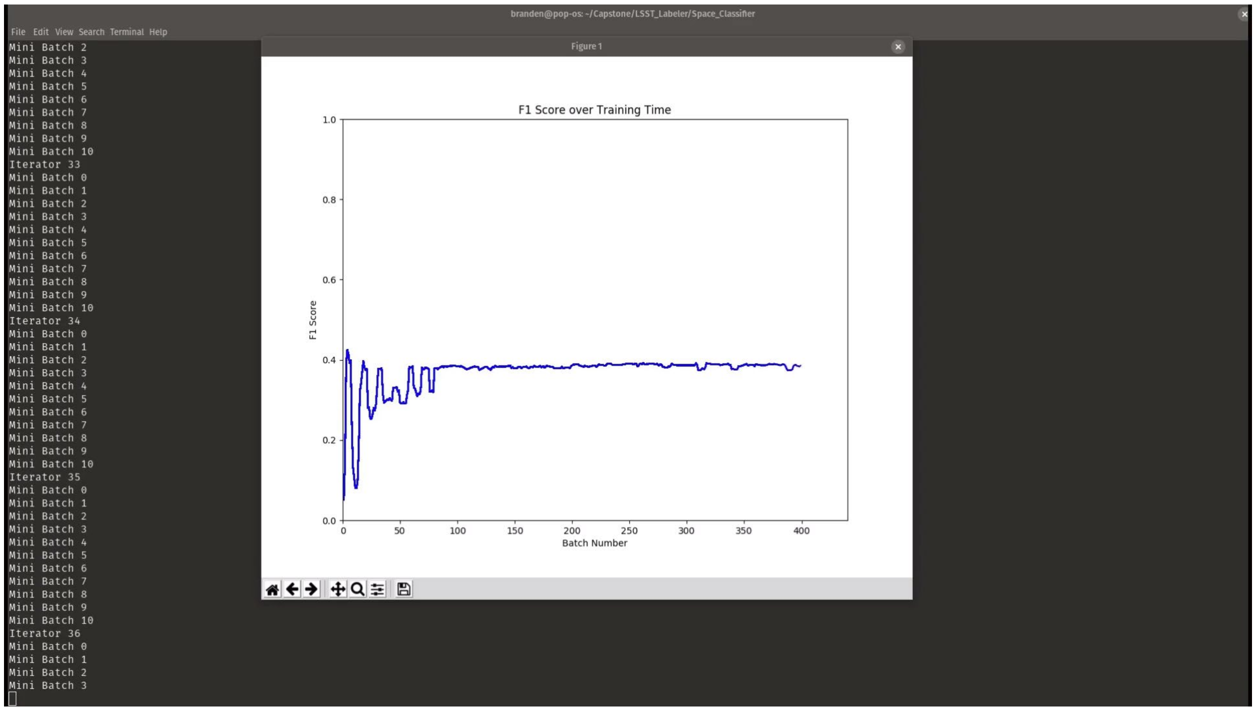
LSST Pipeline


Ds9.tcl
Fri 13:56
SAOImage ds9
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
file edit view frame bin zoom scale color region wcs analysis help
open save header page setup print exit
-0.059 0.97 2.2 3.7 5.6 8.1 11 16 22

```
File
Object
Value
LINEAR-AXIS2  x  10247.531  y  5548.0032
Physical      x             y
Image         x             y
Frame 0       x  0.161506      0        °
```

Figure 7: An example of the massive amount of data in the LSST pipeline. Above is a section of one FITS file with all the stars marked in green circles.

Machine Learning


branden@pop-os: ~/Capstone/LSST_Labeler/Space_Classifier
File Edit View Search Terminal Help
Figure 1
F1 Score over Training Time
F1 Score
Batch Number
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
300
350
400

```
Mini Batch 2
Mini Batch 3
Mini Batch 4
Mini Batch 5
Mini Batch 6
Mini Batch 7
Mini Batch 8
Mini Batch 9
Mini Batch 10
Iterator 33
Mini Batch 0
Mini Batch 1
Mini Batch 2
Mini Batch 3
Mini Batch 4
Mini Batch 5
Mini Batch 6
Mini Batch 7
Mini Batch 8
Mini Batch 9
Mini Batch 10
Iterator 34
Mini Batch 0
Mini Batch 1
Mini Batch 2
Mini Batch 3
Mini Batch 4
Mini Batch 5
Mini Batch 6
Mini Batch 7
Mini Batch 8
Mini Batch 9
Mini Batch 10
Iterator 35
Mini Batch 0
Mini Batch 1
Mini Batch 2
Mini Batch 3
Mini Batch 4
Mini Batch 5
Mini Batch 6
Mini Batch 7
Mini Batch 8
Mini Batch 9
Mini Batch 10
Iterator 36
Mini Batch 0
Mini Batch 1
Mini Batch 2
Mini Batch 3
```

Figure 8: An example of the neural network running and training on LSST data, with an accompanying graph depicting the F1 score, which indicates how well the neural network is classifying stars.

