Summer 8-20-2021

# Application of Machine Learning Techniques to Classify and Identify Galaxy Merger Events in the Candels Field

Alex Koch
*University of Maine*, alex.koch1@maine.edu

# APPLICATION OF MACHINE LEARNING TECHNIQUES TO CLASSIFY AND IDENTIFY GALAXY MERGER EVENTS IN THE CANDELS FIELDS

By

Alex Koch

B.S. Purdue University, 2014

A DISSERTATION

Submitted in Partial Fulfillment of

Requirements for the Degree of

Doctor of Philosophy

(in Physics)

The Graduate School

The University of Maine

August 2021

Advisory Committee:

David Batuski, Professor of Physics, Advisor

Dale Kocevski, Associate Professor of Physics & Astronomy, Colby College, Advisor

Neil Comins, Professor of Physics

Andre Khalil, Professor

Liping Yu, Assistant Professor of Physics

# APPLICATION OF MACHINE LEARNING TECHNIQUES TO CLASSIFY AND IDENTIFY GALAXY MERGER EVENTS IN THE CANDELS FIELDS

By Alex Koch

Dissertation Advisors: Dr. David Batuski & Dr. Dale Kocevski

A Dissertation
Submitted in Partial Fulfillment of
Requirements for the Degree of
Doctor of Philosophy
(in Physics)
August 2021

Galaxy mergers are dynamic systems that offer us a glimpse into the evolution of the cosmos and the galaxies that constitute it. However, with the advent of large astronomical surveys, it is becoming increasingly difficult to rely on humans to classify the vast number of astronomical images collected every year and find the images that capture these systems. In recent years, researchers have increasingly relied on machine learning and computer vision classifiers, and while these techniques have proven useful for classifying broad galaxy morphologies, they have struggled to identify galaxy mergers.

A random forest classifier was applied to a subset of galaxies from the Cosmic Assembly Near-infrared Extragalactic Legacy Survey (CANDELS) to classify merger and non-merger events. 283 merging and 283 non-merging galaxies were selected from the five CANDELS fields, totaling a combined 566 galaxies for training and validation. The classifier was trained on a set of parameters measured for each galaxy, including mass, star formation rate, galactic half-light radius, as well as Concentration and Asymmetry measurements. The classifier performed with a mean accuracy of 92.31% and a precision of 0.9332 on the validation dataset.

Additionally, a computer vision convolutional neural network was trained to analyze and classify images of merger and non-merger events in the same fields. Due to the small number of merger events present in the CANDELS fields, data augmentation was utilized to increase the dataset significantly and boost performance. The computer vision classifier performed with an accuracy of 87.87% and a precision of 0.8683 on validation data. The pre-trained convolutional neural network was then used to predicted classes for a dataset containing active galactic nuclei (AGN) hosting galaxies and a control sample, although no correlation was found between predicted classes and whether the galaxy hosts an AGN.

## DEDICATION

This work is dedicated to my grandfather, Bill Koch, who first taught me to use science to seek answers to my questions.

## ACKNOWLEDGEMENTS

I would like to acknowledge and thank my thesis committee for guiding me through my research during my time at the University of Maine, especially my advisors Dr. David Batuski and Dr. Dale Kocevski for giving me the opportunity to work on this project. I would also like to thank Dr. Salimeh Yasaei-Sekeh, Assistant Professor of Computer Science, and Dr. Terry Yoo, Associate Professor of Computer Science, for graciously allowing me to join their "Introduction to Machine Learning" and "Computer Vision" courses, respectively, and for offering advice on implementing and optimizing the machine learning and computer vision algorithms used in this work. Finally, special thanks to my family for supporting my decision to attend graduate school in the faraway land of Maine, as well as my fiancé for supporting me and encouraging me, especially in the final stretches of the program.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1 INTRODUCTION

## 1.1 Previous Work in Galaxy Classification

In 2011 NASA began its Cosmic Origins (COR) Program, which aims to understand the history of our Universe. One of the COR Program's primary missions is to discover how the astronomical systems we observe in the present epoch evolved through history.[1] The COR Program is a unified effort between Hubble Space Telescope (HST) and the Spitzer Space Telescope (SST). In the future, the James Webb Space Telescope (JWST), the now-in-development infrared successor to HST, will join the effort to discover our cosmic origins.[40]

In order to study our cosmological past, it is necessary for astronomers to peer into the deepest reaches of the Universe. Because light travels at a finite speed, the galaxies we see today are the galaxies of yesteryear. For example, a galaxy that is located one million light years from Earth emits light that takes one million years to reach our planet. As a result, the image that the observer sees through a telescope is of the galaxy as it appeared one million years ago. Astronomers can create a "timeline" of the Universe by observing systems at further and further distances; observing more distant objects is equivalent to looking further back in time. However, it is also necessary to observe as many systems as possible at each distance so that the statistical analysis can take place for each epoch of the Universe's history. By studying the shapes, sizes, and interactions between galaxies during each epoch, we can begin to see a broad picture of how these systems have changed and evolved over time.

However, large extragalactic surveys performed by HST, such as the Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey (CANDELS), have imaged nearly half a million galaxies alone.[30] Other surveys like the Sloan Digital Sky Survey (SDSS) have observed upwards of a million galaxies,[1] and it is becoming increasingly impractical to spend the extreme

number of human-hours it would take astronomers to classify each of these galaxies manually. As a result, astronomers have begun turning to machine learning applications to aid in their classifications.[10]

Machine learning involves designing a computer program to learn how to perform a task without the programmer specifically coding it to do so.[56] By training a program to classify images of distant galaxies, astronomers no longer need to spend as many human-hours sifting through hundreds of thousands – or even millions – of images and classifying each one individually. Instead, one can classify a small subset of images on which to train the program. Once the program has been properly trained and validated on a testing dataset, it can move onto the remaining images for classification.

Some large-scale crowdsources efforts, such as Galaxy Zoo Project,[46] have been utilized to tackle the task of classifying galaxies in massive datasets. However, these efforts rely on citizen scientists to participate in them. Users can create a Galaxy Zoo account, complete a brief training and calibration session, and begin sifting through countless images of galaxies, classifying each one manually.[67] Although this system has worked in the past to extract useful scientific data,[18,31,49] even Galaxy Zoo is beginning to buckle under the enormity of these large surveys. To prepare for impending next-generation surveys from the Euclid Survey Telescope[24] and Large Synoptic Survey Telescope (LSST),[39] Galaxy Zoo is looking to apply machine learning techniques to help mitigate the workload of their citizen volunteers.[1,5,14,45]

Researchers have already had success using machine learning to classify distant galaxies in the CANDELS fields.[32,37] Some of these efforts have used random forest (RF) classifiers,[37] a type of machine learning algorithm that reads input parameters and learns to classify the objects based on those parameters. These efforts have been successful for identifying broad morphology

types.[37] However, RF classifiers have not been successful in identifying galaxy merger events.[37] This is in large part due to the lack of galaxy parameters that indicate merging galaxies. When dealing with image data, it is far better to use computer vision (CV) algorithms, a type of convolutional neural network that learns to identify features present within images. These learned features are then used to classify each image (i.e. "merger" or "non-merger"). Astronomers in the CANDELS collaboration have not yet had a chance to test CV algorithms in identifying galaxy merger systems.

### 1.2 Galaxy Morphology

In order to discuss the significance of galaxy merger events, it is important to understand the types of galaxies that partake in these complex interactions. In 1926, Edwin Hubble devised a scheme for classifying galaxy types based on their shapes and morphologies.[35] Hubble developed this classification system by studying several thousand photographic images of galaxies (known as "extra-galactic nebulae" at the time) taken in 1923. This task took Hubble three years to complete,[35] and astronomers still use his classification system today.[9] In Hubble's sequence, there are four main classes of galaxies: Elliptical (E), Spiral (S), Barred Spiral (SB), and Irregular (Irr) [Figure 1.2.1].[35]

Elliptical Galaxies are defined by their ellipsoidal shape. Hubble's E classification is further divided into E*n* classes, where n is an integer representing the ellipticity of the galaxy and varies from 0 (spherical) to 7 (highly elongated). Elliptical galaxies tend to be more massive than spiral and irregular galaxies and contain more red stars than blue stars.[20] Cool, red stars have much longer lifetimes than their hot, blue counterparts.[54] The fact that ellipticals are dominated by red stars indicates that the blue stars have all died out, leaving only the longer-living red stars. In these galaxies, star formation is thought to have been shut down,[42] otherwise the galaxy would

be able to replenish its supply of blue stars. The exact mechanics of the star formation shutdown process are still a topic of much debate in the astrophysics community.[12,15,16,25,38,59]

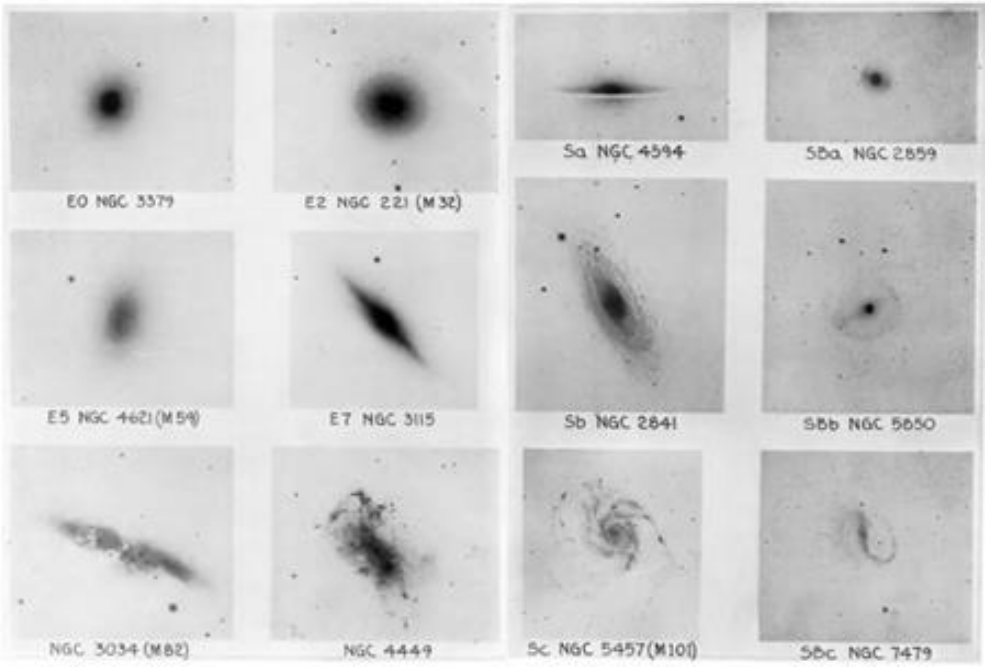


Figure 1.2.1: Hubble Images. Example images of different galaxy morphology classes and subclasses – Elliptical (E0 – E7), Spiral (Sa, Sb, Sc), Barred Spiral (SBa, SBb, SBc), and Irregular (Irr – bottom left two images) – published by Edwin Hubble in 1926.[35]

Spiral galaxies, on the other hand, are named for their dusty disks which orbit around a central bulge in the galactic nucleus. Again, Hubble's S class is divided into subclasses: Sa, Sb, and Sc, where a, b, and c represent how closely the spiral arms are wound around the galactic nucleus. Sa galaxies have arms tightly wound around the nucleus, while Sc galaxies have widely spread spiral arms.[35] Spiral galaxies tend to contain more blue stars, indicating that star formation is still ongoing.

Barred Spirals have a dense, nuclear bulge with a bar running across it. Spiral arms emerge from the ends of the bar. Like the S class, SB is divided into SBa, SBb, and SBc subclasses,

where a, b, and c represent the prominence of the bar as well as the nuclear bulge and tightness of the winding arms. The Milky Way Galaxy is classified as an SB type galaxy.

Finally, Irregular galaxies have no identifiable shape or structure.[35] There are no spiral arms or bars present, and they do not take elliptical forms. Irr galaxies sometimes appear to have been galaxies of another class before unusual gravitational forces acted on then, misshaping them.

### 1.3 Galaxy Mergers

Galaxies do not exist alone in the Universe, however. They tend to group up, forming clusters of galaxies.[6,44] Galaxy clusters also tend to group up themselves to create superclusters.[44] The Milky Way Galaxy is part of a cluster of ~30 galaxies called the Local Group.[6] The Local Group in turn is part of the Virgo Supercluster, which consists o over one million galaxies and is centered on the Virgo Cluster.[63]

Galaxies in the center of these large clusters are more tightly bound to one another resulting in a higher probability of strong gravitational interactions and merger events.[22] As a result, many galaxies in the densest regions begin to have near collisions, tidal interactions, friction, and eventual merging. Galaxy mergers can occur in two basic forms: major and minor mergers. Minor mergers occur when a much larger galaxy “swallows up” a smaller, neighboring galaxy. In these instances, the smaller galaxy is absorbed into the larger one, while the larger galaxy largely remains unchanged and retains its previous structure. The Milky Way Galaxy is believed to have participated in several minor mergers throughout its lifetime, swallowing up nearby dwarf galaxies.[23,48] The second flavor of galaxy mergers takes the form of major mergers. When a major merger occurs between two spiral galaxies, the tidal forces between them rip each galaxy apart, distorting them and destroying their spiral structures. Galaxies consist mostly of gas and

empty space, so they pass through each other, then re-merge. This process repeats until the system eventually relaxes, leaving a single, elliptical galaxy.[22]

It has been observed that regions of higher galaxy density contain more elliptical galaxies in them [Figure 1.3.1].[22] It is believed that one of the primary mechanisms for spiral galaxies to evolve into elliptical galaxies is through major mergers most often found in these dense regions.[22]

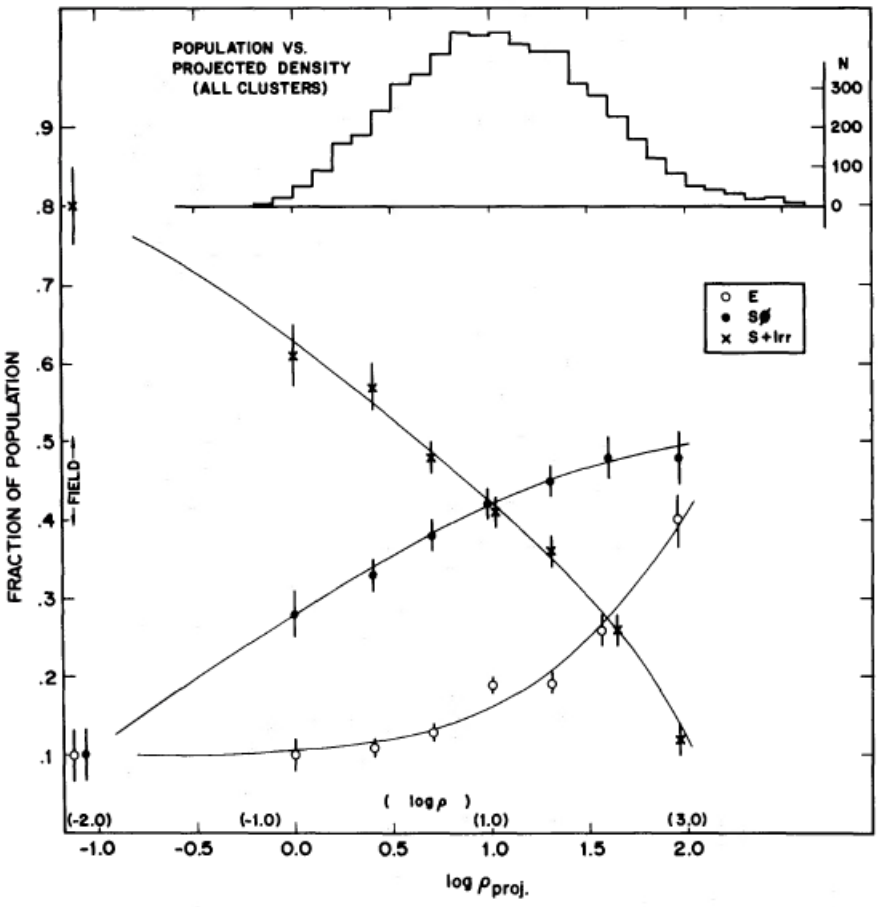


Figure 1.3.1: Galactic Population Density. The fraction of galactic population of each class of galaxy as a function of total number density of galaxies, $\rho_{proj}$, in a cluster.[22]

Galaxy mergers also induce a massive increase in star formation rates for the galaxies participating in them, generating up to 1000 solar masses of new stars per year.[57] This is a result of high-density regions of gas and dust contained in the galaxies merging, collapsing, and generating more stars due to tidal disruptions and shockwaves. These dramatic increases in star formation rates are called starbursts.

### 1.4 Galactic Parameters in Relation to Morphology

Edwin Hubble became the first astronomer to make reasonably accurate distance measurements to galaxies other than the Milky Way using Cepheid brightness and periodicity.[36] He did this by measuring the Doppler-like redshift due to the galaxies' recession speeds – although we now know that this redshift is due to *cosmological* redshift. The distances Hubble calculated were nonetheless correct, and were calculated using

$$D = \frac{v}{H_0} = \frac{cz}{H_0} \; [Eq.\,1.4.1]$$

where D is the distance to the galaxy (measured in megaparsecs), v is the galaxy's recession velocity, $H_0$=69.8±1.9 (km/sec)/Mpc is the Hubble contant,[36] c is the speed of light, and z<<1 is the cosmological redshift. Because c and $H_0$ are both constant, the redshift of a galaxy is equivalent to a measurement of distance, with higher redshift equating to a further distance. Today, there is some debate over the true value of $H_0$.[27,28,43] The measurements used in this study were calculated using a Hubble constant of $H_0$=70 (km/sec)/Mpc.[30]

Several measured parameters can be used as indications of galaxy morphology. One of the most important parameters was first measured by José Luis Sérsic, and it measures the light intensity of a galaxy as a function of distance from the galactic center. He called this measurement the Sérsic Profile:

$$\ln I(R) = \ln I_0 + kR^{1/n} \; [Eq.\,1.4.2]^{58}$$

where I is the intensity, R s the distance from the galactic center, $I_0$ is the intensity at R=0, k is a scaling factor, and n is the Sérsic Index [Figure 1.4.1]. The Sérsic Profile is a generalization of de Vaucouleurs' Law, which describes the light profile of elliptical galaxies.[19] The Sérsic Index, n, is an indication of galaxy morphology. A Sérsic Index of n~4 indicates an elliptical shape, while a Sérsic Index of n~1 generally indicates a spiral shape.

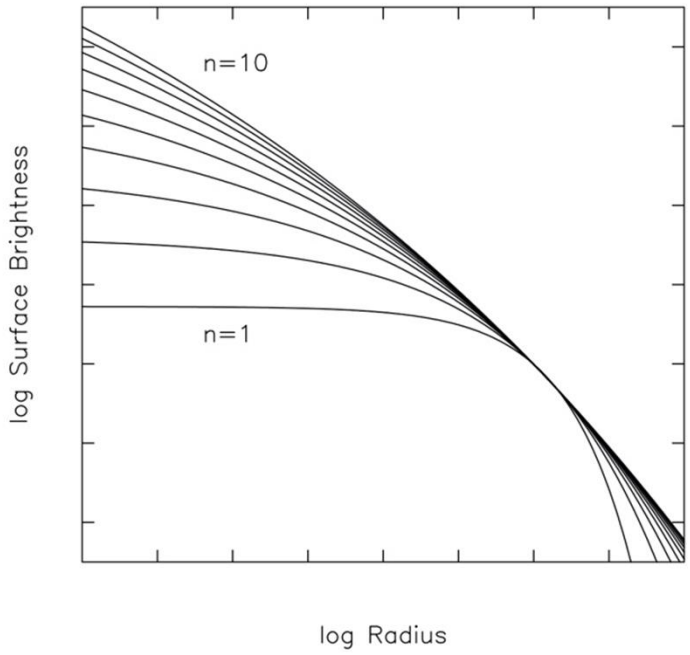


Figure 1.4.1: Sérsic Profile. The Sérsic Parofile for different values of n. Galaxies with a Sérsic Index of n~4 indicate an elliptical shape, while a Sérsic Index of n~1 indicates a spiral shape.[65]

While the Sérsic Profile and de Vaucouleurs' Law can be expressed in terms of galactic radius R, it is often difficult to determine the true radii of galaxies since they do not have clearly defined edges. At further distances from the galactic nucleus, a galaxy's image becomes fainter and more ambiguous, making it difficult to define a true galactic radius. Instead, astronomers

often use a galaxy's half-light radius to express a galaxy's size. The "half-light" or "effective" radius is the radius which contains half of a galaxy's total luminosity.

Color Index (CI), a quantitative measurement of a galaxy's color profile, can be used as another indication of galaxy morphology.[60] As stated earlier, elliptical galaxies tend to have a red color, due to the shutdown of star formation, while spiral galaxies are bluer, since their star formation processes have not quiesced. CI is calculated by taking the difference between magnitudes of a galaxy through two different filters, i.e.

$$CI = B - V \ [Eq. 1.4.3]$$

Where, in this example, B and V are the magnitude through the Johnson B and V filters, respectively. CI follows a bimodal distribution for galaxy morphology [Figure 1.4.2].[60]

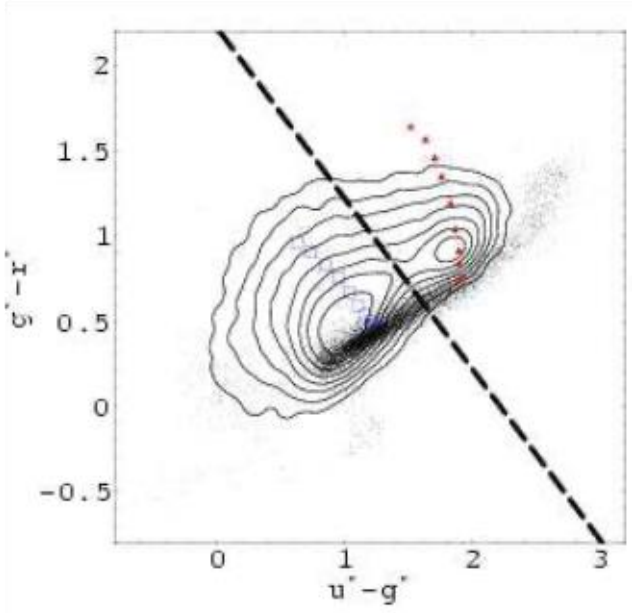


Figure 1.4.2: Color Index. The color-color profile of galaxies in the SDSS field. Blue squares represent spiral galaxies, red triangles indicate elliptical galaxies, and dots represent stars. The dashed line represents the u*-r* = 2.22 separator, where u*, g*, and r* are the magnitudes measured through ~3500 Å, ~4800 Å, and ~6200 Å filters, respectively. Galaxies below the separator are spiral while galaxies above the separator are elliptical. The contours represent Gaussian standard deviation in steps of σ/4.

While parameters like Sérsic Index and Color Index have been important for identifying broad morphology types for galaxies,[20,60] it has been difficult to use them for identifying galaxy mergers.