Document Outline

The remainder of this report is organized into three sections. First, the Technical Outcomes section will provide a detailed account of the technical implementation of the project. This section is broken down by the major components of the project, the LSST pipeline, AWS resources, the Neural Network, and the User Interface (i.e. the web app).

The second section details the process outcomes, describing the success and shortcomings of the project. This section goes over the project assumptions and how they changed over the course of the project. It also includes, the primary sprint schedule, changes made to the sprints, team member contributions and roles.

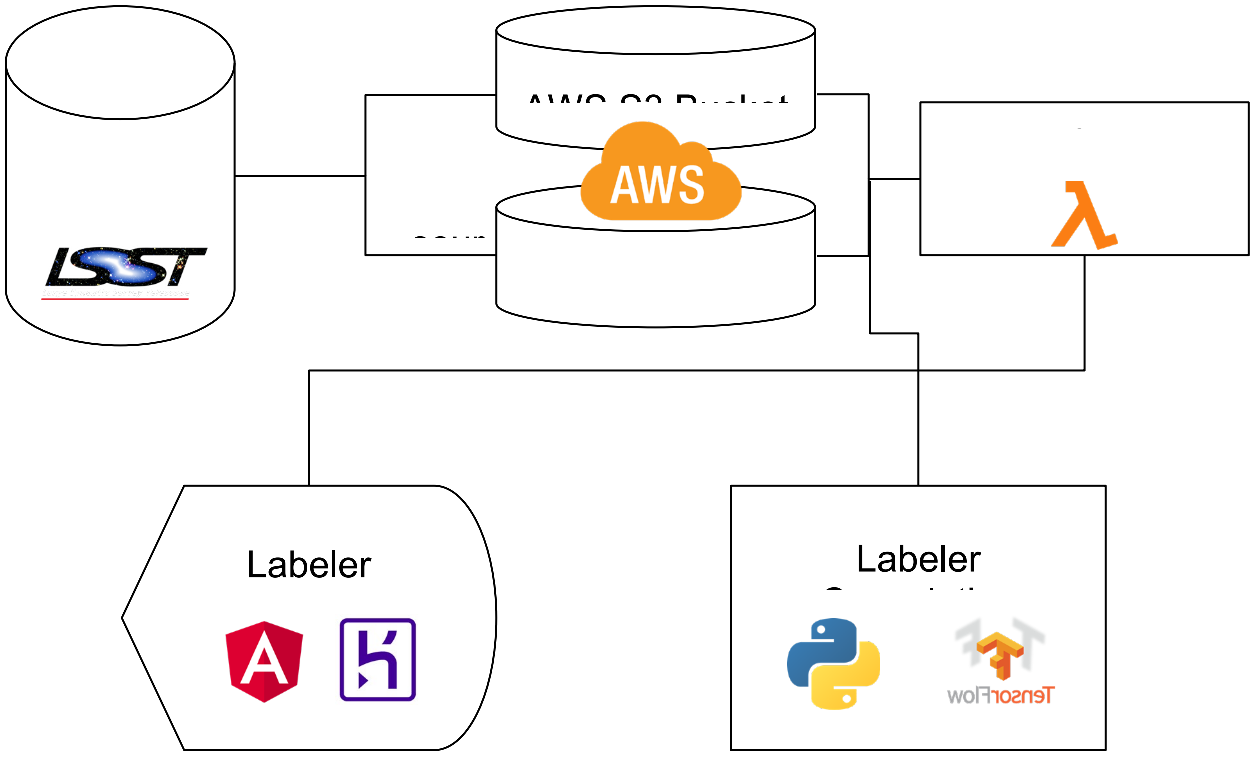
The final section is the conclusion. This will summarize the project and explain how our goals were achieved. An evaluation of the project's' outcomes and the team's process will be given. The conclusion section also documents how the project could be improved if picked up by another team that may be interested in testing the viability of the optimizing the AI.

Technical Outcomes

Overview

The LSST pipeline calibrates LSST data and uploads it to a combination of AWS resources, including S3 buckets and DynamoDB tables. Then an AWS API allows the web app to access this Data. The Neural Network accesses the data in the AWS resources via a Python library.

Product Diagram



Components

LSST Pipeline

Data Pipeline

The LSST Science Pipeline allows users to access and manipulate images of a full sky survey. The pipeline was used to take raw imagery provided by the LSST team and create viewable images. The pipeline was installed on a university lab computer.[11] Once the pipeline environment was set up, the LSST Data Butler, which provides a generic mechanism for persisting and retrieving data using mappers, was used to access and calibrate the sample data.[12]

Data Calibration

Once data was retrieved and a Butler repository was set up, the images were calibrated. At this point the data consisted of many individual calibrated exposures. These exposures were then added to create deeper mosaic images. To do this, several scripts provided by the Pipeline were used. First, a sky map was created which is a tiling of the celestial sphere, and is used as a coordinate system for the final coadded image. These sky maps can either be of the whole sky or a selected region. In addition, these sky maps use the World Coordinate System (WCS), a coordinate system often used in astronomical FITS files. The images are then warped onto the sky map. Following this, all the warped images are coadded, meaning that data from the HSC-I and HSC-R bands are combined.[13] Finally, forced photometry is then applied to these coadded images which is used to accurately detect sources.[14] All of this calibration is done in the pipeline environment.[15]