A galaxy's Asymmetry value (A), as described by Abraham et. al (1996)[1], is generated by first separating an image of the galaxy from the background sky. Then, the galaxy image is rotated and subtracted from itself. A highly disturbed galaxy typically has a larger Asymmetry value. The opposite measurement is a galaxy's Concentration (C). C is the intensity-weighted second order moment of the image:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y) \ [Eq. 1.4.5].$$

Finally, we have two non-parametric measurements of a galaxy's structure: the Gini (G) and M20 (or $M_{20}$) coefficients. G is historically used in economics as a measurement of the distribution of a population's wealth. However, astronomers have adapted G to describe the distribution of *light* that we receive from a galaxy. For a discrete population (galaxy), G is defined as

$$\frac{1}{2\bar{X}n(n-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} \left| X_i - X_j \right| \ [Eq. 1.4.6]$$

where n is the population (number of pixels) and X is the wealth per individual (pixel flux).[47]

The second order total moment $M_{tot}$ is the sum of a galaxy's pixel fluxes multiplied by the squared distance from the galactic center:

$$M_{tot} = \sum_i^n f_i[(x_i - x_c)^2 + (y_i - y_c)^2] \ [Eq. 1.4.7]$$

where $f_i$ is the pixel flux for each pixel contained in the galaxy's segmentation map, and $(x_c,y_c)$ is the center pixel of the galaxy.[47]

$M_{20}$ is the normalized second order moment of the brightest 20% of a galaxy's flux. To calculate $M_{20}$, we rank-order the pixels by flux and sum the second order moments of each next-brightest pixel until the sum of fluxes equals 20% of the total flux. We then normalize to $M_{tot}$ and take the log:

$$M_{20} = \log_{10}\left(\frac{\sum_i M_i}{M_{tot}}\right), while \sum_i f_i < 0.2 f_{tot} \; [Eq.\, 1.4.8].^{47}$$

$M_{20}$ has the benefit of relying on the *square* of the pixel distance to the galactic center $(x_c,y_c)$, which is a now a free parameter. This differentiates it from C and allows for more sensitivity to multiple nuclei, which are commonly present in merger events.[47]

**1.5 CANDELS**

The Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey (CANDELS) is a combination wide- and deep-sky survey initiated in 2010. The survey was designed to observe the first third of galactic evolution by studying over 250,000 galaxies with $1.5 < z < 8$ up to a limiting HST-measured magnitude of $H_{mag} = 27.7$. The survey studied five fields and is divided into two parts. The deep survey observed the Great Observatories Origins Deep Survey North/South (GOODS-N, GOODS-S), which covers a ~12.5 arcmin$^2$ field. The wide survey studied GOODS-N and GOODS-S as well as the Cosmic Evolution Survey (COSMOS), the Extended Groth Strip (EGS), and the Ultra Deep Survey (UDS). The survey totals roughly 800 arcmin$^2$. The fields are summarized in [Table 1.5.1].[30]

| Field | Coordinates | Tier | WFC3/IR Tiling | HST Orbits/Tile | IR Filters[a] | UV/Optical Filters[b] |
|---|---|---|---|---|---|---|
| GOODS-N | 189.228621, +62.238572 | Deep | ~3 × 5 | ~13 | *YJH* | *UV,UI*(*WVz*) |
| GOODS-N | 189.228621, +62.238572 | Wide | 2 @ ~2 × 4 | ~3 | *YJH* | *Iz*(*W*) |
| GOODS-S | 53.122751, −27.805089 | Deep | ~3 × 5 | ~13 | *YJH* | *I*(*WVz*) |
| GOODS-S | 53.122751, −27.805089 | Wide | ~2 × 4 | ~3 | *YJH* | *Iz*(*W*) |
| COSMOS | 150.116321, +2.2009731 | Wide | 4 × 11 | ~2 | *JH* | *VI*(*W*) |
| EGS | 214.825000, +52.825000 | Wide | 3 × 15 | ~2 | *JH* | *VI*(*W*) |
| UDS | 34.406250, −5.2000000 | Wide | 4 × 11 | ~2 | *JH* | *VI*(*W*) |

Table 1.5.1: CANDELS Fields. Summary of the five CANDELS fields, including the wide and deep surveys in the two GOODS fields, where wide fields cover a larger field, and deep fields are imaged with more integration.[30]

### 1.5.1 CANDELS Data

The CANDELS images were taken using the Hubble Space Telescope (HST) Wide Field Camera 3 (WFC3) instrument. The images were taken across a broad range of wavelengths using WFC3's two channels: ultraviolet/optical (UVIS) and near-infrared (IR). The UVIS channel is sensitive to 200-1000 nm wavelengths of light and has a field of view of 162x162 arcsec, while the IR channel can image ~800-1700 nm wavelengths and has a view of 136x123 arcsec. Each field in CANDELS is a mosaic taken over 902 orbits of HST, equating to roughly two months of observing time [Figure 1.5.1.1].[30]

Using all 4 HST observation bands (F606W, F850LP, F125W, and F160W), CANDELS researchers created cutout images ("postage stamps") of all galaxies observed in the GOODS-S field. After randomly selecting 100 galaxies and having five people classify them, it was discovered that fainter galaxies with $H_{mag} > 24.5$ were difficult to classify. As a result, a limit of $H_{mag} < 24.5$ was implemented for classification. No other cutoffs based on redshift, stellar mass, etc. were made. In the end, cutouts of 7634 galaxies were generated, as well as segmentation maps for them using Source Extractor (SExtractor), a~~n~~ program designed to identify objects present in an astronomical image.

Additionally, astronomers have already measured several galaxy properties from the CANDELS images including magnitude, mass, galactic half-light radius, Sersic index, and color magnitudes. Several morphological parameters have been measured as well, including Gini, $M_{20}$, Asymmetry, and Concentration.

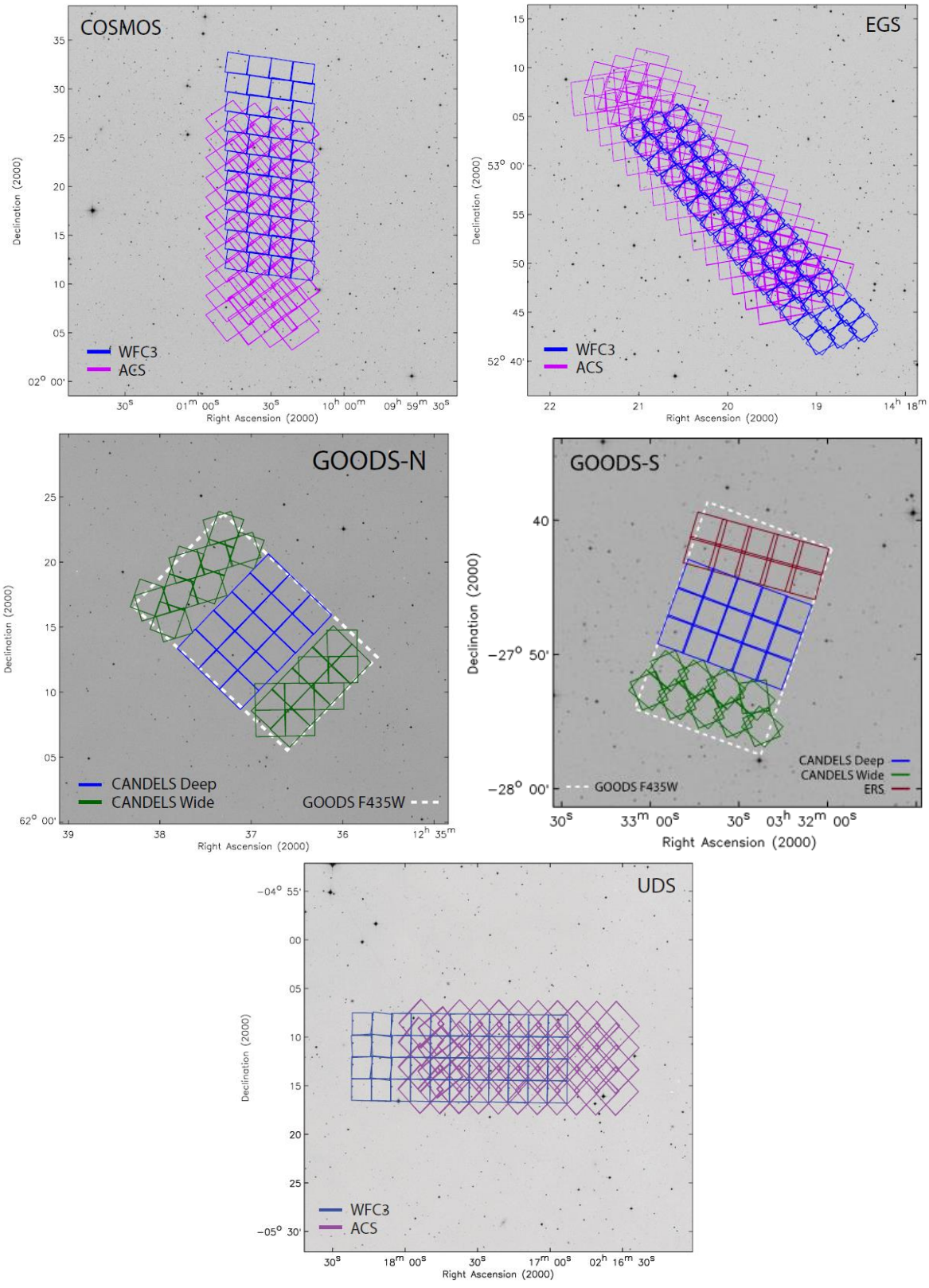


Figure 1.5.1.1: CANDELS Fields. HST mosaic grids of the five CANDELS fields.[30]

### 1.5.2 CANDELS Galaxy Morphologies

Galaxies in GOODS-S with $H_{mag} < 24.5$ have been classified by CANDELS researchers using a proprietary classification scheme.[41] There are five morphological classes used: Disk, Spheroid, Irregular/Peculiar, Compact/Unresolved, and Unclassifiable. Disk, Spheroid, and Irregular/Peculiar classes follow the Hubble Sequence described in Section 1.2. The Compact/Unresolved class is used for galaxies that are either point sources (i.e., stars) or are so small that no structure can be determined. Galaxies are Unclassifiable if they cannot be placed in any of the other classes. This may occur if there was an error in the image, if it is too small to determine structure, or if SExtractor incorrectly identified a star, star cluster, nebula, etc. as a galaxy.

There are three interaction classes used to describe each galaxy: Merger, Interaction within SExtractor segmentation map, and Non-interacting companion. The "Merger" flag is used if the image contains a single galaxy that appears to have undergone a merger event. This could be evidenced from tidal features such as tidal arms or loops. "Interaction within SExtractor segmentation map" is used to describe an image that contains *two* galaxies that show interaction features. Finally, "non-interacting companion" describes an image of two galaxies that appear to be close together in the sky do not appear to be interacting with each other. All of the morphological and interaction classes are not mutually exclusive, and classifiers could use multiple classes to describe the same image, with exception to the Unclassifiable flag, which was only used if none of the other classes applied.

Twelve structure classes were presented for human classifiers to select. Two of these are the Tidal Arms and Asymmetric classes. Tidal arms are stretched or elongated arms of a galaxy, which are formed when a nearby galaxy exerts gravitational "tidal" forces. Similarly, a galaxy

can become asymmetric when a nearby galaxy disturbs it through gravitational interactions. These two structural classes are good indications that a galaxy is undergoing or has undergone a merger event in the past [Figure *1.5.2.1*].

Figure 1.5.2.1: Galactic Structural Classes. Example images of galactic structural classes. The Tadpole Galaxy (left) exhibits a prominent tidal arm[52], while Messier 66 (right) is an asymmetric galaxy.[8]

Researchers in the CANDELS group created a Graphical User Interface (GUI) for classifying galaxy postage stamps using Perl/Tk and SAOImageDS9 [Figure *1.5.2.2*]. Users were each given a "chunk" of 200 images to classify. Once the chunk was completed, the user was assigned the next chunk. The GUI presents the user with postage stamps in each of the following HST bands: F606W, F850LP, F125W, and F160W; as well as the contour map of each galaxy. The user is then asked to fill out checkboxes for each of the morphological and interaction classes that apply to the image. A minimum of three users classified galaxies in the GOODS-S wide field. A minimum of five users classified galaxies in the GOODS-S deep field at each of the three image integration depths (2-, 4-, and 10-epoch depth), for a total of fifteen classifications for each galaxy. For calibration, each user was assigned a set of 25 galaxies to classify, which demonstrate the range of possible classes. Additionally, the first chunk assigned to each user was identical. In total, 65 classifiers worked on the GOODS-S images.[41]

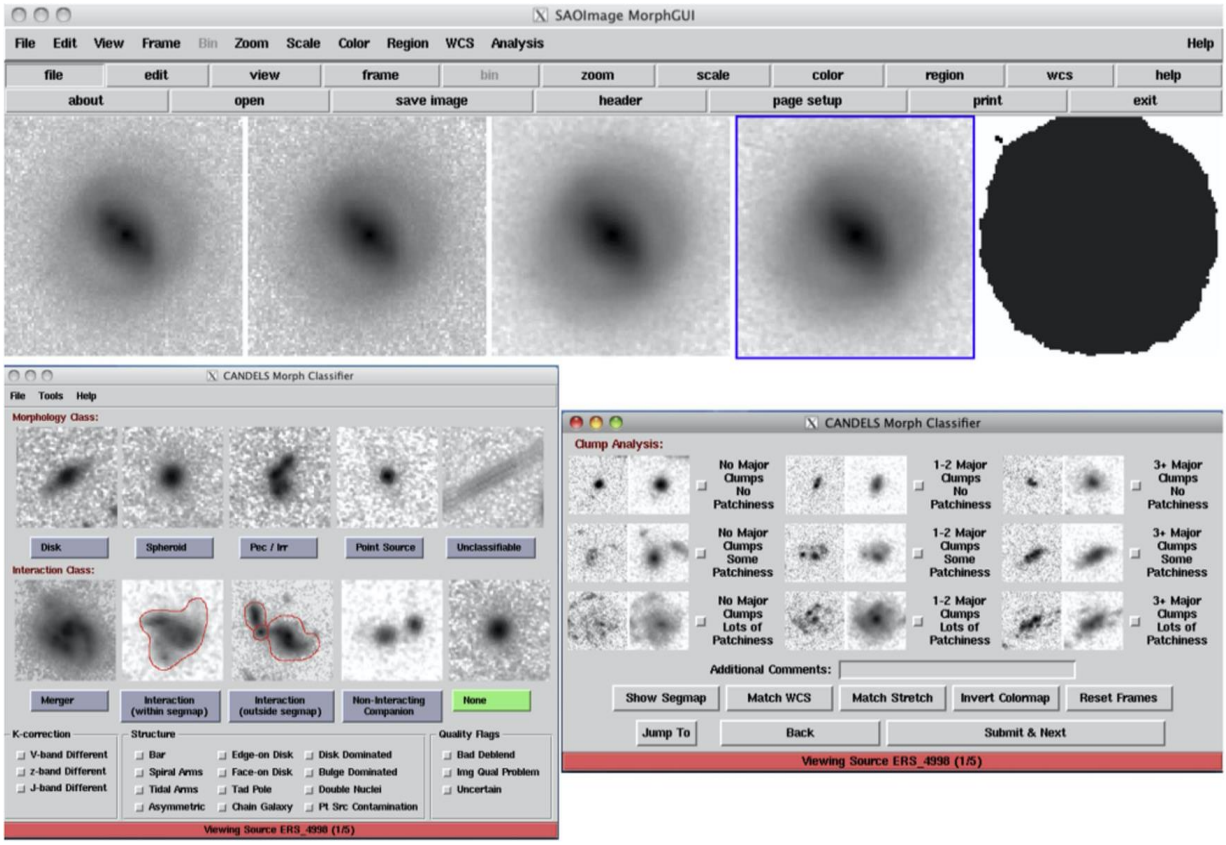


Figure 1.5.2.2: CANDELS Classification GUI. The GUI designed by CANDELS researchers to classify galaxies in the CANDELS wide and deep fields.

# CHAPTER 2 MACHINE LEARNING

"[Machine learning is the] field of study that gives computers the ability to learn without being explicitly programmed." – Arthur Samuel (1959)[56]

One of the first wide-spread applications of machine learning was introduced to the public in the 1990's as the spam email filter. Users who received spam emails would submit unwanted emails to a spam folder. A computer program would look through the folder and identify common traits that were found among the emails (i.e., words like "Congratulations!" or "Act now!", as well as the email addresses from which they were sent). As more and more spam emails were analyzed by the program, it could effectively identify certain emails as spam, automatically dump them into a spam folder for the user and could even generate new rules for identifying unwanted emails. Since the 1990's spam filters have become so robust that users rarely need to manually flag emails as spam anymore.[29]

Machine learning techniques have advanced well beyond the capabilities of early spam filters. Today, machine learning techniques are being applied in countless ways. Pharmaceutical transport companies are using them to recommend shipping containers for drug companies. Facebook uses a neural network called Deepface for facial recognition when tagging users in uploaded images.[11] Voice-command apps such as Apple Siri, Microsoft Cortana, and Amazon Alexa were all trained to comprehend human speech through machine learning. Among all these applications, there are several different "types" of machine learning algorithms. This project focuses on two machine learning techniques for galaxy classification: Random Forest and Computer Vision classifiers.

## 2.1 Supervised and Unsupervised Learning

Machine learning classifiers come in two forms: Unsupervised and Supervised. An Unsupervised learning algorithm is provided raw data with no labels from the user. The program's job is learning how to group or cluster data points together into classes. Unsupervised learning is useful if the user knows there are different classes but is unsure what the classes are. An Unsupervised learning algorithm can generate classes for the user.

When using a Supervised learning method, the user provides all the data, as well as labels for each data point. For example, a user might pass a spam filter several emails. Each email also contains a label, labelling it either "spam" or "not spam." As the program analyzes each email, it also looks at the label for each one. It learns traits that spam-labelled emails tend to have, versus traits that not-spam emails tend to have when creating classification rules. Supervised learning is an effective tool for classification tasks since the labels are already established. Because the CANDELS galaxies have already been classified by humans, this project uses a Supervised learning method.

## 2.2 Decision Trees and Random Forest Classifiers

Random forest (RF) classifiers are based on the idea of a decision tree [Figure 2.2.1]. A decision tree attempts to classify data based on different parameters ("features") that each data point has. The tree uses features to split the data from the "root node" into separate groups called "internal nodes." Each internal node is then branched into deeper "internal nodes" until all data has been successfully classified into distinct groups called "leaves."

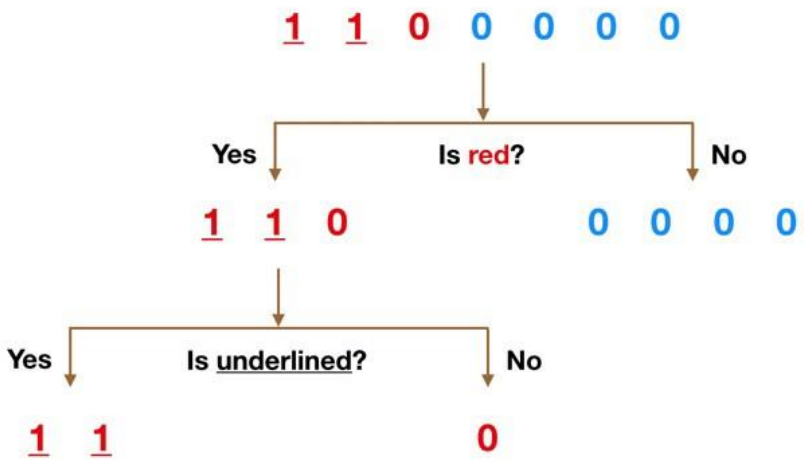


Figure 2.2.1: Decision Tree: The numbers (data) are first divided into two nodes based on color. Along the left branch, they are further classified based on whether they are underlined or not.

Consider the training data $S = \{(x_1^d, y_1), \ldots, (x_n^d, y_n)\}$, where $x_i^d$ is a feature vector of d dimensions, and $y_i$ is a label. The classifier's goal is to split the dataset into nodes which each contain datapoints of a single label k:

$$S_k = \{(x, y) \in S \mid y = k\}$$

A binary classifier splits the dataset into subsets $S_k$ by selecting a feature $x_i^j$ and threshold t such that any datapoints above the threshold belong to one class, and datapoints below the threshold belong to the other:

$$S_1 = \{(x, y) \in S \mid x_i^j < t\}$$

$$S_2 = \{(x, y) \in S \mid x_i^j > t\}$$

One way to measure the label distribution in a node is to calculate the node's Impurity. There are several different impurity measurements, including Gini Impurity, Entropy, and

Misclassification. The Scikit-Learn packages used in this study use the Gini Impurity G(S) by default:

$$G(S) = \sum_{k=1}^{K} p_k(1 - p_k) \text{ [Eq. 2.2.1 ]}$$

where $p_k$ is the probability of randomly selecting a datapoint in the set which has label k, and K is the total number of distinct labels. For a binary classifier, which contains datapoints of only two labels, G(S) can be reduced to

$$G(S) = 2p_1(1 - p_1) \text{ [Eq. ][Figure 2.2.2]}$$

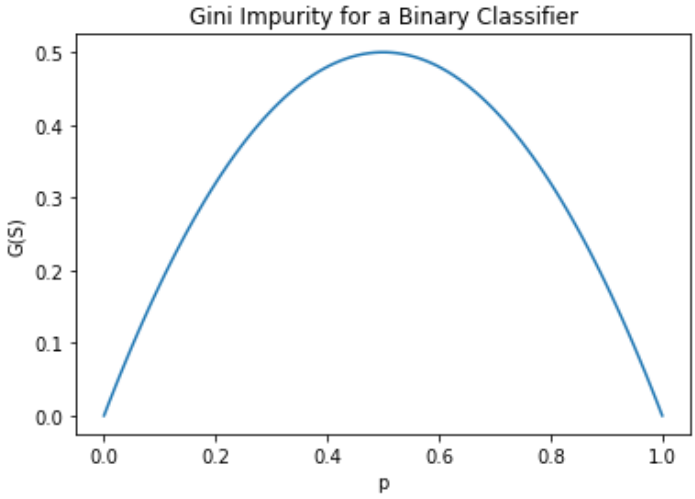


Figure 2.2.2: Gini Impurity for a Binary Classifier Decision Tree

We can determine optimal split at a node by looking at the Information Gain. To calculate Information Gain, first the algorithm calculates the Impurity of the parent node G(parent). Once the split is made based on a feature and threshold, the weighted average Impurity of the children is taken:

$$\overline{G(children)} = \sum_{i=1}^{K} p_i G_i \text{ [Eq. 2.2.3 ]}$$

where $p_i$ is the percentage of data which is split into each node

$$p_i = \frac{S_i}{S} \; [Eq. 2.2.4 \,].$$

By taking the weighted average, we can differentiate between small subsets with high Impurity and large subsets with low Impurity, which are preferred. The Information Gain H is the difference between the parent's Impurity and the weighted average of the children's Impurities:

$$H = G(parent) - \overline{G(children)} \; [Eq. 2.2.5 \,]$$

By maximizing Information Gain, the classifier learns which feature and threshold most effectively splits the dataset at each node. The algorithm measures the Information Gain for all features and thresholds. For n datapoints, each with j features, the number of threshold values that result in unique datapoint distributions is just n, so the number of possible splits at each node is $n * j$. Although the algorithm must check all these features/thresholds at each node in the tree, modern computers can perform these calculations extremely cheaply, making decision tree classifiers a surprisingly fast algorithm even with large datasets.

However, a single decision tree is prone to error. The decisions it makes along each branch of the tree may not generate nodes most effectively or efficiently. The parameters it uses may not accurately represent the classes into which data is meant to be placed. Because the algorithm continues to create nodes until all leaves are pure (or as pure as they can be), decision trees are very prone to overfitting, which occurs when the model performs well on training data but does not generalize. These problems make decision trees poor classifiers by themselves. To avoid these pitfalls, data scientists use what is known as a "random forest."[33,34]

A random forest classifier uses an ensemble of decision trees to create a "forest" and uses the aggregate score to classify each data point. It is important to note that each decision tree must be independent (or nearly independent) of any other tree. The algorithm ensures independence in a few fundamental ways.

The first method that random forest classifiers use to establish independence among trees is through a method known as "bagging." Bagging involves subsampling the dataset S with replacement into m subsets. Each subset still contains n number of datapoints, but the datapoints are randomly selected from S and can be selected more than once. Each subset will randomly feature some points more than others. Each tree overfits in the end, but in theory their errors will cancel with each other. Another benefit that bagging provides is that we can get some sense of error.

Another method of reaching independence is random feature selection. Normally, a decision tree has all d features available to it at each node and decides which feature and threshold most effectively divides the data into nodes. However, a *random* forest will randomly select $k < d$ features available at each node. This ensures that each tree uses different parameters when selecting branches at each node. While k is a hyperparameter that can be set by the researcher, typically $k = \sqrt{d}$ (rounded up) is the optimal value.