Data Extraction

First, the coadded images are extracted from the pipeline and uploaded to AWS so that the Convolution Neural Network and the web app can access them. As outlined above, images are wrapped onto a sky map and then coadded to get the deep images that users will be viewing. A sky map is composed of one or more tracts. Those tracts contain smaller regions called patches. Both tracts and patches overlap their neighbors.[16] Then breaking it down even further, the tract is divided into a 3-by-3 grid of patches.[17] The sample data consists of one tract and

[11] See Appendix B
[12] https://confluence.lsstcorp.org/display/LSWUG/Data+Butler
[13] The Hyper Supreme Camera (HSC) has multiple bands given that different parts of the camera are sensitive to different wavelengths of light (see https://www.naoj.org/Projects/HSC/forobservers.html)
[14] For more on LSST photometry see the High Level Technical Specifications section
[15] Done by following this tutorial https://pipelines.lsst.io/v/DM-11034/getting-started/coaddition.html
[16] LSST Science Pipelines. "Getting Started Tutorial Part 4: Coadding Images" Getting Started Tutorial Part 4: Coadding Images - LSST Science Pipelines V16.0 (current) Documentation. Accessed September 27, 2018. https://pipelines.lsst.io/getting-started/coaddition.html.
Copyright 2015-2018 Association of Universities for Research in Astronomy
[17] LSST Science Pipelines. "Getting Started Tutorial Part 4: Coadding Images"

seven useable patches. These seven patches are written to FITS files and have been uploaded to our AWS environment.

Next, using the aforementioned Butler, each source that is identified in the image is uploaded into the AWS database.[18] Each source is labeled as either a star or an extended source (anything that is not a star); this label and the sources location in the image (in WCS coordinates) are uploaded to the database. The top 36% brightest stars and extended sources are pulled from the Butler table. The labels and locations of these sources will be used by the neural network and the web app in order to identify and classify the sources.

LSST Data Specifications

The LSST data is in the FITS file format. This file format is commonly used in astronomy because it "has special (optional) features for scientific data, for example it includes many provisions for describing photometric and spatial calibration information, together with image origin metadata."[19] To display this file format, the SAO Image DS9 application is used by developers to display images locally.[20] In order to display these FITS files in the web app, a similar product called JS9 is embedded into the web page.[21]

Star Photometry

Photometry is the science of the measurement of light. Star photometry is the science of the measurement of light from stars. Stars are bright symmetric light sources. Galaxies, on the other hand, are asymmetrical and elliptical in imagery due to their spiral arms. Comets are distinguishable from other celestial objects such as stars and galaxies – because they have their own distinctive photometry. Galaxies and comets, however, are not uniform like stars. It is this uniform symmetry that the neural network and users of the webapp will be looking for.

[18] These qualities are defined in the following tutorial https://pipelines.lsst.io/getting-started/multiband-analysis.html
[19] https://en.wikipedia.org/wiki/FITS
[20] http://ds9.si.edu/site/Home.html
[21] https://js9.si.edu/

User Interface

The User Interface will be a single scrolling webpage developed in Angular. There will be five main components of the webpage:

Welcome Page

When the user first opens the webpage, there is a simple welcoming message and display. The display provides three main links to other parts of the webpage, *labeling*, *about*, and *login*. Each link is located in the right hand corner of the upper menu bar. If the user does not choose to utilize the buttons to navigate the page, they are able to manually scroll past the welcome page down to the 'Login', 'Labeling' and 'About' sections of the webpage.

Register / Login

This functionality is not fully completed. The login modal is functional; when a user clicks the login link, a modal window is displayed with text inputs that the user can fill out. A modal is a window that appears over the main UI and disables the main UI behind it. Modals focus the user's attention on the new window. At this stage of implementation, no user information is collected, stored, or validated during login.

Tutorial

The tutorial is a series of instructional modal windows, that users can access. After each tutorial section, users can interact with the labeling section to test what they learned. Each modal contains information about how to identify a certain characteristic that might be seen in an image. The modals provide training instructions to the user to be able to identify stars along with whether or not the described characteristic is an identifying feature of a star or if an image contains no stars but other noise. Users can move forward and backwards using the Next and Previous buttons on the bottom of each modal. However, in order to progress through the tutorial and move to the next modal, the previous modal must be completed.

Labeling Section

The labeling section of the website lets users select yes and no classifications for each image that appears on the left side of the screen. A green circle is displayed around the source in the image. When the users classify an image using the Yes and No buttons located to the right of the image, text is displayed on the screen telling them if the classification was correct.