Each decision tree produces a classifier $h_j(x)$. The random forest classifier H(x) is the average of each decision tree classifier:

$$H(x) = \frac{1}{m}\sum_{j=1}^{m} h_j(x) \ \ [Eq.\ 2.2.6\ ].$$

Typically, a fraction of the dataset is not used for training the algorithm. This set is called the "validation set," and typically consists of around 20% of the total dataset. After the classifier is trained, the validation set is given to it. While we know the classes $y_i$ of this set, the classifier itself will not be told what class each datapoint belongs to. In the case of a random forest, H(x) is used to predict the class of each datapoint in the validation set. Because each tree is independent, the "wisdom of the masses" is a much more stable and reliable classifier.[64]

### 2.3 Neural Networks, Deep Learning, and Computer Vision

Artificial Neural Networks (ANN's) were first proposed in a rudimentary form by McCulloch & Pitts (1943) and were meant to simulate the human brain's nervous system.[50] However, the computational power required to simulate – or even approximate – the human mind was beyond the technology of the time. It wasn't until the computer graphics card boom of the 1990's, driven largely by the advent of and consumer demand for 3D video games like id Software's *Quake*, that machine learning scientists could revisit this idea and begin making effective use of it. Graphics processing units (GPUs) specialize in performing complex matrix operations, which are the same mathematical operations that neural networks are designed to perform.

An ANN's primary task is to take data $x^d$ and generate a function h(x) to generate meaningful output. While ANN's have been utilized for speech recognition such as Apple's Siri and Amazon's Alexa, they are also robust tools used for image recognition and are sometimes referred to as "computer vision" algorithms when dealing with image data.

In the case of images, input data $x^d$ takes the form of pixel values. Each pixel $x_i$ needs to first be mapped to a feature space $\Phi(x_i)$, which takes the form

$$\Phi(x_i) = \sigma(Ux_i + b)$$

where $U \in \mathbb{R}^{h \times d}$, $b \in \mathbb{R}^{1 \times d}$, and σ(x) is called an activation function. Next, $\Phi(x_i)$ is used as input for the classifier function:

$$h(x) = w^T \Phi(x_i) + C$$

where w is a weight, T indicates the transpose matrix operation, and C is a constant. In the end, h(x) is weighted sum of non-linear activation functions. The ANN learns to optimize $\Phi(x_i)$ and w during training. It does this by minimizing the loss function during each training epoch:

$$L = \sum_{i=1}^{n} \ell(h(x_i), y_i).$$

For a binary classifier, the binary cross entropy loss function is used:

$$L = -\frac{1}{n}\sum_{i=1}^{n} y_i \log\big(h(x_i)\big) + (1 - y_i) \log\big(1 - h(x_i)\big) \ [Eq.\ 2.3.1]$$

Gradient descent is used to find the minimum of L. Using chain rule, the gradient of the loss function is calculated with respect to w and U:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \cdot \frac{\partial L}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n} \left(\frac{y_i}{h(x_i)} - \frac{1 - y_i}{1 - h(x_i)}\right) \Phi(x_i)$$

$$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial \Phi} \cdot \frac{\partial \Phi}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial a} \cdot \frac{\partial a}{\partial U} = -\frac{1}{n}\sum_{i=1}^{n} \left(\frac{y_i}{h(x_i)} - \frac{1 - y_i}{h(x_i)}\right) w^T \Phi(x_i) \frac{\partial \sigma}{\partial a}$$

Where $a = U\Phi(x_i)$ and $\frac{\partial \sigma}{\partial a}$ depends on which activation function is used. If a feature crosses a threshold t, the neuron "fires" and activates the function (1). Otherwise, the neuron does not activate the function (0). Because ANN's were inspired by human anatomy, the sigmoid function

[Figure 2.3.1: Artificial Neural Network Activation Functions (Left)] was used as the activation function in early ANN's:

$$S(x) = \frac{1}{1 + e^{-x}} \; [Eq. 2.3.2]$$

However, unlike our brains, ANN's use gradient descent to minimize L. The sigmoid function asymptotically approaches a maximum of 1, so it has a zero gradient for larger values of x. Modern ANN's use the Rectified Linear Unit (ReLU) function [Figure 2.3.1: Artificial Neural Network Activation Functions (Right)], a non-linear function that always has a non-zero gradient past the threshold t:

$$f(x) = \max(t, x) \; [Eq. 2.3.3].$$

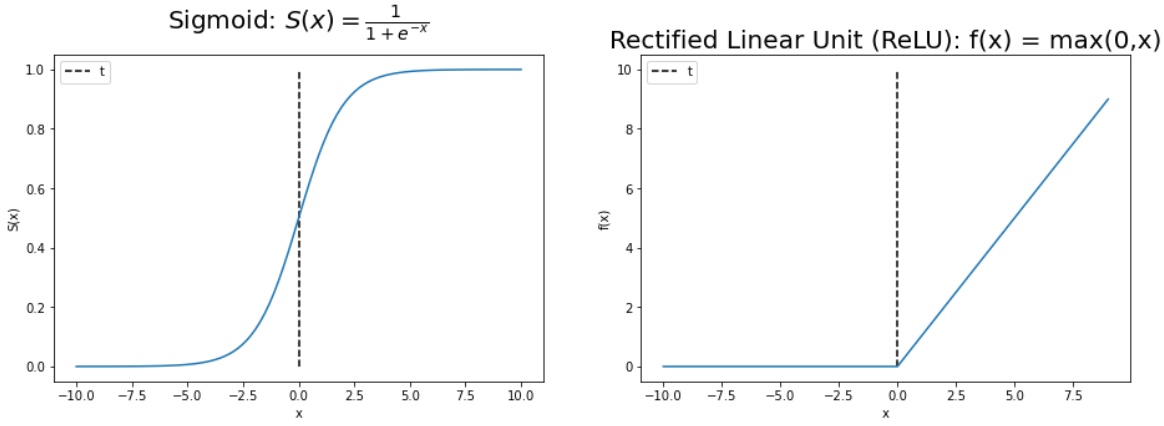


Figure 2.3.1: Artificial Neural Network Activation Functions. (Left) Sigmoid. (Right) ReLU.

Even in modern binary and multi-class classifiers the sigmoid function is still used right before the output layer, which returns the probability between 0 and 1 of a datapoint belonging to each class.

Gradient descent is then applied to update w and U during each training epoch:

$$w \to w - \alpha \frac{\partial L}{\partial w}$$

$$U \to U - \alpha \frac{\partial L}{\partial U}$$

where α is a hyperparameter associated with the learning rate and determines the size of the step during each training epoch. The algorithm "steps" through the feature space along the largest (negative) gradient and iterates on w and U at each step, eventually settling into a good local minimum. Generally, the sum of all feature (pixel) gradients is used to update U and w during each training epoch. For example:

$$\frac{\partial L}{\partial U} = \sum_{i=1}^{n} \frac{\partial \ell(h(x_i), y_i)}{\partial U}$$

However, ANN's use *stochastic* gradient descent, where only a single, randomly selected feature is used to approximate the gradient:

$$\frac{\partial L}{\partial U} \approx \frac{\partial \ell(h(x), y)}{\partial U}$$

While this is a poor approximation in and of itself, ANN's use it to effectively undergo a random walk toward the minimum of L.

The feature space may have several local minima, and some will be better than others. For instance, there may be a sharp, narrow minimum. Even if it happens to be the global minimum, it will not generalize well to real data, effectively overfitting the network. Once this "pit" has been fallen into, a gradient descent algorithm would always be trapped in it, since the local gradient will never point away back out of the hole. However, the algorithm will never land on the true

global minimum, so stochastic gradient descent will give the algorithm a chance to randomly walk out of any narrow minima in the feature space. Only the wide, stable minima will truly trap the function so that it will not randomly "walk" out. By minimizing to these stable minima, the classifier function h(x) will generalize well to real data and will avoid overfitting.

Additionally, modern ANN's implement adaptive learning rates to optimize training. The learning rate α is initially large, and the network will take big steps through the feature space. Once it has settled into a wide, stable minimum from which is cannot escape, it will randomly bounce around the true local minimum. At this point, the learning rate is reduced so that the network takes smaller steps across the feature space. Although the network will never hit the true local minimum, by gradually reducing the learning rate it will get as close to it as possible without overfitting.

The gradient descent algorithm will always follow the true gradient to find a minimum. However, it may stumble into a poor local minimum. Due to the random walk nature of stochastic gradient descent, the algorithm settles on a stable minimum faster than gradient descent. This makes ANN's more effective and less computationally costly to train.

The final function h(x) is a sum of individual activation functions. The complexity of h(x) can be increased by increasing the dimensionality of U. However, a more effective way of increasing the complexity is by implementing layers within the neural network:

$$\Phi(x_i) = \sigma(U\Phi'(x_i) + b)$$

$$\Phi'(x_i) = \sigma(U'\Phi''(x_i) + b')$$

$$\Phi''(x_i) = \sigma(U''\Phi'''(x_i) + b'')$$

The more layers the neural network implements, the more complex functions can be generated during training. Using backpropagation, stochastic gradient descent can be implemented throughout the network's layers to generate extremely complex classifier functions. The final form or h(x) after training is often unknown to the user.

ANN's described up to this point are called full-connected neural networks, since $\Phi(x)$ is applied to each data point at every layer. However, images are typically locally invariant; the objects present in the image should be able to undergo translation, reflection, and possibly rotation and still be recognizable. An array representation of and image will vary widely after undergoing these transformations. To make image classifiers more robust in their training, image convolution and pooling layers are introduced into the network. This type of ANN is known as a convolutional neural network (CNN).

CNN's alter $\Phi(x)$ at each layer to adjust for image classification. A small matrix, called a kernel, is propagated across the image to create a feature map. Each layer applies several different kernels to the input to create a multi-dimensional array and are designed to extract specific features commonly used in image classification. The best convolutions to apply for a given task are learned during training using backpropagation and learned weights. It is important to note that non-linear activation functions are still applied in these layers to "activate" $\Phi(x)$ and to eventually generate h(x).

Once kerels have been applied to the input, a pooling layer is used to down-sample the data and reduce computational cost in deeper layers of the network. Max pooling and average pooling are the most common method of down-sampling because they effectively reduce the data size while still retaining enough information about the features that exist within an image.

Early layers of the CNN extract simple features: horizontal/vertical/diagonal lines, circles, etc. Deeper layers can extract more complex features using backpropagation and learned weights. For example, deeper layers of a facial recognition algorithm would learn to recognize a person's eyes as a combination of horizontal/vertical lines and circles. More complex features can be recognized the deeper the CNN is. The final layers of a CNN are structured like fully connected neural networks as described earlier, where the input is the convolved/pooled feature maps extracted in earlier layers in array form. Input to the fully connected layers of a CNN can consist of millions of features mapped in the convolution and pooling layers.[64]

### 2.4 Performance Measures

When applying a machine learning algorithm, it is important to optimize it. One can begin optimization while preparing the training data set from which the system will learn. When designing the training data, it is important to use a balanced dataset. For example, only about 10% of observable galaxies are undergoing merger events. Providing a dataset to a galaxy merger classifier where 90% of datapoints fall into one class, the classifier might learn that it is best to classify every datapoint as being a member of that class – the classifier will still be correct 90% of the time. Therefore, it might be wiser to provide a balanced training dataset, where half of the galaxies belong to each class.

Another optimization that should be made to any training set is through "feature engineering." Feature engineering is the practice of training a machine learning algorithm on relevant features.[29] For example, in a random forest classifier, it is important to train a morphology classifier on parameters that are good indications of a galaxy's Hubble class, such as Sersic Index or Color Index. The galaxies in the CANDELS fields have over 500 parameters measured for each, but most of these features are irrelevant to classifying their morphologies.

Therefore, it would be useless, as well as computationally and time consuming, to train an algorithm on non-applicable variables.

It is important to avoid overfitting and underfitting a model on training data. Overfitting occurs when a model is trained to perform very well on training data but does not generalize well.[29] This typically occurs when the classifier function h(x) is too complex (i.e., using a high-degree polynomial function to fit linear data). Underfitting occurs when a model is too simple to detect patterns in the data. Underfitting and overfitting can both me mitigated by using feature engineering and hyper-parameter tuning to generate a model which generalizes well to validation and real-world data.

Once the model and the training data are optimized, the program is tested on a validation dataset. A confusion matrix [Figure 2.4.1] can be used to visualize the accuracy. By using the values in the confusion matrix, one can calculate performance metrics for the model.

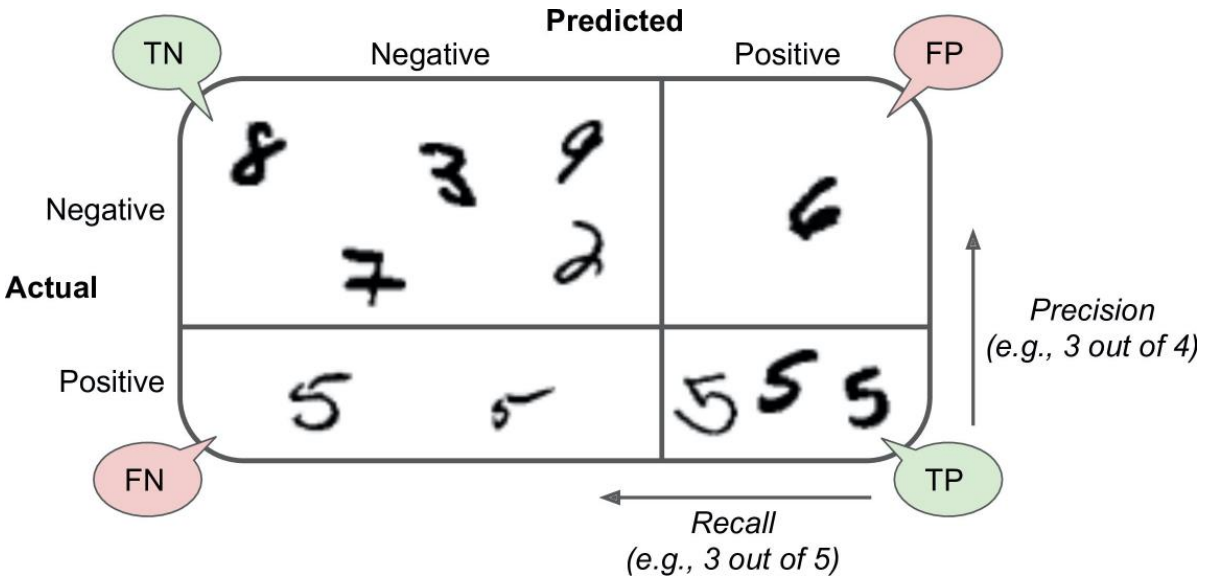


Figure 2.4.1: Confusion Matrix. A confusion matrix for an algorithm that classifies MNIST digits as 5's and not-5's.[29]

The first performance metric is the most intuitive. Accuracy tells the user the percentage of correct classification predictions of the validation data points.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \; [Eq. 2.4.1]$$

where TP, TN, FP, and FN are the number of true positive, true negative, false positive, and false negative predictions, respectively.

The next performance measure, precision, indicates the accuracy of positive predictions:

$$Precision = \frac{TP}{TP + FP} \; [Eq. 2.4.2]$$

Precision is often paired with another metric called recall, which is the ratio of correct positive identifications.

$$Recall = \frac{TP}{TP + FN} \; [Eq. 2.4.3]$$

Recall is also known as the True Positive Rate (TPR).

It is important to note that, in practice, precision and recall are a tradeoff. As a model has higher precision, the recall will suffer and vice-versa. When training a machine learning model, it is up to the designer to decide whether recall or precision is more important. In this study, recall was prioritized since the goal was to train a model that will correctly classify as many positive datapoints (mergers) as possible, even if some false positives (non-mergers) were captured by the classifier as well.

Precision and recall metrics are often combined into a single metric called the F1 Score. The F1 Score of a model is the harmonic mean of precision and recall:

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad [Eq.\ 2.4.4]$$

Because the F1 Score is the harmonic mean, its value can range from 0-1, and the only way a classifier can earn a high F1 Score is if both the precision and recall are high.[29]

A visual way of representing the performance of a binary classifier is by utilizing a receiver operating characteristic (ROC) curve. Originally developed to measure the effectiveness of radar technicians during World War II, and ROC curve plots TPR against the false positive rate (FPR) as the classification threshold is increased, where FPR is calculated using

$$FPR = \frac{FP}{FP + TN} \quad [Eq.\ 2.4.5]$$

A random classifier will generate an ROC curve that is a linear line with slope 0.5 through the ROC space. This line is called the "line of no discrimination" and would be akin to flipping a coin and predicting the outcome. A perfect classifier, meanwhile, will have an TPR of 1 and a FPR of 0 at all thresholds, resulting in a single point in the upper left corner of the plot.

Real world classifiers, of course, lie somewhere between these two extremes. The better a classifier performs, the closer the line will approach the upper left quadrant of ROC space. On the other hand, a classifier that makes worse than random predictions will generate an ROC curve that approaches the lower right quadrant. However, one could invert the predictions of a worse-than-random classifier to make a good classifier. ROC curves are quantified by the area under the curve (AUC):

$$AUC = \int_0^1 TPR\big(FPR(x)\big)dx = \int_{-\infty}^{+\infty} TPR(T)FPR(T)dT \quad [Eq.\,2.4.6]$$

Because TPR and FPR are normalized, AUC ranges from 0 to 1, where better classifiers have a larger AUC.[29]

# CHAPTER 3 METHOD

## 3.1 Random Forest Data Preparation

Because CANDELS researchers had already measured galaxy data and morphological parameters as described in Section 1.5, a random forest classifier was applied first to the datasets. The data, including all five CANDELS fields, consisted of 186,435 galaxies. Any galaxies with H > 24.5 were removed, and only galaxies with mass > $10^9$ $M_\odot$ - estimated by measuring the galaxy's luminosity - and 0.5 < z < 2.5 were retained since fainter, less massive – therefore fainter - and more distant galaxies led to unreliable human morphological and interaction classifications. Human classifications are a normalized percentage of the votes that the class received from human classifiers. VC_F_MERGER, VF_F_ASYM, and VC_F_TIDAL were the percentage of human classifiers who said each galaxy was undergoing a merger event, was asymmetrical, and/or exhibited tidal arms, respectively. However, because merging galaxies are complex, dynamic systems that can exhibit one or more these traits, a merger fraction (MERGE_FRAC) was developed as the average of the three human classification fractions:

$$MERGE\_FRAC = \frac{VC_{F_{MERGER}} + VC_{F_{ASYM}} + VC_{F_{TIDAL}}}{3} \quad [Eq.\ 3.1.1]$$

In this study, the classifier was trained separately to identify galaxy mergers using each classification – VC_F_MERGER, VC_F_ASYM, and VC_F_TIDAL – as well as the MERGE_FRAC threshold. Different thresholds were used to determine positive classes for each individual trait based on the number of galaxies that exhibited each trait. The relatively small number of data points available for training and validation is the biggest limiting factor in this study, so thresholds were higher for traits that more galaxies exhibited and are outlined in Table 3.1.1. Any galaxy that had a MERGE_FRAC score greater than or equal to 0.66 was classified as

a merger, and the remaining galaxies were considered non-mergers. This ensured that any merger-class galaxy would exhibit at least two of the traits.

The features on which the classifier was trained are Gini, M20, concentration, asymmetry, mass, star formation rate (SFR), and galactic half-light radius (rkpc), so only galaxies where those parameters were measured and calculated were included in the dataset. Finally, the dataset was balanced so that an even number of merger and non-merger galaxies was retained. Since the number of non-mergers greatly outweighed mergers before this step, a number of non-merger galaxies equal to the number of mergers was randomly selected from the data pool to be included and were required to have a VC_F_MERGER, VC_F_ASYM, VC_F_TIDAL, or MERGE_FRAC of 0.

| **Feature** | **Positive Class Threshold** | **Total Data Points** | **Training Data Points** | **Positive Class Training Points** | **Negative Class Training Points** | **Validation Data Points** | **Positive Class Validation Points** | **Negative Class Validation Points** |
|---|---|---|---|---|---|---|---|---|
| **Merger** | 0.6 | 566 | 424 | 204 | 220 | 142 | 79 | 62 |
| **Tidal Arms** | 0.6 | 1272 | 947 | 489 | 458 | 316 | 146 | 170 |
| **Asymmetric** | 0.8 | 12682 | 9439 | 4757 | 4682 | 3146 | 1583 | 1563 |
| Merger fraction | 0.6 | 570 | 424 | 217 | 209 | 142 | 68 | 74 |

Table 3.1.1: Random Forest Classifier Data Summary

Data points were randomly distributed into a training set and a validation set, with 75% of the data points being used for training and the remaining 25% used for testing and validation. Because data was randomly distributed among the two subsets, they remained approximately balanced, although there were small variations in the class distribution.

### 3.2 Random Forest Classifier Optimization and Implementation

The Sci-kit Learn RandomForestClassifier() function, initialized to a random state, was used as the classification algorithm, and the RandomizedSearchCV() function was used to search for the optimum hyperparameters.[26] The function randomly selects hyperparameter settings from

user-defined values and uses five-fold cross-validation of the training data to minimize Gini impurity. This process is repeated until the optimum hyperparameters are found. The optimized hyperparameters determined by the RandomizedSearchCV() function are summarized in Table 3.2.1. The random forest classifier was then trained on the training data set with the optimized hyperparameters.

| Hyperparameter | Description | Value |
|---|---|---|
| n_estimators | Number of trees in the forest | 140 |
| max_features | Number of features considered at each tree node | 2 |
| max_depth | Maximum level in each decision tree | None |
| min_samples_split | Minimum number of datapoints considered at each node before the node is split | 2 |
| min_samples_leaf | Minimum number of datapoints allowed in each leaf node | 4 |
| bootstrap | Method for sampling (True = with replacement, False = without replacement) | False |

Table 3.2.1: Random Forest Classifier Hyperparameters. Hyperparameters, descriptions, and optimized values used in the random forest classifier.

### 3.3 Computer Vision Data Preparation

Because the computer vision algorithms take image data as input, postage stamps of galaxies needed to be generated from the CANDELS field mosaics. The same cuts to Hubble magnitude, mass, and redshift were made to the full dataset. Additionally, the thresholds for positive and negative classifications were retained. However, galaxies where no SFR, rkpc, A, C, G, $M_{20}$

measurements were made could still be included since only image data and human classifications were required, resulting in a slightly increased sample size.

World Coordinate System (WCS) coordinates were also obtained from each CANDELS field mosaic using the WCS() function from Python's Astropy library. Each galaxy's right ascension (RA) and declination (Dec) were extracted from .FITS tables and converted to WCS coordinates using the wcs_world2pix() function. The WCS coordinates were then used to find each galaxy within its corresponding mosaic. Using the Cutout2D() function in Astropy, a 50x50 pixel image of each galaxy was extracted from the mosaic. Pixel value arrays were stacked along the fourth dimension to create one Numpy array for each positive and negative class. To ensure a balanced dataset, the positive and negative data arrays were generated alongside each other, and the number of negative-class images was not allowed to exceed the number of positive-class images.[4]

### 3.3.1 Data Augmentation

Due to the complexity of the features that a neural network is tasked to learn, CNN's typically need thousands, if not millions, of images on which to train. The small number of viable merger images available in the CANDELS fields would be a limitation, so we use data augmentation to artificially increase the sample size.

Data augmentation involves duplicating images while randomly applying transformations to them. While the transformations are subtle enough that the duplicated images still represent the objects contained within them, the pixel distributions of the duplicates will be altered enough that they can be considered different images during training and validation without overfitting.

As an additional form of augmentation, postage stamps of the same galaxy are cut out from multiple mosaics imaged using different WFC3 filters and treated as separate images. The light signal through separate filters is different enough that the feature maps extracted during 2D convolutions will be different. The filters used in this study are F105W, F125W, F140W, and F160W, which are used with the WFC3 IR channel [Table *3.3.1.1*]. The mosaics which were created using these filters were chosen because together they cover a wide enough wavelength range to ensure that the galaxies included in the dataset – which have a redshift range of 0.5-2.5 – would still appear visually in the image postage stamps. The more highly redshifted galaxies appeared clearer in the higher wavelength filters. Postage stamps that did not contain a visible image of the galaxy were manually removed from the dataset post-augmentation.

| **Filter Name** | **λ [nm]** | **fwhm [nm]** |
|---|---|---|
| F105W | 1045 | 310 |
| F125W | 1250 | 300 |
| F140W | 1400 | 400 |
| F160W | 1545 | 290 |

Table 3.3.1.1: WFC3 IR Channel Filters: WFC3 IR filters used for CANDELS mosaics. Separate postage stamps were generated for each galaxy in each of the filters and used for data augmentation.[51]

Unfortunately, not all WFC3 filter mosaics covered the entirety of the CANDELS fields. As a result, some galaxies were in sections of the fields that were not imaged by all filters. These image cutouts appeared empty or cutoff [Figure *3.3.1.1*]. These images were removed from the augmented dataset, except for the asymmetry classifier. Due to the size of the augmented asymmetry data set, it was not feasible to remove empty or corrupted images.