AWS Resources

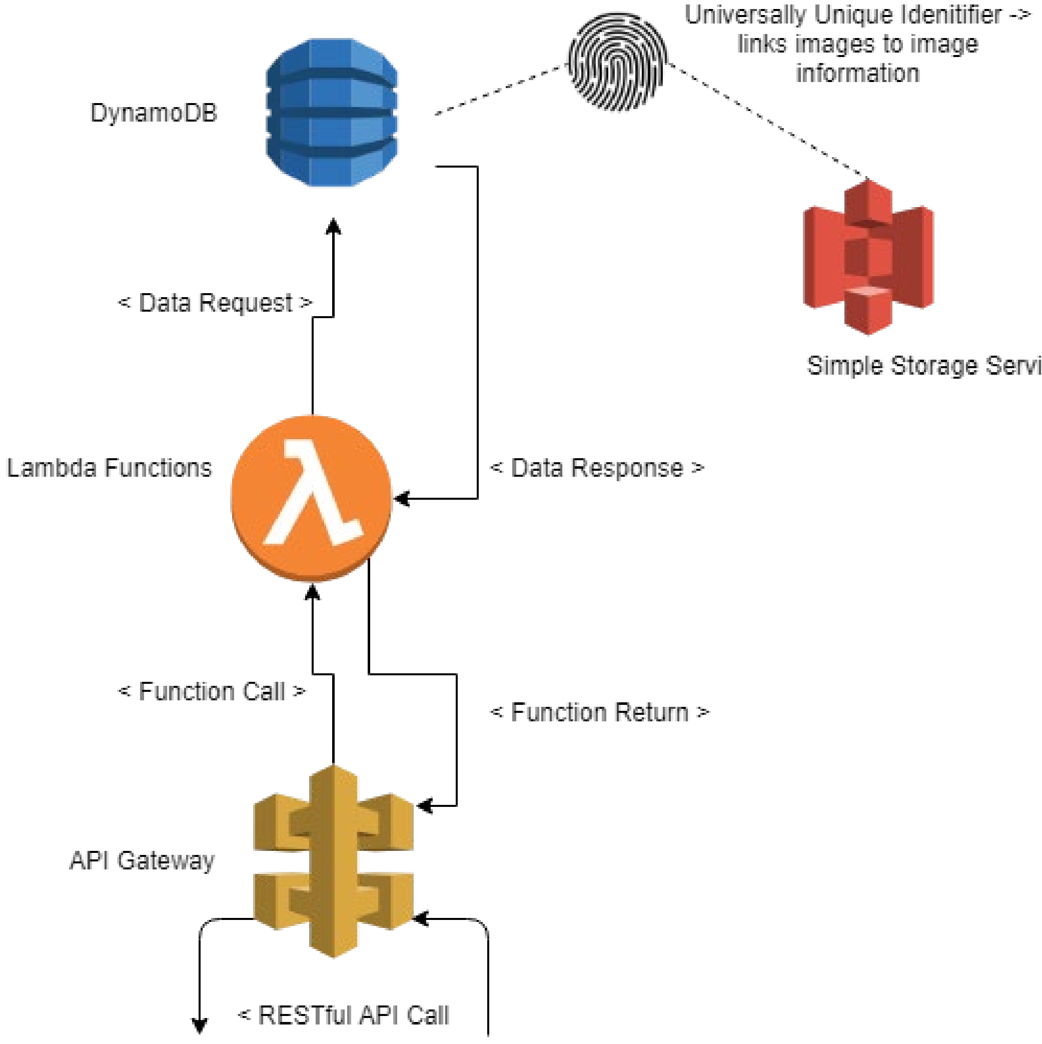


Figure 9: Diagram of the relationships between components in the team's AWS cloud pipeline. The particular functionalities of each component will be described in detail in the following paragraphs.

Simple Storage Service

The Amazon Simple Storage Service (S3) is a cloud storage service that allows the telescope FITS files to be uploaded and downloaded from both our web app and our machine learning AI. The S3 is kept separate from the API. The API acts as a communication layer which receives requests and responds with image locations. Images are uploaded once from the data pipeline and downloaded by the AI and web application. The images are uploaded to the S3 bucket via the Amazon AWS library Boto3. Boto3 contains several functions for managing AWS including S3, DynamoDB, and Lambda Functions. The S3 exists to be a central source to hold all of our FITS image data.

API Gateway

The Amazon API Gateway is a cloud hosted API which receives ‘get’ API requests from the web application and returns source data. It acts as a public proxy to wrap API functions written using AWS Lambda Functions. Get requests are sent to the API Gateway at a public URL. The API has one endpoint in use which returns a random source’s data. When the endpoint is hit the API Gateway triggers the associated Lambda Function, which returns status code and the data requested (Lambda functions are defined below).

Lambda Functions

Lambda Functions are cloud hosted functions that can perform any necessary computations and data retrievals from the DynamoDB database and S3 cloud storage. Our Lambda Functions are written in Python 3.6 and handle requests for sources sent by the Web App. Using the Boto3 Python 3.6 library, the Lambda function can retrieve any entry in the DynamoDB and return it in a formatted JSON object. The result of the Lambda Functions combined with the API Gateway is that images are very quickly served to users who access our web app.