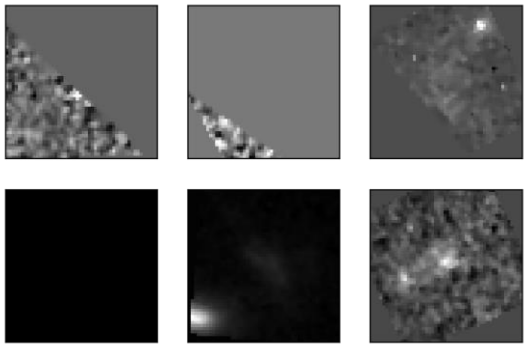

Figure 3.3.1.1: Corrupted Galaxy Images. Examples of galaxy postage stamps that are cutoff, empty, or blown out. Bad postage stamps were removed from training and validation datasets.

In this study, eight-fold augmentation with rotation, vertical and horizontal mirroring, and brightness scaling is applied [Table 3.3.1.2]. The operations are each randomly applied to the duplicates, ensuring enough degrees of freedom to allow for eight-fold augmentation. While size scaling operations were available, they were not applied so that images would not become "stretched" in the x- or y-dimensions. Galaxies undergoing merger events tend to be elongated due to strong gravitational interactions, and learning could be affected if non-merger class galaxy images appeared asymmetrically stretched in their duplicates. The final dataset sizes for each classifier can be found in [Table *3.3.1.3*].

| **Augmentation Operation** | **Rotation** | **Vertical Flip** | **Horizontal Flip** | **Brightness Scaling** |
|---|---|---|---|---|
| **Boolean / Range** | [-180°, 180°] | TRUE | TRUE | [80%, 120%] |

Table 3.3.1.2: Data Augmentation Parameters. Data augmentation transformations are randomly applied – or not applied – to each image. The number of transformations ensures enough degrees of freedom exist to augment the dataset eight-fold without overfitting or generating true duplicates of any image.

| Feature | Positive Class Threshold | Total Data Points (Pre-augmentation) | Total Data Points (Post-augmentation) | Training Data Points | Validation Data Points |
|---|---|---|---|---|---|
| Merger | VC_F_MERGER ≥ 0.6 | 2254 | 16784 | 12575 | 4209 |
| Tidal Arms | VC_F_TIDAL ≥ 0.6 | 4614 | 31826 | 23853 | 7973 |
| Asymmetric | VC_F_ASYM ≥ 0.8 | 34998 | 279885 | 211308 | 68577 |
| Merger fraction | mergefrac ≥ 0.6 | 2284 | 16977 | 13068 | 3909 |

Table 3.3.1.3: Computer Vision Classifier Dataset Sizes. Computer vision classifier dataset sized pre- and post- augmentation. Pre-augmentation numbers include images taken from the four WFC3 filter mosaics.

### 3.3.2 Final Image Preparation

After augmentation, the pixel data from each postage stamp is converted to a Numpy[13] array and stacked in a 50x50xn array, where n is the number of images. As with the random forest classifier, 75% of the images are used for training, and the remaining 25% are used for validation.

Augmented images are stored in separate directories depending on the class to which they belong and whether they are training and validation data (i.e., training merger images are stored in one directory, training non-merger images are stored in another, validation non-merger images are stored in yet another, etc.). 1D Numpy[13] arrays are generated containing class labels, each with a number of elements equal to the number of augmented images for each class/dataset. The label elements for mergers all have a value of 1, and the label elements for non-mergers have a value of 0.

After the augmented image and label arrays have been created, the dataset image arrays are appended to each other, as are their label arrays. The label array is then appended along the fourth dimension of the corresponding image array. The master training and validation data arrays are shuffled along the first dimension so that each image's label is still paired with it correctly. The pixel data is then normalized, and labels are one-hot encoded [Table *3.3.2.1*]. Finally, the data and label arrays are passed to the neural network for training and validation.

| Class Name | Class Label | One-hot Encoded Class Label |
|---|---|---|
| Non-merger | 0 | [0,1] |
| Merger | 1 | [1,0] |

Table 3.3.2.1: One-hot Encoding. Example of one-hot encoding of binary classification labels. One-hot encoding is implemented for the merger, tidal arms, and asymmetry classifier labels.

### 3.4 Convolutional Neural Network Architecture and Training

The CNN architecture used in this work was adapted from Dieleman et. al (2015), which was originally developed to predict galaxy morphologies.[21] The keras.models library was used to construct the model.[55] The network takes 50x50 pixel images as input and first applies a 2D convolution layer. The first layer applies 32 filters to each image with a kernel size of 3x3 and a default step of 1. Following the first convolutional layer is a max pooling layer with a kernel size of 2x2.

Alternating convolution and pooling layers follow until reaching the fully connected layers. Here, the features are "flattened" into one-dimensional arrays and passed through alternating sigmoid and ReLU layers. The final output layer employs the sigmoid function since the output is a one-hot encoded probability of the image belonging to each class. The classifier trains on a total of 6,834,818 features, and the full architecture is outlined in [Figure *3.4.1*].

The number of training epochs was set arbitrarily high so that the model will continue training. The ModelCheckpoint() function was used so that the best-performing model would be saved as training continued.[55] With ModelCheckpoint() the model will predict classes for the galaxies in the validation dataset after each training epoch. If the model performance on the validation set has improved from the previous epoch, the model will be saved as a .h5 file and will move on to the next training epoch.

Additionally, EarlyStopping() was used to monitor model improvement during training.[55] With EarlyStopping(), the model continues training until it no longer improves its performance on the validation set. As described in Section 2.3, the model may fall into narrow local minima during stochastic gradient descent. Functionally, this means the model may not improve its performance for a few training epochs while it is inside of these minima. However, given enough time it may step out of a narrow minimum, and performance will begin improving again until it finds a stable minimum. To avoid this problem, the user can define the number of epochs to continue training after performance stops improving before stopping training altogether. In this work, the model stops training only after it has not improved performance after 10 consecutive training epochs. Used together, ModelCheckpoint() and EarlyStopping() ensures the model has a chance to continue training until it can be verified that performance is not improving and can revert back to the best-performing model.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 64)        51264
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 128)         147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 2048)              2361344
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              4196352
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 4098
=================================================================
Total params: 6,834,818
Trainable params: 6,834,818
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.4.1: CNN Architecture. CNN architecture used in this project, adapted from Dieleman et. al (2015). Convolution layers alternate 2D convolutions and pooling layers. Fully connected layers alternate ReLU and sigmoid activation functions with a sigmoid output layer. The sigmoid activation function output layer is used in binary classifiers.

# CHAPTER 4 RESULTS AND ANALYSIS

## 4.1 Random Forest Classifier Results

After training, the validation data was passed to the random forest classifier. The results from each classifier are found in [Table *4.1.1*], and confusion matrices were generated [Figure *4.1.1*]. Notably, the classifier that was trained on the merger fraction performed significantly better on the validation dataset. This is likely because the positive class for the merger fraction required that two out of three merger signifiers – tidal arms, asymmetry, and the merger flag – were selected by human classifiers. As a result, these galaxies are the most disturbed galaxies in the data sample. The distribution of feature values among each dataset are shown in [Figure *4.1.2*]. An Anderson-Darling k-sample test was performed on each of the positive-class feature spaces.[3] The test is designed to determine whether discrete data sets are sampled from the same population. The results showed that the null hypothesis – that the data come from the same population – can be rejected at the 0.1% level, indicating that each of the samples come from significantly different populations.

Additionally, the importance of each training feature that the model uses for classification was found. The significance of each feature is a normalized value where the sum of all features' importance is 1. The importance of each training feature for each classifier can be found in [Table 4.1.2]. Correctly and incorrectly predicted test data points were plotted by the two most important features for each classifier [Figure *4.1.3*], illustrating the clustering of classes in the space. As expected, most incorrectly classified datapoints in the testing dataset are in the overlap of clusters.

Because the Merger Fraction classifier performed best, and because the performance of each classifier varied slightly with each training and validation session, the Merger Fraction classifier

was run 1000 times. All four performance metrics were plotted for 1000 runs and a normalized curve was fit [Figure *4.1.4*]. However, perfect performance acts as an insurmountable "wall" for the classifier, so the performance distribution is skewed to lower values. Because of this, a skewed Gaussian curve was a better fit for each distribution:

$$f(x) = 2A * exp\left[\frac{-\left(\frac{x-\mu}{\sigma}\right)^2}{2}\right] * \left[1 + erf\left(\frac{\alpha\left[\frac{x-\mu}{\sigma}\right]}{\sqrt{2}}\right)\right] \quad [Eq.\,4.1.1]$$

where A is the peak of the curve, μ is the mean, σ is the standard deviation, exp is the exponential function, and erf is the error function. For all four performance measures, the $R^2$ value for the skewed normal fit was slightly higher than the normal fit.

Finally, ROC curves were also plotted for each classifier. The merger fraction classifier scored the highest AUC, 0.96, reflecting the relatively high precision, recall, and - by extension - F1 score.

| Classifier | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| **Merger** | 84.48% | 0.8823 | 0.7894 | 0.8333 |
| **Tidal Arms** | 91.79% | 0.7251 | 0.7600 | 0.7421 |
| **Asymmetry** | 81.21% | 0.8082 | 0.8170 | 0.8126 |
| **Merger Fraction** | 92.31% | 0.9332 | 0.9428 | 0.9253 |

Table 4.1.1: Random Forest Classifier Performance Metrics. The results for the merger fraction classifier are averaged over 1000 sessions [Figure 4.1.4].

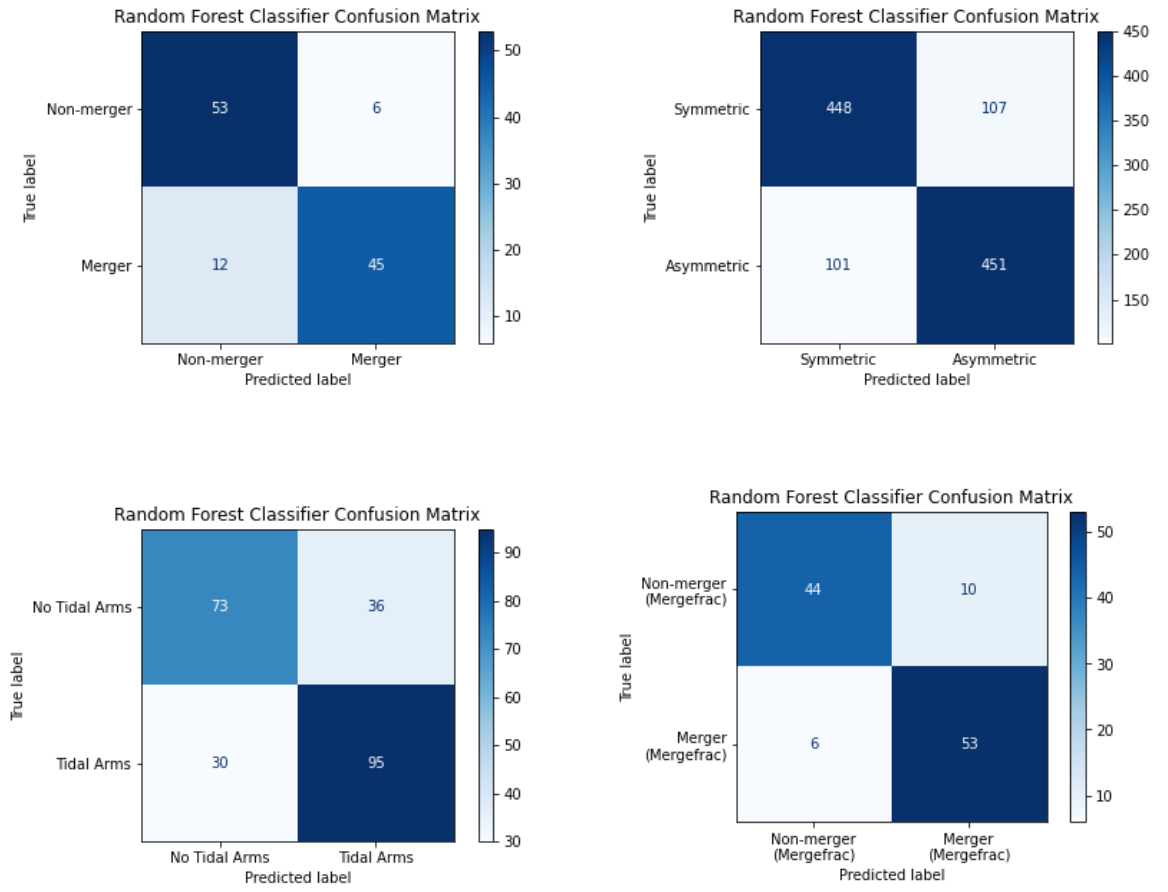


Figure 4.1.1: Random Forest Classifier Confusion Matrices: Merger classifier (upper-left), asymmetry classifier (upper-right), tidal arms classifier (bottom-left), merger fraction (mergefrac) classifier (bottom-right). Within each confusion matrix are the number of true positives (upper-left), false positives (upper-right), false negatives (bottom-left), and true negatives (bottom-right).

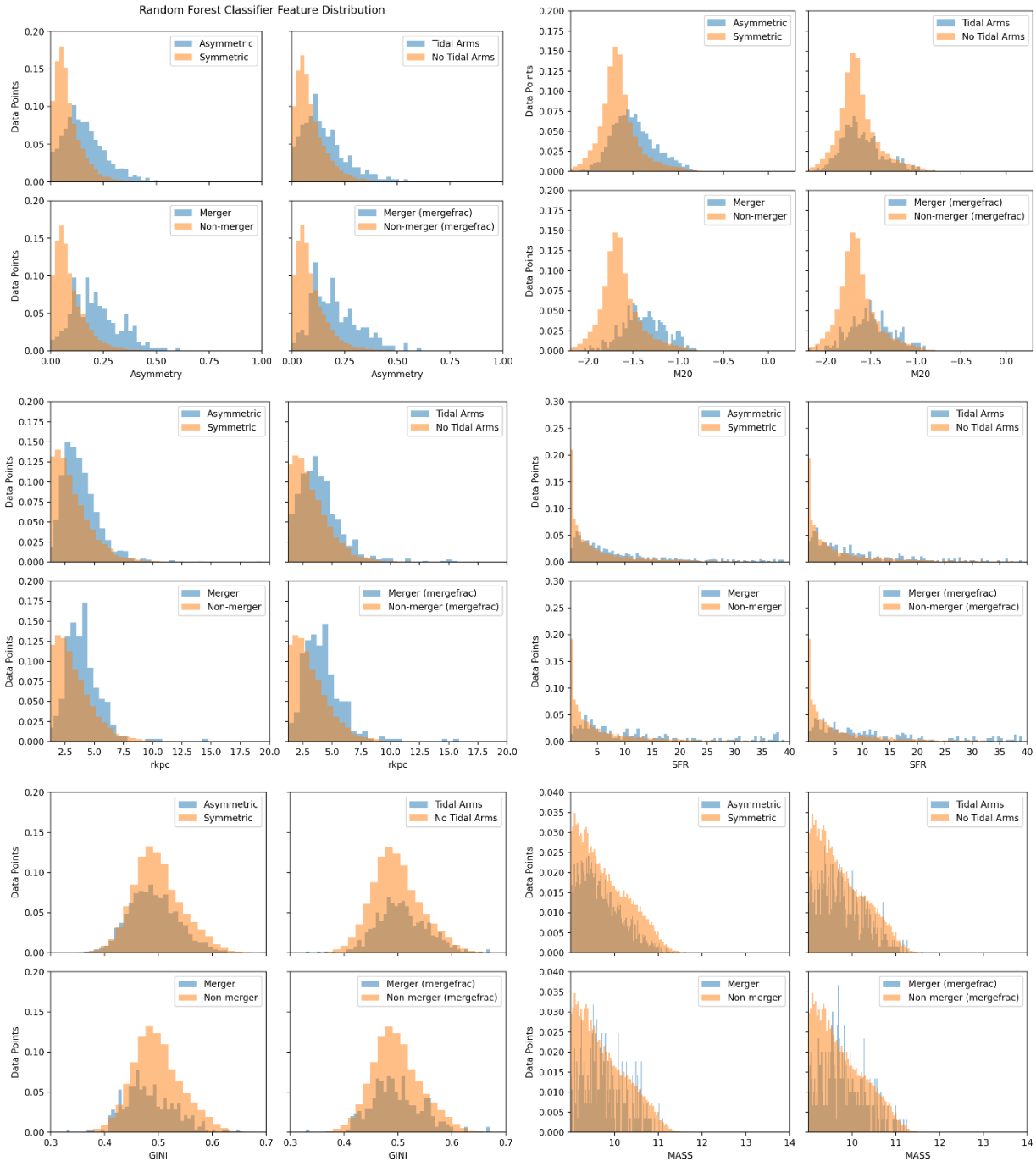


Figure 4.1.2: Random Forest Classifier Feature Distributions. Distributions of features for each random forest classifier. From top left: Asymmetry, $M_{20}$, galactic half-light radius (rkpc), and star formation rate (SFR), Gini coefficient, and Mass. The distributions of Gini coefficients and masses between positive- and negative-class galaxies nearly entirely overlaps, making them poor features for decision making.

| Merger | | Tidal Arms | | Asymmetric | | Merger Fraction | |
|---|---|---|---|---|---|---|---|
| **Feature** | **Importance** | **Feature** | **Importance** | **Feature** | **Importance** | **Feature** | **Importance** |
| Asym. | 0.3217 | Asym. | 0.2394 | rkpc | 0.2353 | Asym. | 0.3771 |
| $M_{20}$ | 0.2385 | rkpc | 0.1970 | Asym. | 0.2092 | rkpc | 0.2184 |
| SFR | 0.1267 | SFR | 0.1466 | $M_{20}$ | 0.1879 | $M_{20}$ | 0.1729 |
| rkpc | 0.1194 | $M_{20}$ | 0.1154 | SFR | 0.1319 | SFR | 0.0982 |
| Conc. | 0.0826 | Gini | 0.1083 | Mass | 0.0832 | Conc. | 0.0557 |
| Mass | 0.0593 | Conc. | 0.0979 | Conc. | 0.0815 | Gini | 0.0393 |
| Gini | 0.0519 | Mass | 0.0953 | Gini | 0.0711 | Mass | 0.0385 |

Table 4.1.2: Random Forest Classifier Feature Importance. Importance of each feature for each random forest classifier. Feature importances for each classifier sum to 1.

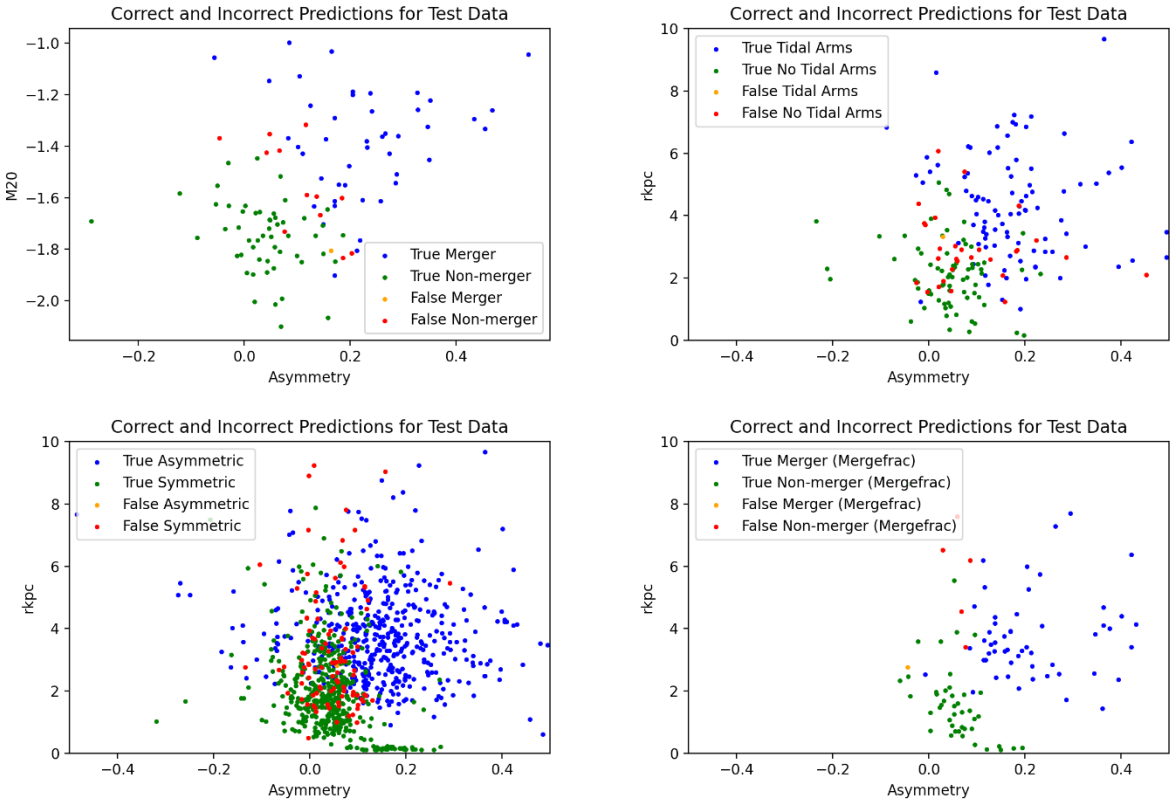


Figure 4.1.3: Random Forest Classifier Predictions. Predictions for test data for each random forest classifier. Data points are plotted in the space of the two highest-ranking features from [Table 4.1.2].

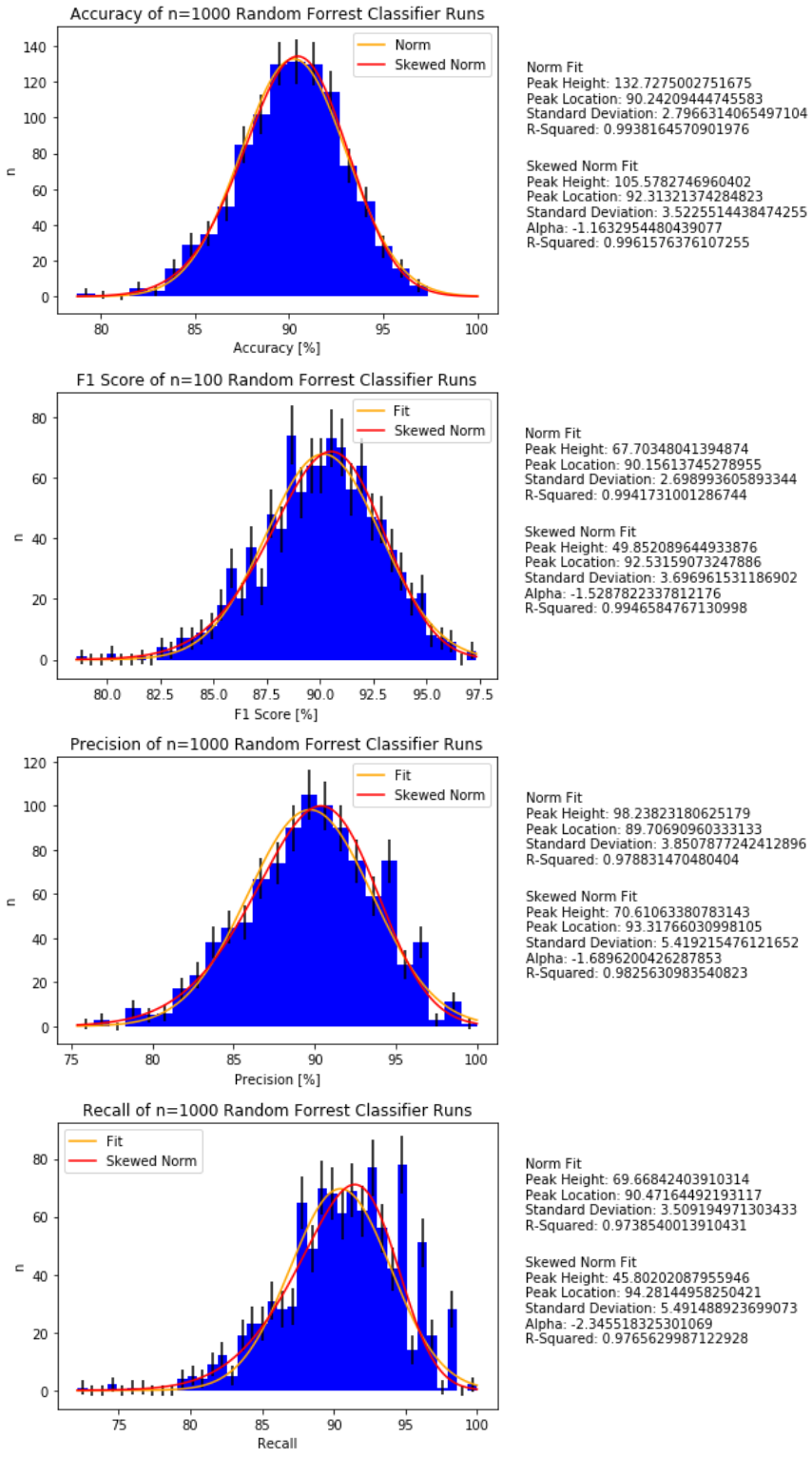


Figure 4.1.4: Merger Fraction Random Forest Classifier Results for n=1000 Sessions. Both a normal and skewed normal curve was fit to each distribution. The skewed normal fit was the better fit for all performance metric, which is reflected in the $R^2$ scores.

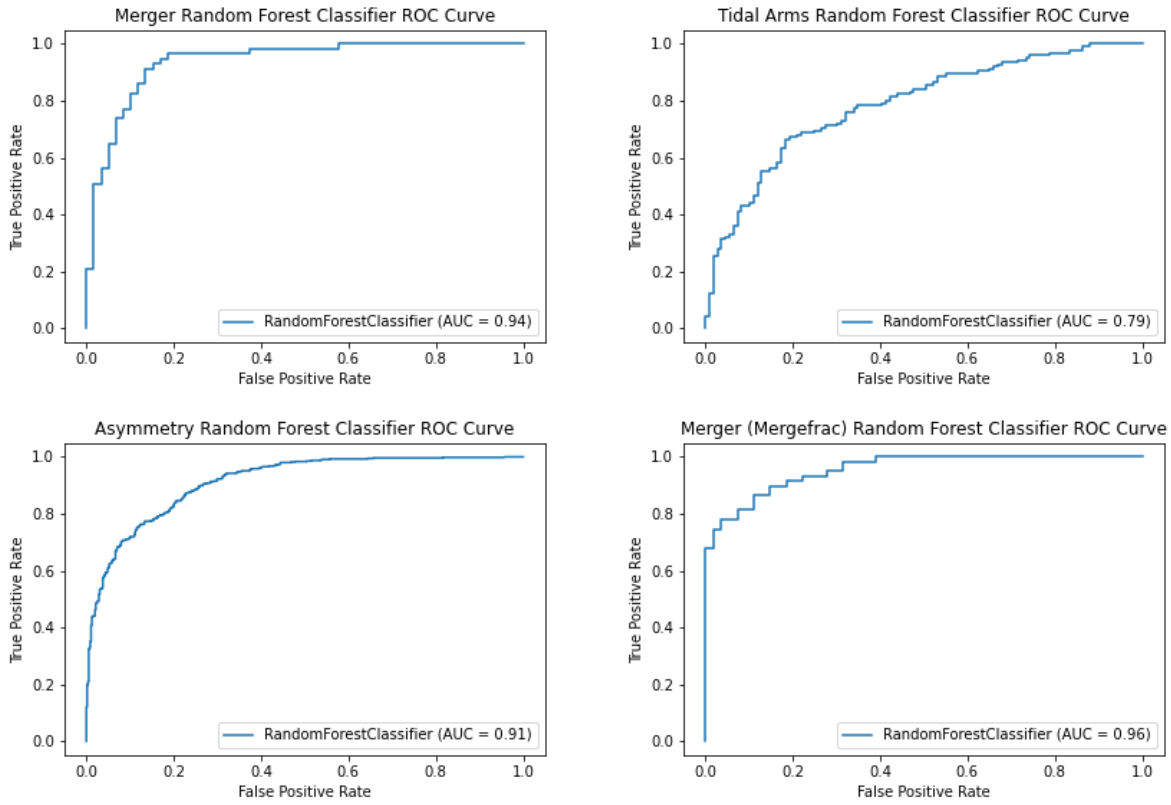


Figure 4.1.5: ROC Curves for Random Forest Classifiers. False positive rate is plotted along the x-axis, and true positive rate is plotted along the x-axis as the classification threshold is increased. AUC scores, which range from 0 (a perfectly wrong classifier) to 1 (a perfect classifier), are included, with the Merger Fraction (mergefrac) classifier scoring highest.

### 4.2 Convolutional Neural Network Results

The convolutional neural network was trained using the data and architecture outlined in Section 3.3. As with the random forest classifier, the merger fraction classifier performed best. This is likely due to the fact that the galaxies selected for the positive class in the merger fraction dataset were the most disturbed, exhibiting at least two of the three merger indicators. The tidal arms, asymmetry, and merger flag classifiers also performed similarly to their random forest counterparts [Table 4.2.1 and Figure 4.2.1].

Because convolutional neural networks undergo several epochs of training, during which the model is constructed and improved, the training and validation accuracy and loss can be plotted for the entire training regimen [Figure 4.2.2 and Figure 4.2.3]. Using the Keras checkpoint function, the model was saved whenever the validation accuracy improved over previous epochs. Each model achieved a maximum accuracy during a different training epoch, with the merger classifier reaching peak validation accuracy last during the twentieth round of training. This is likely due to the complexity of the images and features present in the merger fraction dataset.

The training set loss decreases throughout training as the model learns the features present in the training data. However, as it continues to train it begins recognizing more specific patterns in the training data that it uses to classify the images. In the testing set, this results in loss *increasing*; the classifier is beginning to overfit to the training set and is not generalizing well to the validation set. However, because the model has an early stopping call once loss begins increasing, training will end before the model can truly overfit. It is important to note that loss is not a normalized value and can exceed 1.0. The losses recorded in the final model represent reasonable loss for a binary classifier. This is also reflected in the training/validation accuracies [Figure 4.2.2]. The training accuracy continues increasing, while the validation accuracy does

not. If the model were allowed to keep training past this point, the validation accuracy would begin to decrease even as the training accuracy reached 100%, indicating an overfit model.

As with the random forest classifiers in Section 4.1, ROC curves were plotted for each of the convolutional neural networks [Figure 4.2.4]. ROC curves for the CNNs appear "smoother" than those generated for the random forest classifiers. This is because there are millions of features the model uses to predict classifications, and so the classification threshold can be increased in smaller steps. As expected, the merger fraction classifier scored the highest AUC of all classifiers, with an AUC of 0.951.

For each of the classifiers, a subset of validation image predictions is presented in [Figure *4.2.5* - Figure *4.2.8*]. The tidal arms classifier was expected to perform best, since tidal arms are a relatively easy feature for a human classifier to recognize in an image. However, it ended up performing with the lowest accuracy of the four classifiers. This may be due to the tidal arms only being prominent in some WFC3 filters and not visible in others. Across all classifiers, the misclassified galaxies did not include all – or even multiple – post-augmentation permutations of a single galaxy. This would indicate that only specific permutations of a galaxy, whether by filter or augmentation transformation, were misclassified.

| Classifier | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| **Merger** | 83.75% | 0.8433 | 0.8289 | 0.8361 |
| **Tidal Arms** | 71.79% | 0.7519 | 0.6666 | 0.7067 |
| **Asymmetry** | 81.59% | 0.8226 | 0.8060 | 0.8142 |
| **Merger Fraction** | 87.87% | 0.8683 | 0.8923 | 0.8802 |

Table 4.2.1: Computer Vision Classifier Performance Metrics. Accuracy, precision, recall, and F1 scores for the four computer vision classifiers.

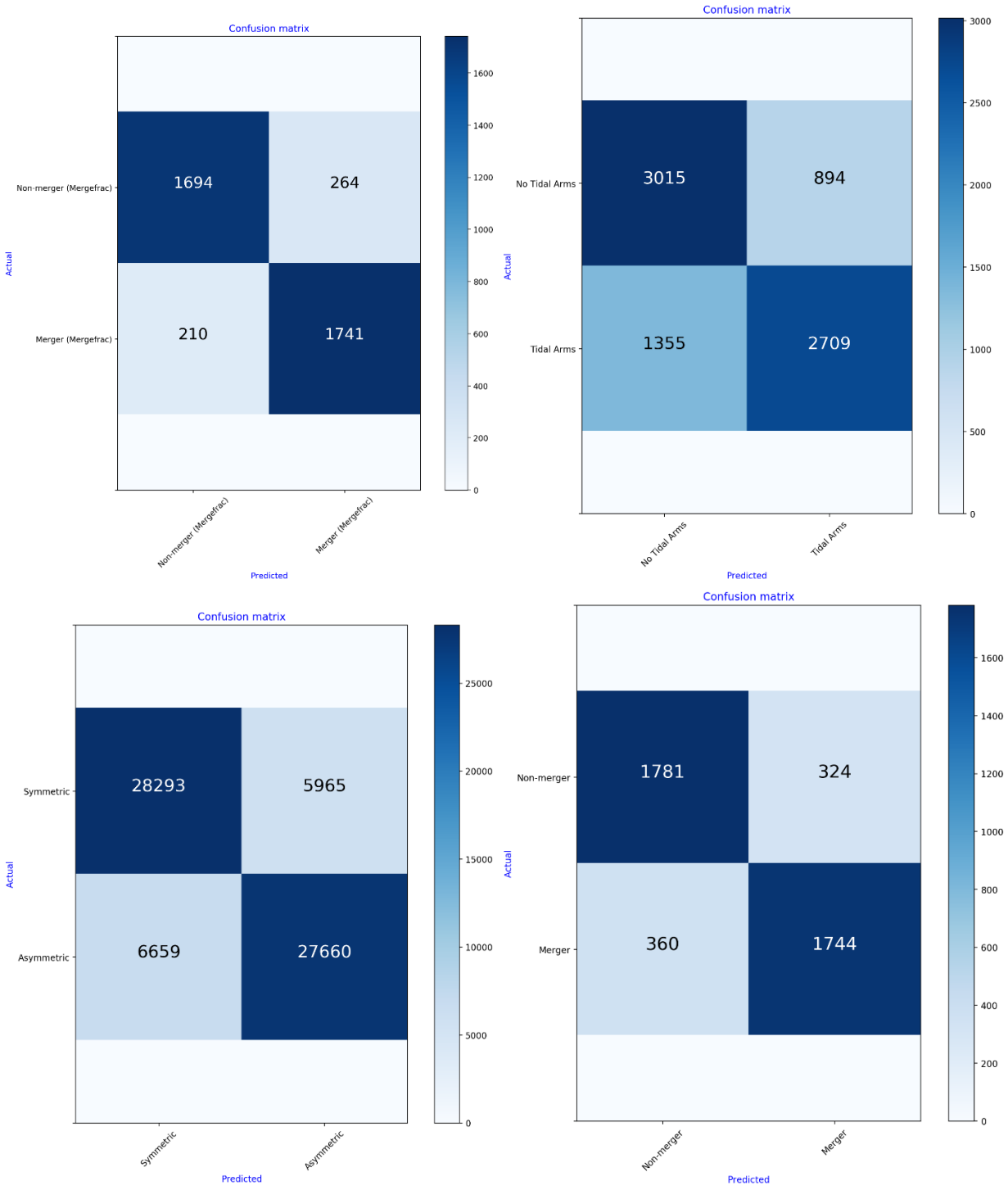


Figure 4.2.1: Computer Vision Classifier Confusion Matrices: Merger classifier (upper-left), asymmetry classifier (upper-right), tidal arms classifier (bottom-left), merger fraction (mergefrac) classifier (bottom-right). Within each confusion matrix are the number of true positives (upper-left), false positives (upper-right), false negatives (bottom-left), and true negatives (bottom-right).

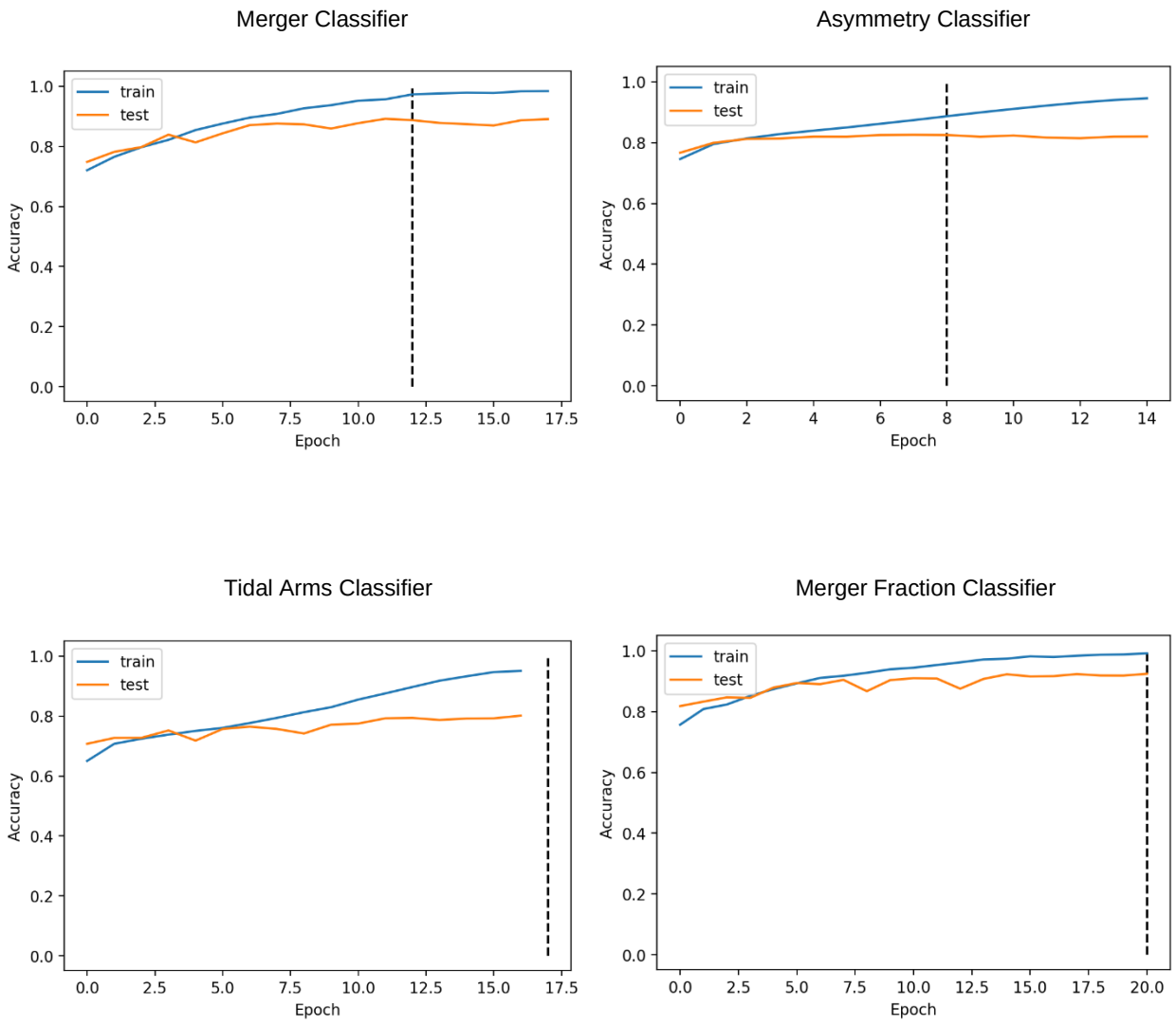


Figure 4.2.2: Computer Vision Classifier Training Accuracy. The accuracy of the training and validation data throughout training epochs is plotted for each of the classifiers: merger classifier (upper-left), asymmetry classifier (upper-right), tidal arms classifier (bottom-left), and merger fraction classifier (bottom-right). The vertical dashed line represents the training epoch from which the final model was saved using early stopping and callback functions. The model with the highest validation accuracy was saved and training ended ten epochs after the minimum loss was achieved [Figure 4.2.3].

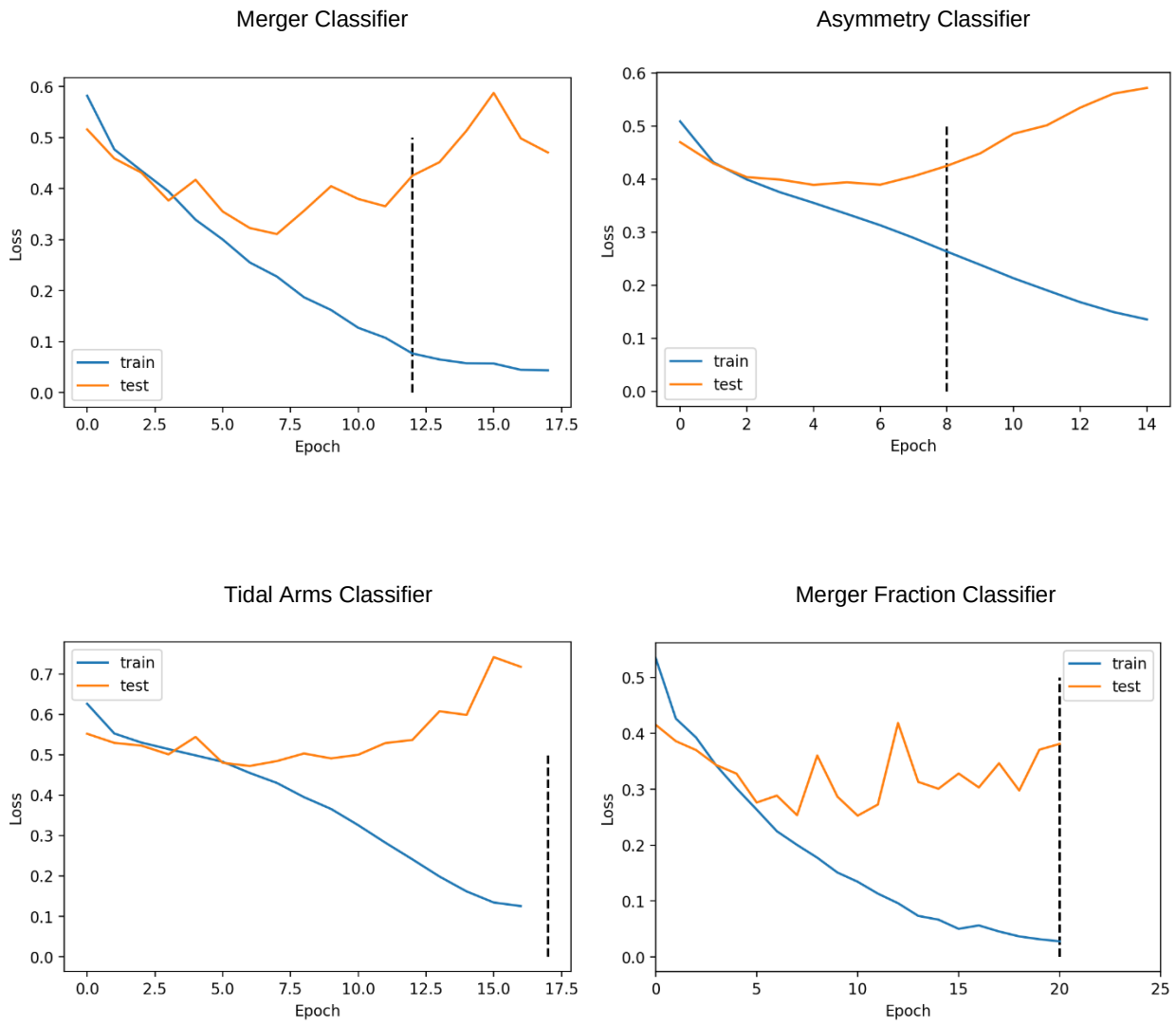


Figure 4.2.3: Computer Vision Classifier Loss. Loss of each computer vision classifier during training: merger (upper-right), asymmetry (upper-left), tidal arms (lower-left), and merger fraction (lower-right). The dashed vertical line represents the training epoch during which the final model was saved using early stopping and callback functions. Training was stopped ten epochs after the minimum loss was achieved and the model was saved when test data accuracy was at its peak [Figure 4.2.2].

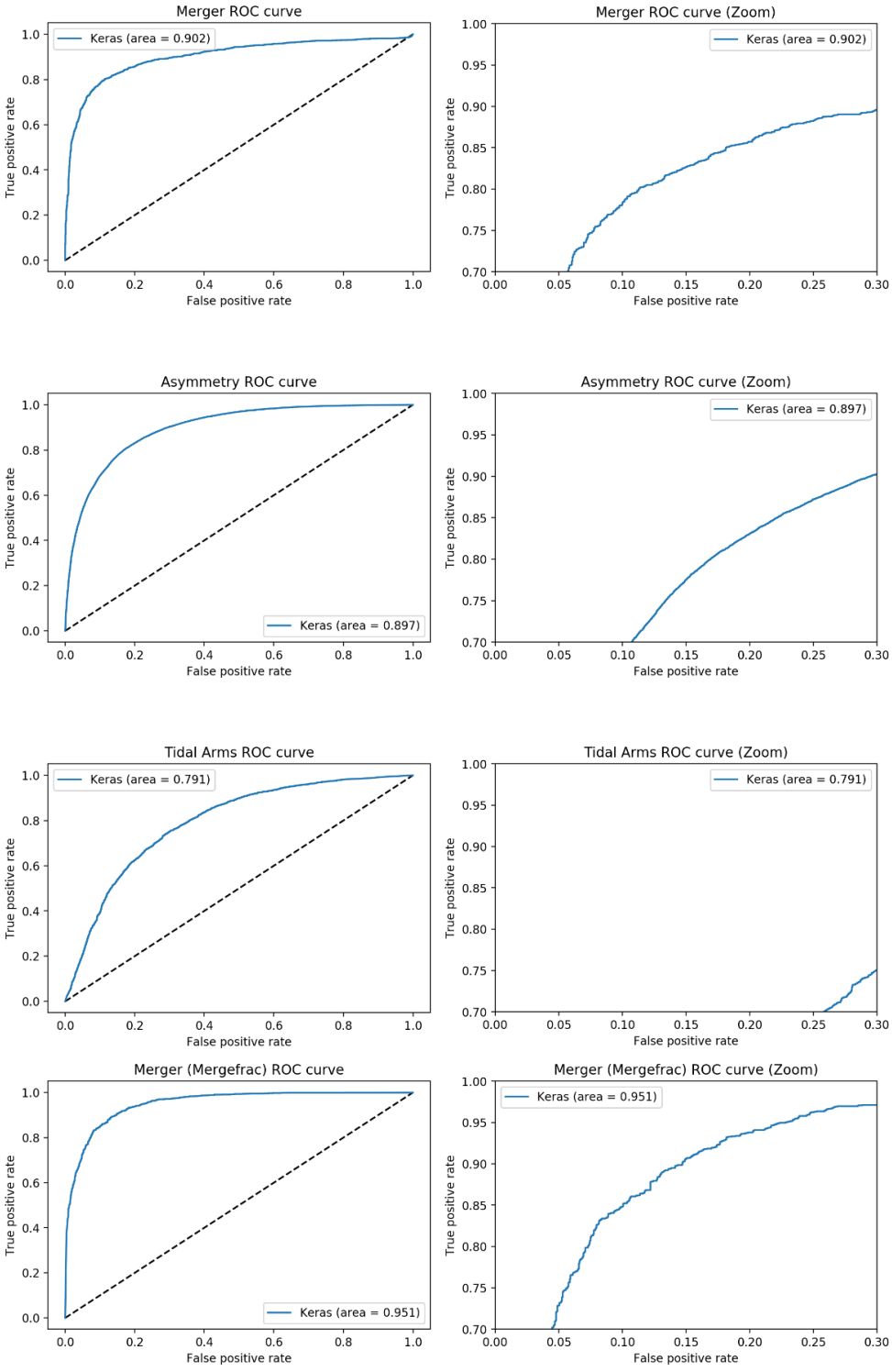


Figure 4.2.4: Computer Vision Classifier ROC Curves. ROC curves for the four computer vision classifiers. The entire ROC curve plots are shown on the left, and a zoomed-in plot of each is on the right.