This project only requires one Lambda function named *random_src*. The *random_src* Lambda function is written in Python and uses the Python library Boto3 to access the DynamoDB table. The function responds with a JSON object containing the source’s x and y coordinates and its label.

DynamoDB

Amazon DynamoDB is the cloud database used to store information about the sources within the FITS images. Each source has the following database columns: id, image id, label, x coordinate, and y coordinate. The ‘id’ value contains the source’s unique identification number. This value is the only value that is guaranteed to be unique for every source. The ‘image id’ value is the identification number assigned to the FITS image. The FITS image id is assigned in another table and references the URL location of the FITS image file in our S3. The ‘label’ column contains either ‘STAR’ or “EXT’ (extension/non-star). ‘STAR’ and ‘EXT’ are the two source classifications. The x and y coordinate values are the source’s central x and y values in the World Coordinate System. The FITS image file contains information on converting from WCS to array indices. DynamoDB works nicely as an API database because it is structured in a JSON format, which is the same format as the HTTP API requests.

Machine Learning

The team used machine learning to make automated predictions for newly discovered sources. First, the machine learning model is trained on a large batch of images. Then, a validation set of images are passed to the model for predicting whether a given image is a star or not. The set of predictions are then analyzed to determine the overall accuracy of the model.

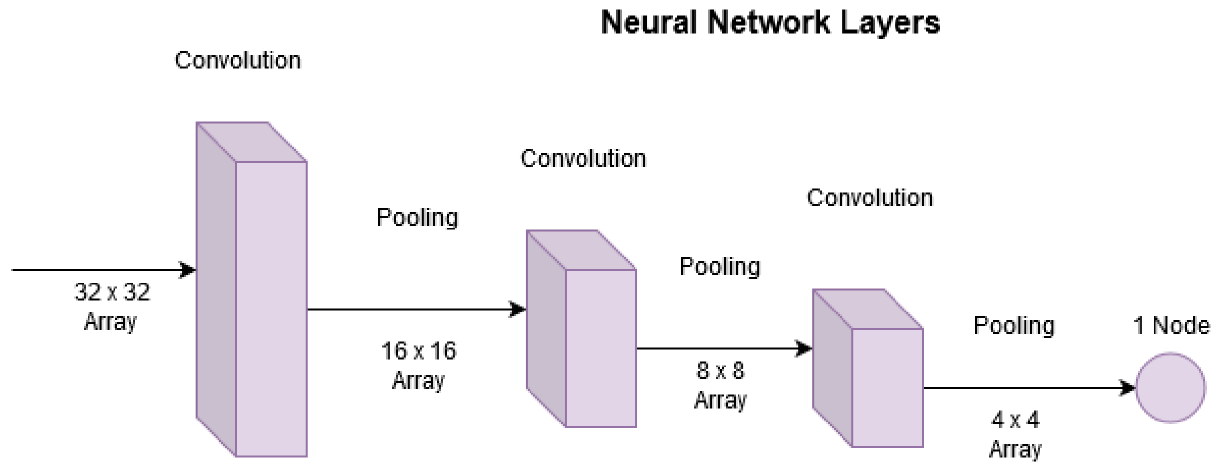


Figure 10: A diagram of the layers within the Convolution Neural Network. The convolution layers filter for shapes. The pooling functions in the hidden layers shrink the size of the vectors as they pass from convolution layer to convolution layer. This decreases the amount of computations that need to be performed at later layers. The final output node makes a prediction.

Training

The machine learning AI is a tool for automating the source classification process. The team chose to implement a binary classifier using a convolutional neural network. The model was built using convolutional neural network tools supplied by the Tensorflow Python library. It is built from three convolutional layers, which filter for lines and shapes, followed by a fully-connected layer containing a single node that gives the star/extension prediction.

To build a new AI predictor using a specific training set, the Python scripts developed by the team load the FITS image files from the AWS S3 storage using the DynamoDB database. The FITS files downloaded from AWS are converted into large arrays (approximately 4200x4200 in size). The source information in the DynamoDB is then used to parse the 4200x4200 arrays into 16x16 arrays containing the specific sources. Next, the set of all 16x16 arrays is passed to the machine learning scripts, where the set of source arrays are split into small batches containing an equal amount of star and extension/non-star sources.

The scripts developed by the team run each of the batches through the machine learning optimization process. With each array that is sent through the ML (machine learning) model, the model optimizer performs backpropagation. This modifies the weights of the final node in the neural network slightly, in order to slowly minimize prediction error. The error is determined by finding the Mean Squared Error between the predicted value (real number between 0 and 1) and the actual label (integer value 1 or 0). Each batch of sources is sent through the neural network dozens of times to increase the overall model's prediction accuracy. Once all the optimizations are completed, the model is saved to local storage.