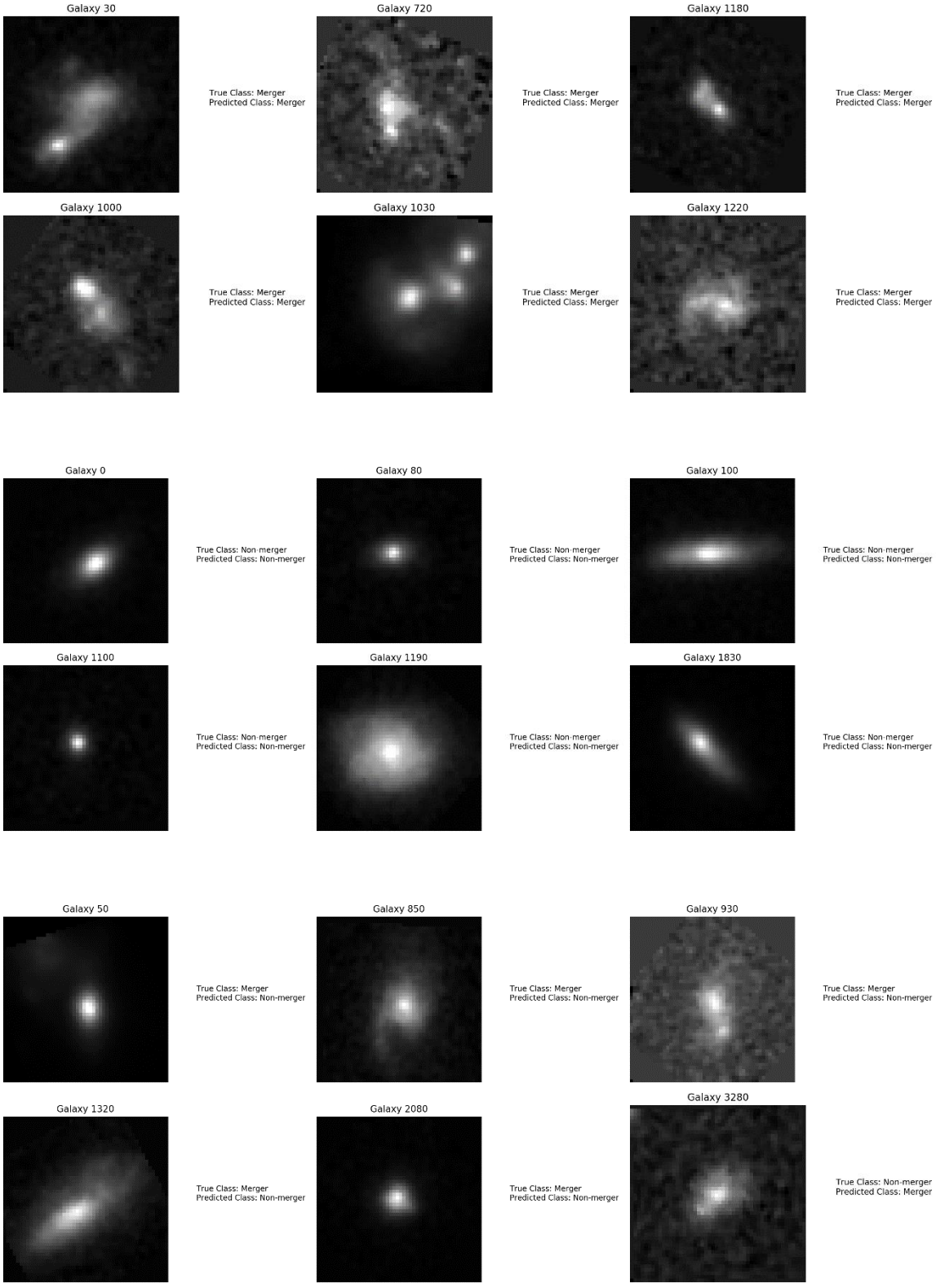


Figure 4.2.5: Merger Classifier Image Predictions. Example image predictions from the validation dataset for the merger classifier. True positives (top), true negatives (middle), and misclassified images (bottom).

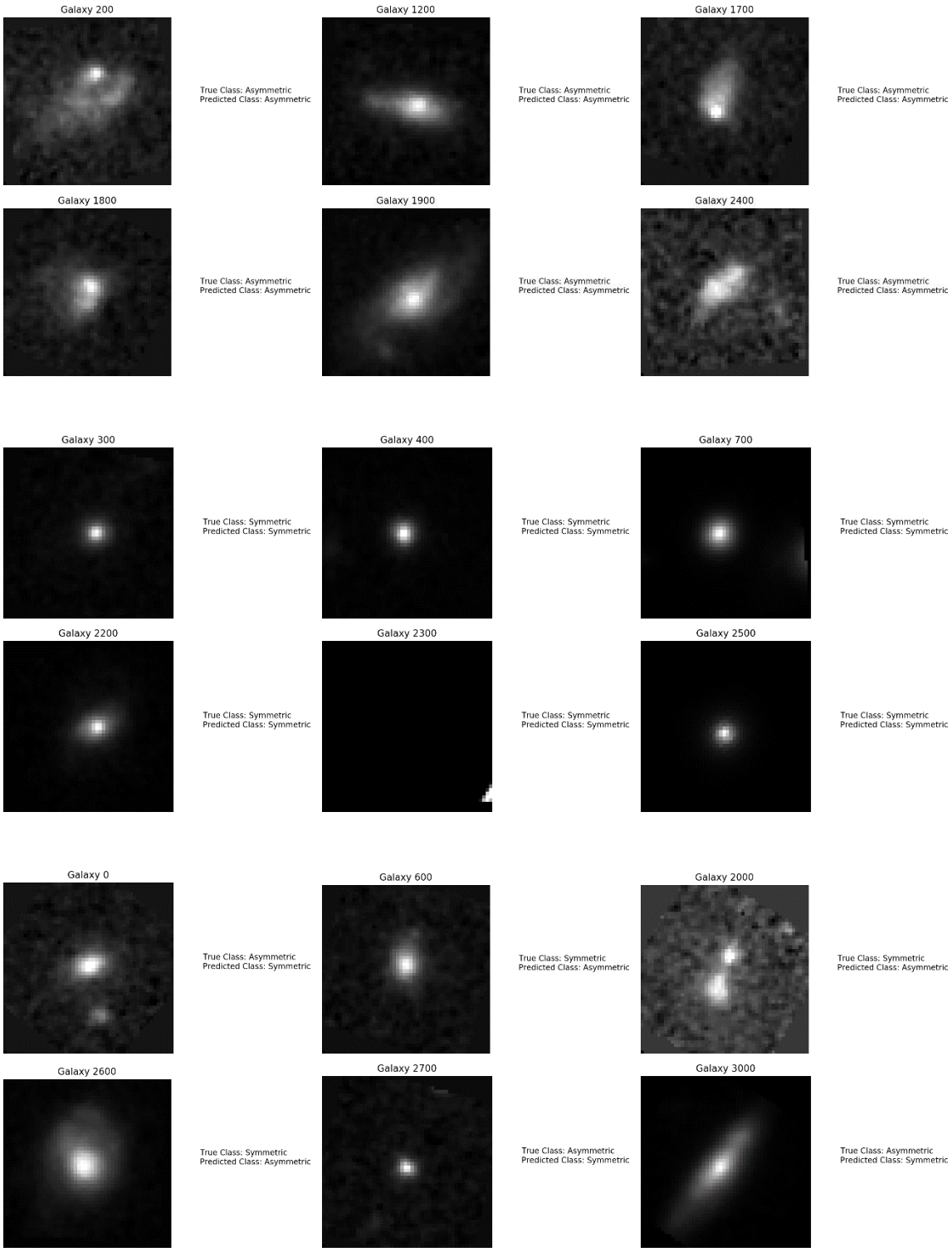


Figure 4.2.6: Asymmetry Classifier Image Predictions. Example image predictions from the validation set for the asymmetry classifier: true positives (top), true negatives (bottom), and misclassified images (bottom).

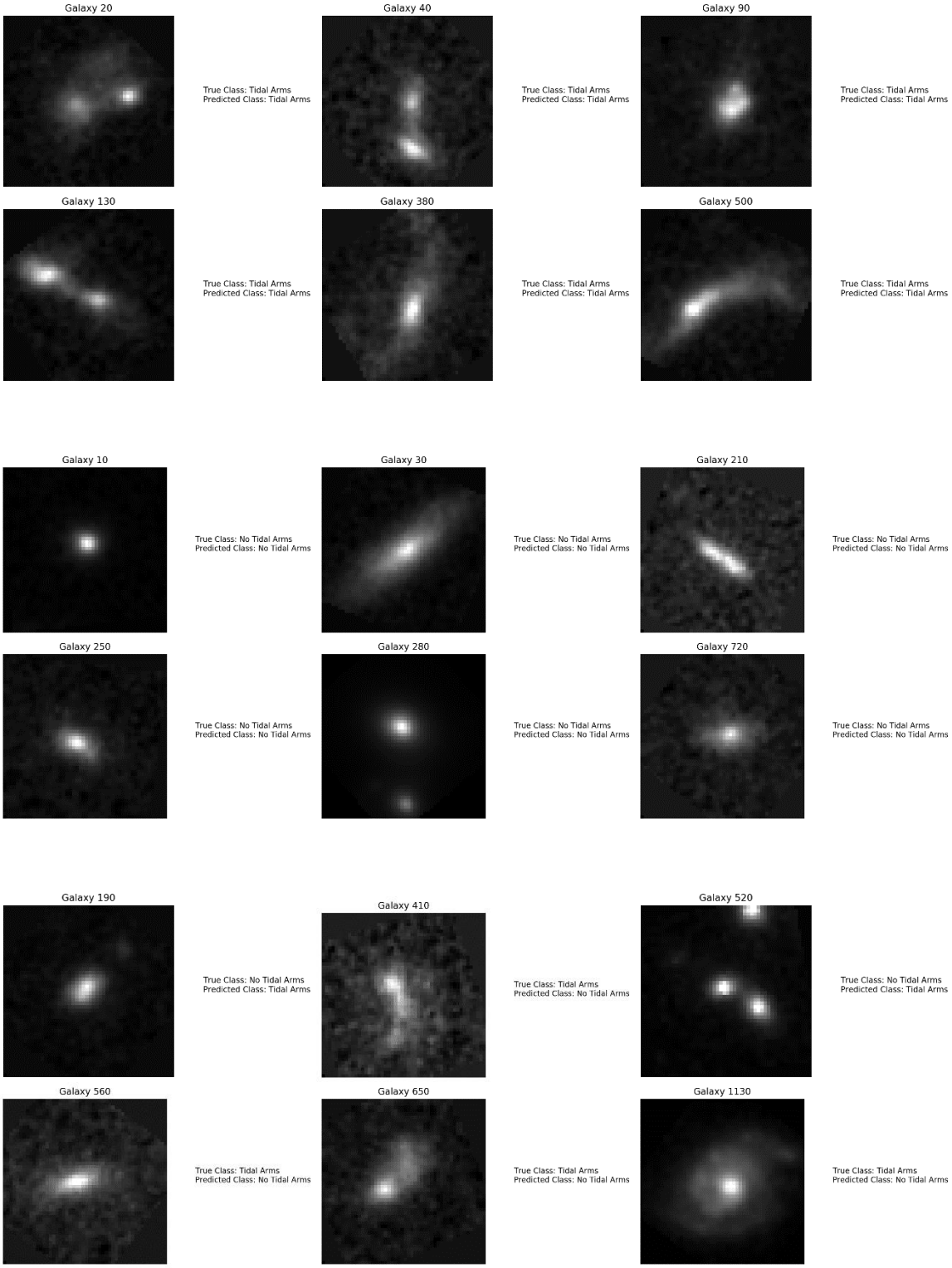


Figure 4.2.7: Tidal Arms Classifier Image Predictions. Example image predictions from the validation dataset for the tidal arms classifier: true positives (top), true negatives (middle), and misclassified images (bottom).

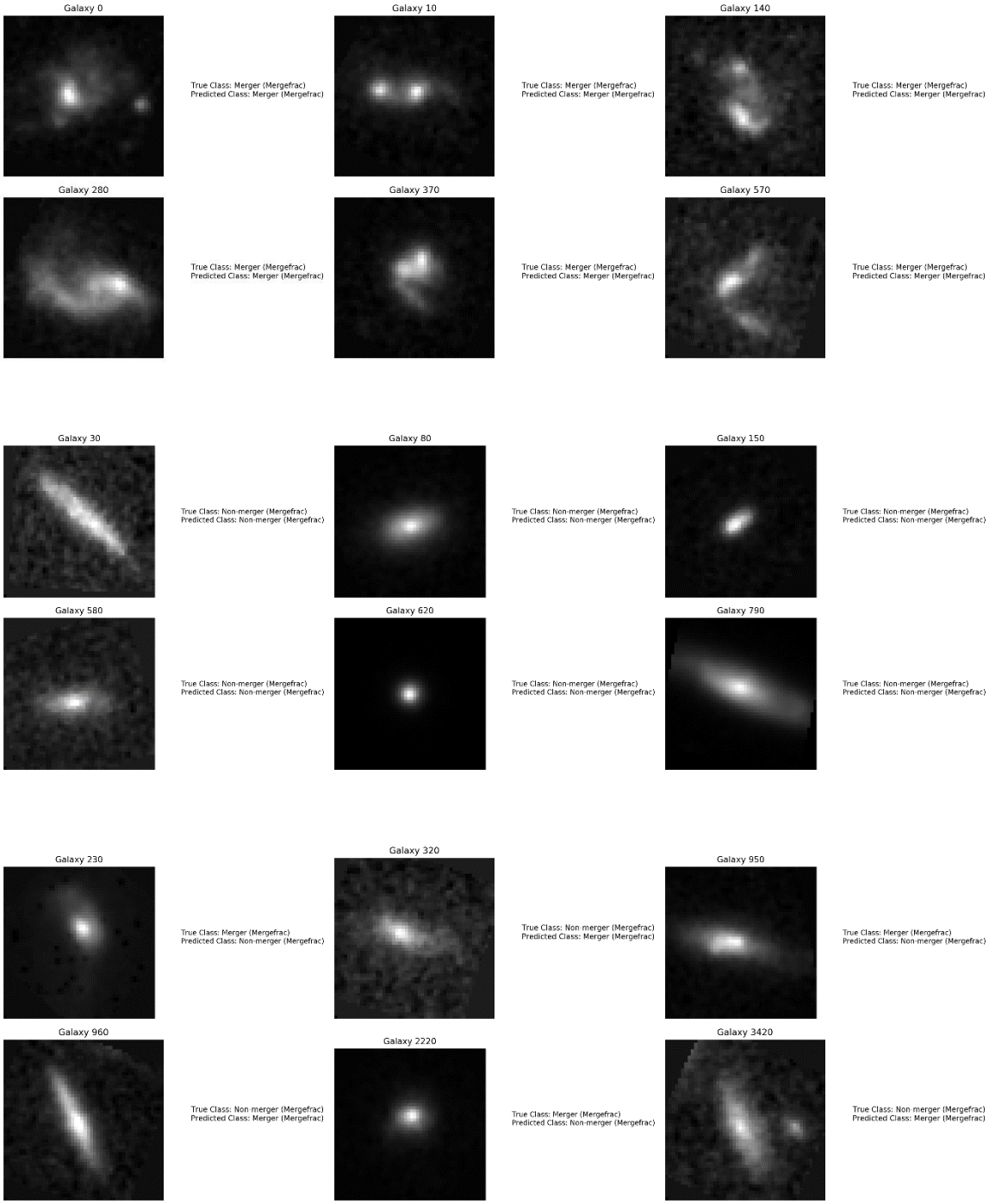


Figure 4.2.8: Merger Fraction Classifier Image Predictions. Example image predictions from the validation dataset for the merger fraction classifier: True positives (top), true negatives (middle), and misclassified images (bottom).

### 4.3 AGN and Super-massive Black Hole Identification

Once the computer vision models were trained, a new dataset was generated containing images of galaxies in the CANDELS fields that contain x-ray selected active galactic nuclei (AGN). AGN are thought to play a prominent role in the way a galaxy evolves and develops through its lifetime.[25,38,59] This dataset was passed through the pre-trained merger fraction computer vision classifier to predict which galaxies were mergers and which were non-mergers. The predicted classes were then cross-referenced with which galaxies in the set host AGN and which do not.

The AGN dataset consisted of 972 AGN-hosting galaxies and 2909 control images. Postage stamps were created from the same filters used for the other classifier datasets: F105W, F125W, F140W, and F160W. Data augmentation was not applied to this dataset since the model was pre-trained using the merger fraction dataset described in Section 3.3. After generating cutouts from the four WFC3 filters, the final dataset contained 3218 AGN-hosting galaxy images and 8548 control galaxy images, for a total of 11766 images.

The pre-trained merger fraction computer vision classifier then predicted the classes – "merger" or "non-merger" – for each galaxy in the new data sample. The class predictions were then cross-referenced with the list of galaxies that did or did not host AGN. Unfortunately, the results of this study showed that the predicted mergers and predicted non-mergers contained the same distribution of AGN-hosting and control galaxies [Figure *4.3.1*]. This indicates that there is no trend between galaxies that host AGN and merger events. However, considering the limitations imposed on this study- namely the sample size used for training and the uncertainty of the human classifications – it would be worth investigating this hypothesis more thoroughly in the future.

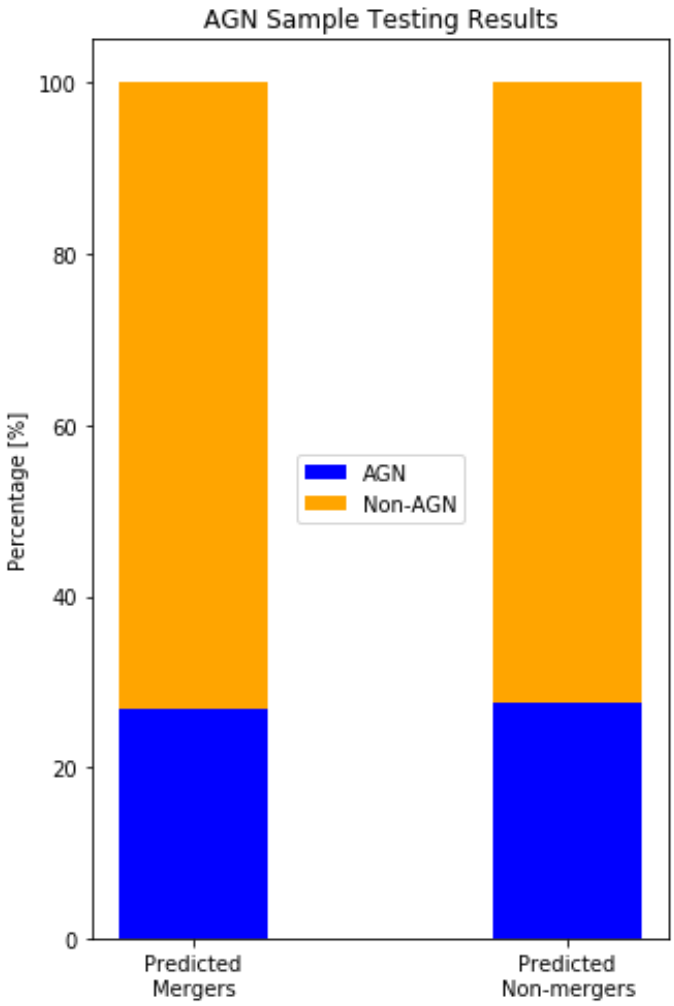


Figure 4.3.1: AGN distribution among predicted mergers and non-mergers.

## 4.4 Final Analysis

The random forest and computer vision classifiers performed at near parity. Among the classifiers, the two merger fraction classifiers attained the highest accuracies rate during testing, as well as the highest precisions, recalls, and F1 scores. This is likely due to the merger fraction datasets that these classifiers were trained on. By requiring that the positive class exhibited at least two of the three merger indicators – as classified by humans – they were the most disturbed systems and represented the "best examples" of merger events in the CANDELS fields. However, there are tradeoffs between using a random forest algorithm against a CNN and vice-versa.

### 4.4.1 Training Time

The random forest classifier has the benefit of being the faster algorithm. The random forest classifiers used in this work could be trained and validated within a manner of seconds, due to the relatively simple calculations that the algorithm needs to complete. The initial training of the RF classifier was performed on a laptop without a GPU, meaning the learning was handled by the 2.3 GHz, 4-core Intel I5 processor. The CNN, on the other hand, took several minutes – or, in the case of the asymmetry classifier, hours - to train. The training time increased noticeably as the dataset size increased. The asymmetry classifier, whose post-augmentation training dataset included 200,000 images – was trained overnight.

The time to train the network could be reduced by implementing higher quality GPUs. In this work, an Nvidia GeForce GTX 960 was used to train the CNN, but in the years since that line of GPUs was released, huge strides have been made to create much faster cards. While the GTX 960 has 2 GB of onboard memory, the newest line of Nvidia GPU's – the GeForce RTX 3080 series - has 12 GB of onboard memory, making it much faster than the 900-series cards. Multiple

cards could also be implemented to reduce the training time even further, which is what large tech companies such as Google and Facebook use for their neural networks.

The relative simplicity of calculations made in the random forest algorithm allow it to be run on machines that do not contain GPUs. As such, research using random forest classifiers do not rely on expensive computers or top-of-the-line GPUs. A random forest classifier would be useful for a researcher that does not have access to such a machine.

### 4.4.2 Data Generation

The data that each algorithm used for training and validation was vastly different. On one hand, the random forest classifier relied on *measured parameters ("features")* of the galaxies it was learning to classify. Researchers were required to measure these features for each galaxy included in the dataset, requiring a great deal of overhead in generating the dataset.

The computer vision classifier, on the other hand, only required images of the galaxies that are used for training and validation. Due to the nature of CNNs, the original data sample was not sufficient for training, and data augmentation needed to be used to artificially increase the sample size. However, the keras.preprocessing packages for Python make data augmentation a simple process to implement.

Because the data for the random forest classifier only consists of values of measured features, very little hard drive memory is required to store the data. The image data used by the computer vision classifiers takes up significantly more storage space. However, modern computer hard drives have enough storage space that data storage was never an issue.

The most significant issue with data generation in this study was the classification of each galaxy. The "true" classes were determined by humans, as discussed in Section 1.5.2. Because

humans are fallible, not all human-determined classes will be *true* classes. Part of the inspiration for implementing the merger fraction was so human error could be somewhat mitigated. The random forest and computer vision classifiers, then, cannot exceed human-level performance. It would be far better to use data where the classes were truly known, which will be discussed in more detail in Section 4.5.

## 4.5 Future Work

JWST is expected to launch in October 2021, and the first data is expected to return by Summer 2022. JWST will be capable of imaging galaxies in the infrared spectrum, allowing researchers to capture images of galaxies that are much fainter and much farther away from Earth. A random forest classifier or CNN like the ones trained in this study could be used to sift through the vast amount of data and predict which systems are mergers and which are non-mergers. However, work - discussed below - could be done before the first data is returned.