Predicting

From local storage, the ML model is imported into memory by the Python script and used for making predictions as to whether a source is a star or not. During the prediction process, the model is restored into a Tensorflow session and a set of 16x16 source arrays are sent through it. The value in the final node of the ML model determines whether the source is a star or not. The result from the prediction process does not cause any backpropagation. This means that predictions are made independent of each other.

Disappointingly, our machine learning model is incapable of detecting differences between the stars and the extension source objects. After evaluating the data and the algorithm, the team believes that there are not enough detectable differences between the star and extension source arrays. No amount of training was able to produce reliable predictions. On average, the model predicts images with 66% accuracy. The team has multiple iterations to the model including scaling source array values to between 0 and 1, building mini-batches with an equal number of star and extension sources, and increasing the number of filters within the convolution layer to detect more patterns.

Process Outcomes

This section will give an overview of the successes and shortcoming of the project. The team was successful in conducting research in order to build several products from the ground up. The research resulted in the successful calibration and extraction of LSST data, a neural network that can successful ingest and train on that data, and finally a webapp that successfully displays FITS files which allows users to interact with data. We will examine how our assumptions and initial sprint schedule changed. In addition the team members' contributions will be outline.

Assumptions

Data Formatting

We assumed that the LSST data would be in a FITS file format. This held true, however, as we continued to research and use the FITS files some of our assumptions about them were found to be incorrect. We had assumed that the coordinates for sources where based off of pixel location within the image however the FITS files in the LSST pipeline had a World Coordinate System (WCS) embedded in the FITS file and the source locations are defined in the WCS. The WCS had to be imported in JS9 on the front end, and the coordinates had to be converted to pixel locations for use in the neural network. In addition, we assumed that the pixel values would be like any other grayscale image from 0-255. However, the pixel values in the FITS files from the pipeline range from negative to positive floats in order to achieve a high-resolution image. This had unforeseen impacts on our neural network.

Data Sets

In our original requirements, we set out to classify comets in LSST images. We assumed that we would be able to build training and validation sets of images of comets. However, after extensive research it became clear that there was not a straightforward way to obtain FITS files with comets labeled - easily identifiable comets do not commonly occur in telescope imagery. In addition, even though there are many images of comets in databases from NASA and other observatories, the data format was not consistent, complicating the data set. Therefore, in the interest of achieving our functional requirements we pivoted and decided to classify stars in LSST FITS files only. Since all the stars had already been identified in the LSST data, it was simpler to create a useable data set using only labeled data from the LSST pipeline.

Preliminary Sprint Schedule

This section outlines the implementation of features in each sprint. A copy of the team's sprint plan has been included. Each sprint from the plan is reviewed and comments have been added designating important events.

Sprint 1 Goals:

- Requirements Document
- Kick-Off Presentation
- Unit Test Plan
- Complete initial system setups including processing pipeline, database, storage, artificial intelligence, and web page

During the first sprint, we spent a lot of time defining and researching the scope of the project, and what products we want to build. By the end of the first sprint we began to narrow down what the scope of the project. We lost time researching the direction of the project and learning about LSST images. a lot of time trying to research what the direction of the project would be.

Sprint 2 Goals:

- Send test data throughout entire pipeline from processing to artificial intelligence and web user
- Begin setting up AI system to receive images
- Setup functional database endpoints that takes in an image input and sends image outputs with label attachments
- Create initial active and hosted web page GUI

It was overly ambitious for the team to expect that we could have test data flow through the entire pipeline in Sprint 2. We were not able to have test data flowing to every aspect of the project until second semester. This was because the time to complete all aspects of the pipeline took longer than we had anticipated. The team successfully created a hosted web page on schedule. The team also created the database and AWS API on schedule.

Sprint 3 Goals:

- Refine web GUI with elegant buttons and controls
- Complete initial AI setup
- Engage users in testing and training dataset
- Finalize label list

In Sprint 3, the team made enhancements to the web app graphical user interface (GUI) by adding in responsive buttons and sample images pulled in from the AWS.

The team did not finalize the label list until Sprint 5. This was because we had trouble finding sample images that contained comets.

Sprint 4 Goals:

- Complete full database and storage systems with capabilities to expand if necessary
- Complete database REST API
- Complete default training model for AI using sample images

REST API features were moved to the backlog in this sprint while we focused on the AI and getting usable data from the LSST pipeline into our database. The REST API was completed in sprint 6 when we added the GetRandomImage API call.

The team had a very basic binary classifier set up by Sprint 4. However, it was incredibly rudimentary and was not successful at predicting images. It was beneficial in helping the team build the foundation for the more advanced AI that was developed in Sprint 5.

Sprint 5 Goals:

- Complete processing systems and begin final refinement
- Finalize web app workflows
- AI can sufficiently detect comets and non-comets

In Sprint 5 the team created the framework for the web app's tutorial which included login input-boxes. The machine learning model was developed during this sprint and was able to make predictions on a test dataset (Fashion-MNIST) with 99% accuracy.[22]

Due to the lack of data with labeled comets, we switched our data set to search for and identify stars We were not able to get actual LSST data passed into our machine learning model in time for Sprint 5.