### 4.5.1 K-Nearest Neighbor Algorithm

While the random forest classifier and CNN were effective in recognizing galaxy merger events, they only represent two machine learning techniques. Another simple algorithm that could be implemented in a similar study is the k-nearest neighbor (KNN) algorithm. In this algorithm, the galactic features used in the random forest classifier are plotted in n-dimensional space. To predict the class of a validation data point, the galaxy's features are plotted in the same n-dimensional space as the training data. The algorithm then determines the k-nearest datapoints, where k is a user-defined integer and is often equal to the square root of the number of training datapoints. The validation datapoint is predicted to belong to the same class to which most of the k-nearest data points belong. The clustering that was seen in this random forest study [Figure *4.1.3*] indicate that a KNN study might be an effective classifier.

### 4.5.2 Simulations and Synthetic Data

As mentioned in Section 4.4.2, the biggest challenge in this study was the fact that the true classes of each galaxy were determined by humans. It would be much better if the training and validation data passed to the classifiers had truly known classes. The most straightforward solution to this problem would be to use cosmological simulation data. Cosmological simulations like the Simulating Multiscale Astrophysics to Understand Galaxies (SMAUG) Project[60] are designed to understand the earliest stages of our Universe and the galaxies that constitute it. SMAUG researchers will be able to identify individual galaxies, as well as flag merger events, and extract useful features from them (redshift, Hubble magnitude, galactic half-light radius, etc.). Additionally, the simulation will be able to produce HST-like images that will be comparable to WFC3 CANDELS images. Due to the nature of cosmological simulations, images of the same galaxy can be captured from different perspectives, effectively increasing the sample size even further.

A more novel approach to generating synthetic data would be to implement a computer vision technique called a generative adversarial network (GAN). A GAN consists of two major components: a generator and a classifier. The generator's job is to create synthetic images that can "trick" the classifier. The classifier is passed a dataset containing both real images and the synthetic images created by the generator and is tasked with determining which images are real and which are fake. The classifier results are sent back to the generator, which uses them to create new synthetic images that better reflect real data. This process goes back and forth between the generator and classifier until the generator has learned to create synthetic images that are indistinguishable from real ones.

Using a cosmological simulation or GAN, new synthetic images of mergers and non-mergers could be generated. It has been shown in the past that machine learning and computer vision algorithms can be trained on synthetic data and tested on real data with relatively high accuracy.[53,61] A random forest classifier or CNN trained on representative synthetic data would be training on data where the true class is known, and could potentially outperform a human classifier.

## REFERENCES


1. Abraham, R.G., et al (1996). Galaxy Morphology to I = 25 mag in the Hubble Deep Field. arXiv: 9602044v1 [astro-ph]

2. Alger, M.J., Banfield, J.K., Ong, C.S., Rudnick, L., Wong, O.I., Wolf, C., Andernach, H., Norris, R.P., & Shabala, S.S. (2018) - Radio Galaxy Zoo- Machine Learning for Radio Source Host Galaxy Cross-Identification. arXiv: 1805.05540v1 [astro-ph.IM]

3. Anderson, T.W. & Darling, D.A. (1954). A Test of Goodness of Fit. Journal of American Statistical Association, 49:268, 765-769, DOI: 10.1080/01621459.1954.10501232

4. The Astropy Collaboration, A. M. Price-Whelan, B. M. Sipőcz, H. M. Günther, P. L. Lim, S. M. Crawford, S. Conseil, D. L. Shupe, M. W. Craig, N. Dencheva, A. Ginsburg, J. T. VanderPlas, L. D. Bradley, D. Pérez-Suárez, M. de Val-Borro, T. L. Aldcroft, K. L. Cruz, T. P. Robitaille, E. J. Tollerud, C. Ardelean, T. Babej, M. Bachetti, A. V. Bakanov, S. P. Bamford, G. Barentsen, P. Barmby, A. Baumbach, K. L. Berry, F. Biscani, M. Boquien, K. A. Bostroem, L. G. Bouma, G. B. Brammer, E. M. Bray, H. Breytenbach, H. Buddelmeijer, D. J. Burke, G. Calderone, J. L. Cano Rodríguez, M. Cara, J. V. M. Cardoso, S. Cheedella, Y. Copin, D. Crichton, D. DÁvella, C. Deil, É. Depagne, J. P. Dietrich, A. Donath, M. Droettboom, N. Earl, T. Erben, S. Fabbro, L. A. Ferreira, T. Finethy, R. T. Fox, L. H. Garrison, S. L. J. Gibbons, D. A. Goldstein, R. Gommers, J. P. Greco, P. Greenfield, A. M. Groener, F. Grollier, A. Hagen, P. Hirst, D. Homeier, A. J. Horton, G. Hosseinzadeh, L. Hu, J. S. Hunkeler, Ž. Ivezić, A. Jain, T. Jenness, G. Kanarek, S. Kendrew, N. S. Kern, W. E. Kerzendorf, A. Khvalko, J. King, D. Kirkby, A. M. Kulkarni, A. Kumar, A. Lee, D. Lenz, S. P. Littlefair, Z. Ma, D. M. Macleod, M. Mastropietro, C. McCully, S. Montagnac, B. M. Morris, M. Mueller, S. J. Mumford, D. Muna, N. A. Murphy, S. Nelson, G. H. Nguyen, J. P. Ninan, M. Nöthe et al. (2018). The Astropy Project: Building an inclusive, open-science project and status of the v2.0 core packages. arXiv:1801.02634

5. Banerji, M., et al (2010). Galaxy Zoo - Reproducing Galaxy Morphologies Via Machine Learning. The Royal Astronomical Society, 406, 342-353

6. Bergh, S. (2003). History of the Local Group. arXiv: astro-ph/0305042v2

7. Blanton, M. et al (2017). Sloan Digital Sky Survey IV- Mapping the Milky Way, Nearby Galaxies, and the Distant Universe. The Astronomical Journal, 154:28 (35pp). doi: 10.3847/1538-3881/aa7567

8. Block, Adam. Messier 66, an asymmetric spiral galaxy. Anne's Astronomy News. http://annesastronomynews.com/photo-gallery-ii/galaxies-clusters/processed-with-maxim-dl-2/.

9. Buta, R., Mitra, S., de Vaucouleurs, G., and Corwin, Jr., H.G. (1994). Mean Morphological Types of Bright Galaxies. The Astronomical Journal, 107, 1

10. Calleja, J. & Fuentes, O. (2003). Machine Learning and Image Analysis for Morphological Galaxy Classification. Royal Astronomical Society, 349, 87-93

11. CBS News. “Facebook's DeepFace Shows Serious Facial Recognition Skills.” CBS News, CBS Interactive, 19 Mar. 2014, https://www.cbsnews.com/news/facebooks-deepface-shows-serious-facial-recognition-skills/

12. Ceverino, D., Dekel, A., Tweed, D., & Primack, J. (2015). Early Formation of Massive, Compact, Spheroidal Galaxies with Classical Profiles by Violent Disc Instability or Mergers. Royal Astronomical Society, 447, 3291-3310

13. Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke & Travis E. Oliphant. Array programming with NumPy, Nature, 585, 357–362 (2020), DOI:10.1038/s41586-020-2649-2

14. Cheng (2019) - Optimising Automatic Morphological Classification of Galaxies with Machine Learning and Deep Learning using Dark Energy Survey Imaging. arXiv: 1908.03610v1 [astro-ph.GA]

15. Choi, E., Ostriker, J., Naab, T., Somerville, R., Hirschmann, M., Nuñez, A., Hu, C., & Oser, L. (2017). Physics of Galactic Metals- Evolutionary Effects due to Production, Distribution, Feedback, and Interaction with Black Holes. The Astrophysical Journal, 844:31, (15pp). doi: 10.3847/1538-4357/aa7849

16. Choi, E., Somerville, R., Ostriker, J., Naab, T., & Hirschmann, M. (2018). The Role of Black Hole Feedback on Size and Structural Evolution in Massive Galaxies. The Astrophysical Journal, 866:91 (12pp). doi: 10.3847/1538-4357/aae076

17. “Cosmic Origins (COR).” NASA, NASA, https://cor.gsfc.nasa.gov/

18. Davis, T. & Young L.M. (2019). Gas Accretion as Fuel for Residual Star Formation in Galaxy Zoo Elliptical Galaxies. arXiv: 1909.01230v1 [astro-ph.GA]

19. de Vaucouleurs, G. (1948). Research of Extragalactic Nebulae. Annales d’Astrophysique, 11, 247

20. de Vaucouleurs (1977). Qualitative and Quantitative Classifications of Galaxies. Proceedings of a Conference at Yale University, May 19-21, 1977. New Haven: Yale University Observatory, 1977, p43

21. Dieleman, S., Willet, K.W., & Dambre, J (2015). Rotation-invariant convolutional neural networks for galaxy morphology prediction. arXiv: 1503.07077v [Astro-ph.IM]

22. Dressler, A. (1980). Galaxy Morphology in Rich Clusters- Implications for the Formation and Evolution of Galaxies. The Astrophysical Journal, 236:351-365

23. Duffau, S., Zinn, R., Vivas, A.K., Carraro, G., Méndez, R.A., Winnick, R., & Gallart, C. (2005). Spectroscopy of QUEST RR Lyrae Variables - The New Virgo Stellar Stream. arXiv: astro-ph/0510589v2

24. *Euclid: Mapping the Geometry of the Dark Universe*. Definition Study Report, European Space Agency/SRE, 2011

25. Fabian, A.C. (2012). Observational Evidence of AGN Feedback. arXiv: 1204.4114v1 [astro-ph.CO]

26. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)

27. Freedman, W.L., et al (2000). Final Results from the Hubble Space Telescope Key Project to Measure the Hubble Constant. arXiv:astro-ph/0012376v1

28. Freedman, W.L., et al (2019). *The Carnegie-Chicago Hubble Program*. VIII. An Independent Determination of the Hubble Constant Based on the Tip of the Red Giant Branch. arXiv:1907.05922v1 [astro-ph.CO]

29. Géron, A. *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow*. O'Reilly Media, Inc., 2019.

30. Grogin, N., et al (2011). CANDELS - The Cosmic Assembly Near-Infrared Deep Extragalactic Legacy Survey. The Astrophysical Journal Supplement Series, 197:35 (39 pp). doi: 10.1088/0049/197/2/35

31. Hart, R.E., Bamford, S.P., Keel, W.C., Kruk, S.J., Masters, K.L., Simmons, B.D., & Smethurst, R.J. (2018). Galaxy Zoo- Constraining the Origin Of Spiral Arms. arXiv: 1805.01782v1 [astro.ph-GA]

32. Hausen, R., & Robertson, B. (2019). Morpheus: A Deep Learning Framework For Pixel-Level Analysis of Astronomical Image Data. arXiv: 1906.11248v1 [astro-ph.GA]

33. Ho, T. (1995). Random Decision Forests. AT&T Bell Laboratories

34. Ho, T. (1998). The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20, 8

35. Hubble, E. (1926). Extra-Galactic Nebulae. Astrophysical Journal, 64, 321

36. Hubble, E. (1929). A Relation Between Distance and Radial Velocity Among Extra-Galactic Nebulae. Astrophysical Journal, 15, 168

37. Huertas-Company, M., et al (2015). A Catalog of Visual-Like Morphologies in the 5 Candels Fields Using Deep Learning. The Astrophysical Journal Supplement Series, 221:8 (23pp). doi: 10.1088/0067-0049/221/1/8

38. Ishibashi, W. & Fabian, A.C. (2012). AGN Feedback and Triggering of Star Formation in Galaxies. arXiv: 1209.1480v1 [astro-ph.GA]

39. Ivezic, Ž., et al (2019). LSST- From Science Drivers to Reference Design and Anticipated Data Products. The Astrophysical Journal, 873:111 (44pp)

40. JWST User Documentation 2016- Baltimore, MD. Space Telescope Science Institute 2019, November 19, https://jwst-docs.stsci.edu

41. Kartaltepe, J., et al (2015) - CANDELS Visual Classifications/Scheme, Data Release, and First Results. The Astrophysical Journal Supplement Series, 221:11 (17pp). doi: 10.1088/0067-0049/221/1/11

42. Kennicutt Jr., R.C. (1998). Star Formation in Galaxies Along the Hubble Sequence. arXiv: astro-ph/9807187v1

43. Kozmanyan, A., et al (2019). Deriving the Hubble constant using Planck and XMM-Newton observations of galaxy clusters. Astronomy & Astrophysics, Vol. 621, A34.

44. Kravtsov, A., & Borgani, S. (2012). Formation of Galaxy Clusters. arXiv: 1205.5556v1 [astro-ph.CO]

45. Kuminski, E., George, J., Wallin, J., & Shamir, L. (2014). Combining Human and Machine Learning for Morphological Analysis of Galaxy Images. The Astronomical Society of the Pacific, 126: 959-967

46. Lintott, C., et al (2010). Galaxy Zoo 1: Data Release of Morphological Classifications for Nearly 900,000 Galaxies. arXiv: 1007.3265v4 [astro-ph.GA]

47. Lotz, Jennifer M., et al (2004). A New Non-Parametric Approach to Galaxy Morphological Classification. arXiv: astro-ph/0311352v2

48. Martin, N.F., Ibata, R.A., Bellazzini, M., Irwin, M.J., Lewis, G.F., & Dehnen, W. (2003). A Dwarf Galaxy Remnant in Canis Major - The Fossil of an In-Plane Accretion onto the Milky Way. arXiv: astro-ph/0311010v1

49. Masters, K., et al (2019). Twelve Years of Galaxy Zoo. arXiv: 1910.08177v1 [astro-ph.GA]

50. McCulloch, W. & Pitts W. A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics, Vol. 5, 1943

51. NASA. (2012, February 6). *WFC3 - Technology - Filters*. NASA. https://wfc3.gsfc.nasa.gov/tech/filters-ir.html.

52. NASA, H. Ford (JHU), G. Illingworth (USCS/LO), M. Clampin (STScI), G. Hartig (STScI), the ACS Science Team, and ESA. (2012, November 1). NASA. https://apod.nasa.gov/apod/ap100926.html.

53. Pearson, W.J., Wang, L., Trayford, J.W., Petrillo, C.E., & van der Tak, F.F.S. (2019). Identifying Galaxy Mergers in Observations and Simulations with Deep Learning. arXiv: 1902.10626v3 [astro-ph.GA].

54. Pogge, R. "Lecture 15: The Main Sequence." Astronomy, Ohio State University, 1 Aug. 2006, http://www.astronomy.ohio-state.edu/~pogge/Ast162/Unit2/mainseq.html.

55. P.W.D. Charles, Project Title, (2013), GitHub repository, https://github.com/charlespwd/project-titlea

56. Samuel, A.L. (1959). Some Studies in Machine Learning Using the Game of Checkers. IBM Journal, 3, 3

57. Schweizer, F. (2005). Merger-Induced Starbursts. arXiv: astro-ph/0502111v1

58. Sérsic (1963). Influence of the Atmospheric and Instrumental Dispersion on the Brightness Distribution in a Galaxy. Boletin de la Asociacion Argentina de Astronomia, Vol. 6, p.41

59. Shin, J., Woo, J., Chung, A., Baek, J., Cho, K., Kang, D., & Bae, J. (2019). Positive and Negative Feedback of AGN Outflows in NGC 5728. arXiv: 1907.00982v1 [astro-ph, GA]

60. "Simulating Multiscale Astrophysics to Understand Galaxies (SMAUG)." Simons Foundation, https://www.simonsfoundation.org/flatiron/center-for-computational-astrophysics/galaxy-formation/smaug/

61. Snyder, G.F., Rogridguez-Gomez, V., Lotz, J.M., Torrey, P., Quirk, A.C.N., Hernquist, L., Vogelsberger, M., & Freeman, P.E. (2019). Automated Distant Galaxy Merger Classifications from Space Telescope Images using the Illustris Simulation. arXiv: 1809.02136v2 [astro-ph.GA].

62. Strateva, S., et al (2001). Color Separation of Galaxy Types in the Sloan Digital Sky Survey Imaging Data. arXiv: astro-ph/0107201v1

63. Tully, R.B. (1982). The Local Supercluster. The Astrophysical Journal, 257: 389-422

64. Weinberger, Kilian (2020, January 19). CORNELL CS4780 "Machine Learning for Intelligent Systems." YouTube. youtube.com/playlist?list=PLl8OlHZGYOQ7bkVbuRthEsaLr7bONzbXS.

65. Wikimedia Foundation. (2021, June 13). Sersic profile. Wikipedia. https://en.wikipedia.org/wiki/Sersic_profile#/media/File:S%C3%A9rsic_models_with_different_(n)_indices.jpg.

66. Wu, C., et al (2018). Radio Galaxy Zoo- ClaRAN — A Deep Learning Classifier For Radio Morphologies. arXiv: 1805.12008v2 [astro-ph.IM]