Sprint 6 Goals:

- Optimize all systems to increase Web App interface speed
- Manually test all systems and verify all unit tests

Sprint 7 Goals:

- Final Report
- Final Presentation

Team Contributions

David Carbonari

- Worked with Mikayla to build the framework for angular web application
- Built the tutorial
- Designed the labeling section with the exception of the embedding of JS9
- Contributed to all reports produced by the team

[22] Fashion-MNIST is a dataset containing 70,000 24x24 images of clothing items. It was downloaded with the python statement: 'from keras.datasets import fashion_mnist'.

Maggie Ryan

- Contributed to all documents produced by the team
- Worked with Mikayla to install LSST pipeline and Linux on lab computer
- Researched methods for incorporating AWS database images in angular application
- Worked with Branden to develop a method for converting WCS coordinates to pixel coordinates for neural network
- Worked with Branden to retrieve images from the database, divide images into training and validation sets and developed a convolution neural network machine learning algorithm

Branden Vennes

- Contributed to all documents produced by the team
- Worked with Maggie and Mikayla on creating the AWS pipeline including the S3 storage, Lambda functions, and API gateway
- Worked with Maggie on writing a Python library for connecting to AWS resources from the LSST Pipeline
- Worked with Maggie on developing the neural network

Mikayla Whiteaker

- Contributed to all documents produced by the team
- Worked with David to build the framework for angular web application
- Embedded JS9 in the angular app
- Worked with Maggie to install and set up the lab computer and the LSST pipeline
- Worked with Branden to build out DynamoDB tables, S3 buckets and Lambda functions

Conclusion

In summary, all but one of the functional requirements of this project have been met. The team has built a functional Angular web application which lets users interact with LSST data. The web application is supported by AWS cloud architecture. The source data is calibrated, formatted and extracted from the LSST pipeline and stored in AWS. Finally, the Convolution Neural Network can train itself using numerous batches of images and make predictions on whether a particular object is a star or not. The team worked well together during the development process and was able to meet the original goal of increasing engagement within the LSST community..

The team successfully built a web app through which users can interact with the source data. This app uses AWS cloud architecture to manage information and images passed between the pipeline, web application, and neural network. The neural network is fully integrated into the team's data pipeline and can train and make predictions on the data inputs from LSST.

Given more time to work on the project there are two main goals that we would have, first improve the neural network to more successfully classify sources, and we would expand the web application to a crowdsourcing tool where users can not only sign on and learn about how to classify data but also classify new images that are unknown sources. For the neural network we would explore more into how to make the differences between different types of sources more prominent. This would require more exploration into the other layers of the FITS files and a better way to preprocess the data. The next steps for the web application would be to add a crowdsourcing component to it. That means that users will have to log in and complete the tutorial and labeling with feedback that is already built, and move onto another labeling section where they would be seeing unlabeled sources. This will expand the web application from being an educational community outreach application to an application where people will be able to get involved and directly assist in the labeling of various space objects. This would also require an improvement to the database such that the user inputs on the unlabeled images would be able to be stored.

Appendices

Appendix A

Financial support for LSST comes from the National Science Foundation (NSF) through Cooperative Agreement No. 1258333, the Department of Energy (DOE) Office of Science under Contract No. DE-AC02-76SF00515, and private funding raised by the LSST Corporation. The NSF-funded LSST Project Office for construction was established as an operating center under management of the Association of Universities for Research in Astronomy (AURA). The DOE-funded effort to build the LSST camera is managed by the SLAC National Accelerator Laboratory (SLAC). The National Science Foundation (NSF) is an independent federal agency created by Congress in 1950 to promote the progress of science. NSF supports basic research and people to create knowledge that transforms the future.

Appendix B

To install the LSST Science Pipeline follow the instructions outlined in the following website, https://pipelines.lsst.io/install/newinstall.html. Be sure to check that your system fulfills the prerequisites.

To set up the LSST Science Pipelines follow the instructions outline in the following website, https://pipelines.lsst.io/install/setup.html. This setup is required every time you wish to create a new pipeline environment, i.e. anytime a new shell is opened this setup must be done to initialize the pipeline environment prior to running bash and python scripts that interact with the pipeline.

Data is cloned using git, thus Git LFS must be installed and configured, follow the instructions outlined in the following website, https://pipelines.lsst.io/install/git-lfs.html. Once a sample data repository is cloned into the pipeline environment the package must be setup, added to the EUPS stack, and a Butler repository created. For an example of this process see the following tutorial,

Appendix C

The following is a detailed explanation of the functional requirements, taken directly from the requirements document, which are used in the assessment of the outcomes of the project.

Functional Requirements

LSST Pipeline Requirements

Install Science Pipelines: A developer needs a bash script to install the pipeline so that the pipeline software can be used to access, calibrate, and manipulate data. (2 hours)

- Install prerequisites: The team's lab machine is running Ubuntu 14 so this will require not only the basic prerequisites listed but also the addition ones listed at https://pipelines.lsst.io/install/prereqs/debian.html.
- To install and test the Science Pipelines the bash script must run all steps outlined here, https://pipelines.lsst.io/install/newinstall.html.

Setup Pipeline Environment: A developer needs a bash script to set up the pipeline environment so that they can use the pipeline software, again to access, calibrate, and manipulate data. (1 hour)

- This bash script will be run each time the developer wants to use the installed pipeline. The script must run all steps outlined here https://pipelines.lsst.io/install/setup.html. The script must be run from within the directory in which the pipeline has been installed.

Calibrate Data: A developer needs a bash script that given data in a Git FLS repository, will create a Butler repository, calibrates all exposures, creates a sky map, coadds all exposures onto the sky map, and applies forced photometry to the coadded images. (5 hours)

- Given a link to a data repository the script will execute all the steps outlined, here https://pipelines.lsst.io/getting-started/data-setup.html# to set up the package and create a butler repository.
- The script will then calibrate all the data in the new butler repository by executing the steps outlined here, https://pipelines.lsst.io/getting-started/processccd.html.
- The script will then create the sky map and coadd all the images by executing the sets outlined here, https://pipelines.lsst.io/getting-started/coaddition.html.
- Finally forced photometry will be applied by executing a variation of the steps outline in this tutorial, https://pipelines.lsst.io/getting-started/photometry.html.

Process Data: A developer needs a python script that gets all measured sources in the coadded images. For each source identified put the source and its location in the database. (15 hours)

- This python must be run in the set up pipeline environment and the script will assume the data has been calibrated by the Calibration script
- This script will make use of the Butler to retrieve metadata about sources and their location in the image and LSST libraries to manipulate the image.[23]
- Finally the AWS library (see AWS Requirements) functionality will be used to put the images in the database with the associated data.

Other Image Sources Requirements

Collect Hubble Space Telescope Images: A developer needs a Python script to scrape the Hubble site for images of comets and stars and then put those images in the database. (4 hours)

[23] For more on the LSST libraries see Appendix C

AWS Requirements

S3 Storage Bucket: Any type of user needs blob storage to store all data so that it may be retrieved at any point. (6 hours)

AWS Database: Users need a database to relate all image data to the appropriate label and other metadata. (8 hours)

Web-Facing API: A web programmer needs to be able to retrieve images and their data from the database and storage system. (6 hours)

- Retrieve tutorial data
- Retrieve random image for labeling from the training data
- Modify labels and metadata associated with images that are being labeled

Pipeline Library: A pipeline programmer needs to be able to put images and associated data into the database. (12 hours)

- Put images in the blob under a unique ID and the store the resulting link in the database with associated data.
    - The Amazon S3 used for blob storage returns a link to the image when an image is put in the blob.
    - Associated data that will be represented in the database includes, but is not limited to, a validation flag, source, label, label certainty.

Web Application Requirements

User sign-in: A web user needs to be able to sign in so that the data they produce can be used in the training of the AI. A developer needs users to sign in to prevent untrained uses from contributing to the training of the AI.

- Usernames and passwords will need to be stored and retrieved to enable login functionality.
- Data indicating whether the user has successfully completed the tutorial will also need to be stored.

User Experience: A web user needs an clear, user centered home page to catch their interest and entice me to interact with the labeler. (10 Hours)

Responsive Buttons: A web user needs buttons to be responsive and save my image classifications. (6 Hours)

Embedded JS9: A web developer needs to use JS9 in order to display FITS image files in the Angular web app. (10 Hours)

JavaScript Server: A user needs to be able to access the web application, the JavaScript server will serve the angular application and be hosted on Heroku.

- The Angular application is first compiled into JavaScript by the Angular compiler. This compiled JavaScript code is then served by a NodeJS server. This is a very simple script that creates and Express App (a NodeJS web application framework)[25] that contains the index HTML file, that was created by the angular compiler and links all the JavaScript together. This express app then just simply listens on the specified port. Thus the Angular app is served. This JavaScript will be hosted on Heroku.[26]

Machine Learning Requirements

Datasets: A machine learning programmer needs a dataset of over 1000 images that can be used to train and test a model. (15 Hours)

- This data set must have correct label associated with the image whether from the source or by labeling from the web app, so that the error of the AI can be computed
- This data set must also be divided into a training set and a validation set for supervised learning.
- An image may be rotated or mirrored to increase the size of the data set.

Neural Network: The learning agent will be a neural network. The network will have a FITS file image as input (see next section). The network will have one or more hidden layers. The network will have at least one output indicating whether the input image is a comet or not.

Map FITS File to Inputs: The FITS files the make of the data sets must be mapped to inputs for the network. Each input will represent a pixel's intensity. The intensity will be the normalized grayscale value of the intensity thus each input is between 0 and 1.

Network Output: The network's output must be between 0 and 1, one being the image is a comet, 0 being the image is not a comet.