67. "Zooniverse." Zooniverse.org, https://www.zooniverse.org/about

## APPENDIX A: RANDOM FOREST CLASSIFIER PYTHON CODE

```
# Classifies whether a galaxy is a merger or non-merger
# Import modules
from astropy import units as u
from astropy.io import fits
from astropy.nddata import Cutout2D
from astropy.table import Table
from astropy.visualization import imshow_norm, SqrtStretch, LogStretch
from astropy.wcs import WCS
import csv
from datetime import datetime
from matplotlib import cm
import matplotlib as mpl
from matplotlib.colors import LogNorm
import matplotlib.pyplot as plt
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits import mplot3d
import numpy as np
import os
import pandas as pd
from pathlib import Path
from pprint import pprint
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split

# Define Paths
homepath = Path(str(os.path.normpath(os.getcwd() + os.sep + os.pardir)))
mos_path = homepath / 'Mosaics'
params_path = homepath / 'StructuralParameters'
data_path = homepath / 'RF Classifiers' / 'Data'
results_path = homepath / 'RF Classifiers' / 'Results' / 'Mergefrac'

# Prepare Training and Testing Data
# Read CANDELS .fits tables into Python
gds = fits.open(params_path / 'CANDELS.GDS.1018.wCAS_VC.fits')
uds = fits.open(params_path / 'CANDELS.UDS.1018.wCAS_VC.fits')
gdn = fits.open(params_path / 'CANDELS.GDN.1018.wCAS_VC.fits')
cos = fits.open(params_path / 'CANDELS.COS.1018.wCAS_VC.fits')
egs = fits.open(params_path / 'CANDELS.EGS.1018.wCAS_VC.fits')

# Extract data array from CANDELS .fits tables
gds_data = Table(gds[1].data)
uds_data = Table(uds[1].data)
```

```
gdn_data = Table(gdn[1].data)
cos_data = Table(cos[1].data)
egs_data = Table(egs[1].data)
gds_data['Field'] = 1  # 1 = GOODS-S
uds_data['Field'] = 2  # 2 = UDS
gdn_data['Field'] = 3  # 3 = GOODS-N
cos_data['Field'] = 4  # 4 = COSMOS
egs_data['Field'] = 5  # 5 = EGS

# Create numpy arrays of features to be trained on
features_gds = np.array([gds_data['ID'], gds_data['HMAG'], gds_data['ZBEST'],
gds_data['VC_F_MERGER'], gds_data['GINI'],
                gds_data['M20'], gds_data['CONC'], gds_data['ASYM'], gds_data['MASS'],
gds_data['SFR'],
                gds_data['RKPC_GALFIT'], gds_data['RA'], gds_data['DEC'],
gds_data['VC_F_ASYM'],
                gds_data['VC_F_TIDAL'], gds_data['X_IMAGE'], gds_data['Y_IMAGE'],
gds_data['Field']])
features_uds = np.array([uds_data['ID'], uds_data['HMAG'], uds_data['ZBEST'],
uds_data['VC_F_MERGER'], uds_data['GINI'],
                uds_data['M20'], uds_data['CONC'], uds_data['ASYM'], uds_data['MASS'],
uds_data['SFR'],
                uds_data['RKPC_GALFIT'], uds_data['RA'], uds_data['DEC'],
uds_data['VC_F_ASYM'],
                uds_data['VC_F_TIDAL'], uds_data['X_IMAGE'], uds_data['Y_IMAGE'],
uds_data['Field']])
features_gdn = np.array([gdn_data['ID'], gdn_data['HMAG'], gdn_data['ZBEST'],
gdn_data['VC_F_MERGER'], gdn_data['GINI'],
                gdn_data['M20'], gdn_data['CONC'], gdn_data['ASYM'], gdn_data['MASS'],
gdn_data['SFR'],
                gdn_data['RKPC_GALFIT'], gdn_data['RA'], gdn_data['DEC'],
gdn_data['VC_F_ASYM'],
                gdn_data['VC_F_TIDAL'], gdn_data['X_IMAGE'], gdn_data['Y_IMAGE'],
gdn_data['Field']])
features_cos = np.array([cos_data['ID'], cos_data['HMAG'], cos_data['ZBEST'],
cos_data['VC_F_MERGER'], cos_data['GINI'],
                cos_data['M20'], cos_data['CONC'], cos_data['ASYM'], cos_data['MASS'],
cos_data['SFR'],
                cos_data['RKPC_GALFIT'], cos_data['RA'], cos_data['DEC'],
cos_data['VC_F_ASYM'],
                cos_data['VC_F_TIDAL'], cos_data['X_IMAGE'], cos_data['Y_IMAGE'],
cos_data['Field']])
features_egs = np.array([egs_data['ID'], egs_data['HMAG'], egs_data['ZBEST'],
egs_data['VC_F_MERGER'], egs_data['GINI'],
                egs_data['M20'], egs_data['CONC'], egs_data['ASYM'], egs_data['MASS'],
egs_data['SFR'],
```

```
                egs_data['RKPC_GALFIT'], egs_data['RA'], egs_data['DEC'],
egs_data['VC_F_ASYM'],
                egs_data['VC_F_TIDAL'], egs_data['X_IMAGE'], egs_data['Y_IMAGE'],
egs_data['Field']])

# Concatenate to create one features array
features = np.concatenate((features_gds, features_uds, features_cos, features_egs), axis=1)
features = np.transpose(features)
print('Total number of galaxies before cuts: ', len(features))

# Create pandas dataframe, labelling columns appropriately
features = pd.DataFrame(features, columns = ['ID','HMAG', 'ZBEST', 'VC_F_MERGER', 'GINI',
'M20', 'CONC', 'ASYM', 'MASS', 'SFR',
                                    'rkpc', 'RA', 'Dec', 'VC_F_ASYM', 'VC_F_TIDAL', 'X_IMAGE',
'Y_IMAGE', 'Field'])
features['merge_frac'] =
features['VC_F_MERGER']+features['VC_F_ASYM']+features['VC_F_TIDAL']

# Set upper/lower limits for Visual Magnitude (Hmag), Redshift (Z), Mass, Star Formation Rate
(SFR), and Galactic Radius (rkpc)
hmag_lower = 0
hmag_upper = 24.5
z_lower = 0.5
z_upper = 2.5
mass = 9
sfr = 0
rkpc = 0
merger_limit = 2.0

# Make data cuts to HMAG, Z, Mass, SFR, and rkpc
mergers = features[features.merge_frac >= merger_limit]
print('Number of mergers before cuts: ', len(mergers))
mergers = mergers[(mergers.HMAG <= hmag_upper) & (mergers.HMAG >= hmag_lower)]
mergers = mergers[(mergers.ZBEST > z_lower) & (mergers.ZBEST <= z_upper)]
mergers = mergers[mergers.MASS >= mass]
mergers = mergers[mergers.SFR >= sfr]
mergers = mergers[mergers.rkpc >= rkpc]

# Remove rows with no values for Gini, M20, Concentration, and Asymmetry
mergers = mergers[mergers.GINI != -99.00]
mergers = mergers[mergers.M20 != -99.00]
mergers = mergers[mergers.CONC != -99.00]
mergers = mergers[mergers.ASYM != -99.00]
mergers['Merger'] = 1
n_mergers = len(mergers)
print('Number of mergers after cuts: ', str(n_mergers))
```

```
# Non-mergers
n_nonmergers = n_mergers
n_nonmergers = round(n_nonmergers)

# Make feature cuts
nonmergers = features[features.merge_frac == 0]
print('Number of nonmergers before cuts: ', len(nonmergers))
nonmergers = nonmergers[(nonmergers.HMAG <= 24.5) & (nonmergers.HMAG >= 0)]
nonmergers = nonmergers[(nonmergers.ZBEST >= 0.5) & (nonmergers.ZBEST <= 2.5)]
nonmergers = nonmergers[nonmergers.MASS >= 9]
nonmergers = nonmergers[nonmergers.SFR >= 0]
nonmergers = nonmergers[nonmergers.rkpc >= 0]
nonmergers = nonmergers[nonmergers.GINI != -99.00]
nonmergers = nonmergers[nonmergers.M20 != -99.00]
nonmergers = nonmergers[nonmergers.CONC != -99.00]
nonmergers = nonmergers[nonmergers.ASYM != -99.00]
nonmergers = nonmergers.sample(frac=1)
nonmergers = nonmergers.iloc[:n_nonmergers]
nonmergers['Merger'] = 0
print('Number of nonmergers after cuts: ', len(nonmergers))

# Shuffle merger dataframe
mergers = mergers.sample(frac=1)

# Concatenate mergers and nonmergers darafraems
data = pd.concat([mergers, nonmergers])
print('Number of data points: ', (n_mergers+n_nonmergers))

# Prepare training/testing data arrays
# Shuffle the data randomly
data = data.sample(frac=1)

# Drop rows with NaN values from data
data.dropna()

# Number of training data points (75% of all data points are training points)
n_training = len(data)*0.75
# Number of training data points
n_training = round(n_training)
# Number of testing data points
n_testing = len(data) - n_training
# Create numpy array containing data features
data_array = np.array(data, float)
# Create training data array
training_data = data.iloc[:n_training]
```

```
# Drop unused features from training data array
training_data = training_data.drop(['HMAG', 'ZBEST', 'VC_F_MERGER'], axis=1)
# Create testing data array
testing_data = data.iloc[n_training:]
# Drop unused features from testing data array
testing_data = testing_data.drop(['HMAG', 'ZBEST', 'VC_F_MERGER'], axis=1)

training_features = training_data.columns[1:8]
print('Training on features: ', str(training_features)[7:-18])
# Create directory for Results
X = str(datetime.now())
X = X.replace(' ', '_')
X = X.replace(':','_')
X = X[:-7]
path = results_path / str(X)
print(path)

try:
    os.mkdir(path)
except OSError:
    print('Creation of directory %s failed' % path)
else:
    print('Creation of directory %s successful!' % path)

# Create .txt file containing data size, magnitude/redshift/etc cuts
filename = path / 'Dataset.txt'
file = open(filename, 'w')
file.write('---- Data Set ----\n\n')
file.write('Fields: GOODS-S, COSMOS, UDS, EGS\n')
file.write('Total number of galaxies before cuts: '+str(len(features))+'\n')
file.write('Number of mergers before cuts: '+str(len(mergers))+'\n')
file.write('Number of nonmergers after cuts: '+str(len(nonmergers))+'\n\n')
file.write('Total number of data points: '+str(len(mergers)+len(nonmergers))+'\n\n')
file.write('Training data size: '+str(len(training_data))+'\n')
file.write('Number of training nonmergers: '+str(len(training_data[training_data.Merger ==
0]))+'\n')
file.write('Number of training mergers: '+str(len(training_data[training_data.Merger ==
1]))+'\n\n')
file.write('Testing data size: '+str(len(testing_data))+'\n')
file.write('Number of testing nonmergers: '+str(len(testing_data[testing_data.Merger == 0]))+'\n')
file.write('Number of testing mergers: '+str(len(testing_data[testing_data.Merger == 1]))+'\n\n')
file.write('---- Data Criteria ----\n\n')
file.write('Mergers classified when merge_frac >= '+str(merger_limit)+'\n')
file.write('Hmag: '+str(hmag_lower)+'< Hmag < '+str(hmag_upper)+'\n')
file.write('Redshift: '+str(z_lower)+'< z < '+str(z_upper)+'\n')
file.write('Mass: m > '+str(mass)+'\n')
```

```
file.write('Star Formation Rate: sfr > '+str(sfr)+'\n')
file.write('Galactic half-light radius: rkpc > '+str(rkpc)+'\n')
file.write('Gini, M20, Asymmetry, Concentration != -99 or NaN')
file.close()

# Prepare classifier for training and testing
# Define General Random Forest Classifier
clf = RandomForestClassifier(random_state=43)

# Prepare grid of hyperparameters to train on
# Number of trees in the forest
n_estimators = [int(x) for x in np.linspace(start=100, stop=200, num=11)]
# Maximum number of features considered at each tree node
max_features = [2]
# Maximum levels in each decision tree
max_depth = [None]
# Minimum number of data points considered at each node before node is split
min_samples_split = [2]
# Minimum number of data points allowed in each leaf node
min_samples_leaf = [4]
# Method for sampling (replacement or no replacement)
bootstrap = [True, False]
# Create and print parameter grid for randomized cross-validation
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print('Hyperparameter candidates:')
print()
pprint(random_grid)
print()
# Define Randomized Search Cross-validation training function
# using RF Classifier as estimator and random_grid
# for parameters
# Define number of folds for cross-validation
cv = 5
rf_random = RandomizedSearchCV(estimator=clf, param_distributions=random_grid,
                               n_iter=11, cv=5, verbose=False, random_state=43)
print('Training and Determining Optimal Hyperparameters...\n')
rf_random.fit(training_data[training_features], training_data['Merger'])
print('Training complete!\n')
best_hyperparams = rf_random.best_params_
# Print optimal hyperparameters
print('Optimal Hyperparameters:\n')
```

```
pprint(best_hyperparams)
# Classifier testing data using optimal hyperparameters
# Define classifier using optimal hyperparameters
best_random = rf_random.best_estimator_
# Classify testing data
preds = best_random.predict(testing_data[training_features])
# Create new column in testing data for Predicted Classes
testing_data['Predicted Merger'] = preds
# Analyze testing results
# Create spreadsheet containing Training and Testing Data
training_data.to_csv(path / 'TrainingData.csv')
testing_data.to_csv(path / 'TestingData.csv')
# Create a confusion matrix and define elements of the matrix
confusion_matrix = pd.crosstab([testing_data['Merger']], preds, rownames=['Actual Merger'],
colnames=['Predicted Merger'])
TN = confusion_matrix.iloc[0,0]
TP = confusion_matrix.iloc[1,1]
FN = confusion_matrix.iloc[1,0]
FP = confusion_matrix.iloc[0,1]
# Print confusion matrix and number of TN, TP, FN, FP
print('---- Confusion Matrix ----')
print('')
print('0 = Non-merger\n1 = Merger\n')
print(confusion_matrix)
print('')
print('True Negatives: ', TN)
print('True Positives: ', TP)
print('False Negatives: ', FN)
print('False Positives: ', FP)
print('')
print('--------------------------')
print('')
# Calculate and print Accuracy, Precision, and Recall
Accuracy = ((TP+TN)/(TP+TN+FP+FN))*100
Recall = TP/(TP+FN)
Precision = TP/(TP+FP)
F1 = 2*(Precision*Recall)/(Precision+Recall)
print('Accuracy = ', str(Accuracy)[:7], '%')
print('Recall = ', str(Recall)[:6])
print('Precision = ', str(Precision)[:6])
print('F1 Score = ', str(F1)[:6])
print('')
print('--------------------------')
print('')
print('Successful Nonmergers Predictions: ', str(TN/(TN+FP)))
print('Successful Mergers Predictions: ', str(TP/(FN+TP)))
```

```
# Feature Importance
feature_importance = best_random.feature_importances_
print(feature_importance)
# Write txt file containing confusion matrix and statistics
filename = path / 'ConfusionMatrix.txt'
file = open(filename, 'w')
file.write('---- Confusion Matrix ----\n\n')
file.write('0 = Non-merger\n1 = Merger\n\n')
file.write(str(confusion_matrix))
file.write('\n\nTrue Negatives: '+str(TN))
file.write('\nTrue Positives: '+str(TP))
file.write('\nFalse Negatives: '+str(FN))
file.write('\nFalse Positives: '+str(FP)+'\n')
file.write('\nAccuracy: '+str(Accuracy)[:7]+'%')
file.write('\nRecall: '+str(Recall)[:6])
file.write('\nPrecision: '+str(Precision)[:6]+'\n')
file.write('\nF1 Score: '+str(F1)[:6]+'\n')
file.write('\nSuccessful Nonmerger Predictions: '+str(TN/(TN+FP))[:6]+'%')
file.write('\nSuccessful Merger Predictions: '+str(TP/(FN+TP))+'%\n\n')
file.write('---- Feature Importance ----\n\n')
file.write('GINI: '+str(feature_importance[0])+'\n')
file.write('M20: '+str(feature_importance[1])+'\n')
file.write('Concentration: '+str(feature_importance[2])+'\n')
file.write('Asymmetry: '+str(feature_importance[3])+'\n')
file.write('Mass: '+str(feature_importance[4])+'\n')
file.write('Star Formation Rate: '+str(feature_importance[5])+'\n')
file.write('Galactic Half-light Radius: '+str(feature_importance[6])+'\n')
file.close()
# Plot a confusion matrix
class_names = ['Non-merger\n(Mergefrac)', 'Merger\n(Mergefrac)']
disp = plot_confusion_matrix(best_random, testing_data[training_features],
testing_data['Merger'],
                             display_labels=class_names,
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title('Random Forest Classifier Confusion Matrix')
disp.plot(cmap=plt.cm.Blues)
plt.title('Random Forest Classifier Confusion Matrix')
plt.savefig(path / 'ConfusionMatrix.png')
# Plot ROC Curve
viz = plot_roc_curve(best_random, testing_data[training_features], testing_data['Merger'])
viz.plot()
plt.title('Merger Random Forest Classifier ROC Curve')
plt.savefig(path / 'ROC_Curve.png')
# Add Accuracy, Recall, and Precision to spreadsheet of run data
from datetime import datetime
```

```
current_datetime = datetime.now()
current_datetime = str(current_datetime)
currentdate = current_datetime[:10]
currenttime = current_datetime[11:19]
currenttime = currenttime.replace('/', ':')
print(current_datetime)
print(currentdate)
print(currenttime)
merger_acc = TN/(TN+FP)*100
nonmerger_acc = TP/(FN+TP)*100
print(merger_acc)
print(nonmerger_acc)
test = np.array(testing_data)
# Create empty array of color values for plotting
color = np.zeros((len(preds)), dtype=str)
predmerger = np.empty((0,18), float)
prednonmerger = np.empty((0,18), float)
for i in range(0,len(preds)):
    # If predicted class of Galaxy i is "Merger," then its plot color is Red
    if preds[i] == 1:
        color[i] = 'reds'
        # Add Galaxy i data to predmerger array
        predmerger = np.append(predmerger, np.array([test[i]]), axis=0)
    # If predicted class of Galaxy i is "Nonmerger," then its plot color is Blue
    if preds[i] == 0:
        color[i] = 'blue'
        # Add Galaxy i data to prednonmerger array
        prednonmerger = np.append(prednonmerger, np.array([test[i]]), axis=0)
# Create data arrays for merger and non-merger parameters for plotting
ID_merger, ID_nonmerger = np.array(predmerger[:,0]), np.array(prednonmerger[:,0])
gini_merger, gini_nonmerger = np.array(predmerger[:,1]), np.array(prednonmerger[:,1])
m20_merger, m20_nonmerger = np.array(predmerger[:,2]), np.array(prednonmerger[:,2])
asym_merger, asym_nonmerger = np.array(predmerger[:,3]), np.array(prednonmerger[:,3])
sersic_merger, sersic_nonmerger = np.array(predmerger[:,1]), np.array(prednonmerger[:,1])
mass_merger, mass_nonmerger = np.array(predmerger[:,1]), np.array(prednonmerger[:,1])
sfr_merger, sfr_nonmerger = np.array(predmerger[:,1]), np.array(prednonmerger[:,1])
rkpc_merger, rkpc_nonmerger = np.array(predmerger[:,1]), np.array(prednonmerger[:,1])
# Plot mergers and non-mergers M20 vs Gini
plt.scatter(m20_merger, gini_merger,c='red', s=5)
plt.scatter(m20_nonmerger, gini_nonmerger, c='blue', s=5)
plt.title('Class Predictions for Test Data')
plt.xlim
plt.xlabel('M20')
plt.ylabel('Gini')
plt.legend(['Merger (Mergefrac)','Nonmerger (Mergefrac)'])
plt.savefig(path / 'TestPreds', dpi=200)
```

```
# Create numpy array of training data
train = np.array(training_data)
# Create empty array of training data color values for plotting
color = np.zeros((len(train)), dtype=str)
# Create empty arrays for Merger and Nonmerger data points
merger = np.empty((0,17), float)
nonmerger = np.empty((0,17), float)
for i in range(0,len(train)):
    # If true class of Galaxy i in training data is "Merger," its plot color value is Red
    if train[i,16] == 1:
        color[i] = 'reds'
        merger = np.append(merger, np.array([train[i]]), axis=0)
    else:
        # If true class of Galaxy i in training data is "Nonmerger," its plot color value is Blue
        color[i] = 'blue'
        nonmerger = np.append(nonmerger, np.array([train[i]]), axis=0)
# Create data arrays for merger and non-merger parameters for plotting
ID_merger, ID_nonmerger = np.array(merger[:,0]), np.array(nonmerger[:,0])
gini_merger, gini_nonmerger = np.array(merger[:,1]), np.array(nonmerger[:,1])
m20_merger, m20_nonmerger = np.array(merger[:,2]), np.array(nonmerger[:,2])
conc_merger, conc_nonmerger = np.array(merger[:,3]), np.array(nonmerger[:,3])
asym_merger, asym_nonmerger = np.array(merger[:,4]), np.array(nonmerger[:,4])
mass_merger, mass_nonmerger = np.array(merger[:,5]), np.array(nonmerger[:,5])
sfr_merger, sfr_nonmerger = np.array(merger[:,6]), np.array(nonmerger[:,6])
rkpc_merger, rkpc_nonmerger = np.array(merger[:,7]), np.array(nonmerger[:,7])
# Plot mergers and non-mergers M20 vs Gini
# Mergers in red
plt.scatter(m20_merger, gini_merger,c='red', s=5)
# Nonmergers in blue
plt.scatter(m20_nonmerger, gini_nonmerger, c='blue', s=5)
plt.title('True Classes for Training Data')
plt.xlabel('M20')
plt.ylabel('Gini')
plt.legend(['Merger (Mergefrac)','Nonmerger (Mergefrac)'])
plt.savefig(path / 'TrainingClasses', dpi=200)
# Mergers in red
plt.scatter(m20_merger, rkpc_merger,c='red', s=5)
# Nonmergers in blue
plt.scatter(m20_nonmerger, rkpc_nonmerger, c='blue', s=5)
plt.title('True Classes for Training Data')
plt.xlabel('M20')
plt.ylabel('rkpc')
plt.legend(['Merger (Mergefrac)','Nonmerger (Mergefrac)'])
plt.ylim((-0.5,10))
plt.savefig(path / 'TrainingClasses_M20_rkpc', dpi=200)
# Generate a 3D Scatter Plot
```

```
ax = plt.axes(projection='3d')
zdata = rkpc_merger
xdata = m20_merger
ydata = gini_merger
ax.scatter3D(m20_merger, asym_merger, rkpc_merger, c='r', s=5, alpha=0.75)
ax.scatter3D(m20_nonmerger, asym_nonmerger, rkpc_nonmerger, c='b', s=5, alpha=0.75)
ax.set_xlabel('M20')
ax.set_ylabel('Asymmetry')
ax.set_zlabel('Galactic Radius* [kpc]')
ax.set_ylim(-0.2,0.5)
ax.set_zlim(0,10)
plt.legend(['Merger (Mergefrac)','Nonmerger (Mergefrac)'])
plt.savefig(path / 'TrainingClasses_3D', dpi=200)
# Mergers in red
plt.scatter(conc_merger, asym_merger,c='red', s=5)
# Nonmergers in blue
plt.scatter(conc_nonmerger, asym_nonmerger, c='blue', s=5)
plt.title('True Classes for Training Data')
plt.xlim(left=1.5)
plt.ylim(bottom=-0.5)
plt.xlabel('Concentration')
plt.ylabel('Asymmetry')
plt.legend(['Merger (Mergefrac)','Nonmerger (Mergefrac)'])
plt.savefig(path / 'TrainingClasses_Conc_Asym', dpi=200)
# Mergers in red
plt.scatter(sfr_merger, rkpc_merger,c='red', s=5)
# Nonmergers in blue
plt.scatter(sfr_nonmerger, rkpc_nonmerger, c='blue', s=5)
plt.title('True Classes for Training Data')
plt.xlim(left=0)
plt.ylim(bottom=0, top=10)
plt.xlabel('Star Formation Rate')
plt.ylabel('Galactic Half-light Radius [kpc]')
plt.legend(['Merger (Mergefrac)','Nonmerger (Mergefrac)'])
plt.xlim(0,50)
plt.savefig(path / 'TrainingClasses_SFR_rkpc', dpi=200)
TrueVals = np.array(testing_data['Merger'])
Problematic_gals = np.empty((0,18), float)
Correct_nonmerger = np.empty((0,18), float)
Incorrect_nonmerger = np.empty((0,18), float)
Correct_merger = np.empty((0,18), float)
Incorrect_merger = np.empty((0,18), float)
Correct_preds = np.empty((0,18), float)
Incorrect_preds = np.empty((0,18), float)

for i in range(0,len(preds)):
```

```
    # If predicted nonmerger
    if preds[i]==0:
        # If predicted nonmerger is correct, its plot color is Green
        if preds[i] == TrueVals[i]:
            # Galaxy i added to list of correctly-predicted nonmergers
            Correct_nonmerger = np.append(Correct_nonmerger, np.array([test[i]]), axis=0)
            # Galaxy i is added to list of Correctly Predicted galaxy classes
            Correct_preds = np.append(Correct_preds, np.array([test[i]]), axis=0)
        else:
            # Galaxy added to list of incorrectly-predicted nonmergers
            Incorrect_nonmerger = np.append(Incorrect_nonmerger, np.array([test[i]]), axis=0)
            # Galaxy added to list of incorrect-predicted galaxy classes
            Incorrect_preds = np.append(Incorrect_preds, np.array([test[i]]), axis=0)
    else:
        if preds[i] == TrueVals[i]:  # If predicted merger is correct
            Correct_merger = np.append(Correct_merger, np.array([test[i]]), axis=0) # Galaxy i
added to list of
#                                                       correctly-predicted mergers
            Correct_preds = np.append(Correct_preds, np.array([test[i]]), axis=0) # Galaxy i is added
to list of
#                                                       Correctly Predicted galaxy classes
        else:  # If predicted merger is incorrect
            Incorrect_merger = np.append(Incorrect_nonmerger, np.array([test[i]]), axis=0) # Galaxy
added to list of
#                                                           incorrectly-predicted mergers
            Incorrect_preds = np.append(Incorrect_preds, np.array([test[i]]), axis=0) # Galaxy added
to list of
#                                                         incorrect-predicted galaxy classes

ID_correct_merger, ID_incorrect_merger = np.array(Correct_merger[:,0]),
np.array(Incorrect_merger[:,0])
ID_correct_nonmerger, ID_incorrect_nonmerger = np.array(Correct_nonmerger[:,0]),
np.array(Incorrect_nonmerger[:,0])
gini_correct_merger, gini_incorrect_merger = np.array(Correct_merger[:,1]),
np.array(Incorrect_merger[:,1])
gini_correct_nonmerger, gini_incorrect_nonmerger = np.array(Correct_nonmerger[:,1]),
np.array(Incorrect_nonmerger[:,1])
m20_correct_merger, m20_incorrect_merger = np.array(Correct_merger[:,2]),
np.array(Incorrect_merger[:,2])
m20_correct_nonmerger, m20_incorrect_nonmerger = np.array(Correct_nonmerger[:,2]),
np.array(Incorrect_nonmerger[:,2])
conc_correct_merger, conc_incorrect_merger = np.array(Correct_merger[:,3]),
np.array(Incorrect_merger[:,3])
conc_correct_nonmerger, conc_incorrect_nonmerger = np.array(Correct_nonmerger[:,3]),
np.array(Incorrect_nonmerger[:,3])
```

```
asym_correct_merger, asym_incorrect_merger = np.array(Correct_merger[:,4]),
np.array(Incorrect_merger[:,4])
asym_correct_nonmerger, asym_incorrect_nonmerger = np.array(Correct_nonmerger[:,4]),
np.array(Incorrect_nonmerger[:,4])
mass_correct_merger, mass_incorrect_merger = np.array(Correct_merger[:,5]),
np.array(Incorrect_merger[:,5])
mass_correct_nonmerger, mass_incorrect_nonmerger = np.array(Correct_nonmerger[:,5]),
np.array(Incorrect_nonmerger[:,5])
sfr_correct_merger, sfr_incorrect_merger = np.array(Correct_merger[:,6]),
np.array(Incorrect_merger[:,6])
sfr_correct_nonmerger, sfr_incorrect_nonmerger = np.array(Correct_nonmerger[:,6]),
np.array(Incorrect_nonmerger[:,6])
rkpc_correct_merger, rkpc_incorrect_merger = np.array(Correct_merger[:,7]),
np.array(Incorrect_merger[:,7])
rkpc_correct_nonmerger, rkpc_incorrect_nonmerger = np.array(Correct_nonmerger[:,7]),
np.array(Incorrect_nonmerger[:,7])
# Plot correct and incorrect predictions for Test Data
plt.scatter(m20_correct_merger, gini_correct_merger, c='blue', s=5) # Correct merger predictions
are Blue
plt.scatter(m20_correct_nonmerger, gini_correct_nonmerger, c='green', s=5) # Correct merger
predictions are Green
plt.scatter(m20_incorrect_merger, gini_incorrect_merger, c='orange', s=5) # Incorrect merger
predictions are Orange
plt.scatter(m20_incorrect_nonmerger, gini_incorrect_nonmerger, c='red', s=5) # Incorrect merger
predictions are Red
plt.title('Correct and Incorrect Predictions for Test Data')
plt.xlabel('M20')
plt.ylabel('Gini')
plt.legend(['True Merger (Mergefrac)', 'True Nonmerger (Mergefrac)',
        'False Merger (Mergefrac)', 'False Nonmerger (Mergefrac)'])
plt.savefig(path / 'PredictedClasses_M20_Gini', dpi=200)
# Plot correct and incorrect predictions for Test Data (Concentration vs Asymmetry)
plt.scatter(conc_correct_merger, asym_correct_merger, c='blue', s=5) # Correct merger
predictions are Blue
plt.scatter(conc_correct_nonmerger, asym_correct_nonmerger, c='green', s=5) # Correct merger
predictions are Green
plt.scatter(conc_incorrect_merger, asym_incorrect_merger, c='orange', s=5) # Incorrect merger
predictions are Orange
plt.scatter(conc_incorrect_nonmerger, asym_incorrect_nonmerger, c='red', s=5) # Incorrect
merger predictions are Red
plt.title('Correct and Incorrect Predictions for Test Data')
plt.xlabel('Concentration')
plt.ylabel('Asymmetry')
plt.ylim(-0.25,0.75)
plt.legend(['True Merger (Mergefrac)', 'True Nonmerger (Mergefrac)',
        'False Merger (Mergefrac)', 'False Nonmerger (Mergefrac)'])
```

```
plt.savefig(path / 'PredictedClasses_Conc_Asym', dpi=200)
# Plot correct and incorrect predictions for Test Data (m20 vs m20)
plt.scatter(asym_correct_merger, m20_correct_merger, c='blue', s=5) # Correct merger
predictions are Blue
plt.scatter(asym_correct_nonmerger, m20_correct_nonmerger, c='green', s=5) # Correct merger
predictions are Green
plt.scatter(asym_incorrect_merger, m20_incorrect_merger, c='orange', s=5) # Incorrect merger
predictions are Orange
plt.scatter(asym_incorrect_nonmerger, m20_incorrect_nonmerger, c='red', s=5) # Incorrect
merger predictions are Red
plt.title('Correct and Incorrect Predictions for Test Data')
plt.xlabel('Asymmetry')
plt.ylabel('M20')
plt.legend(['True Merger (Mergefrac)', 'True Non-merger (Mergefrac)',
          'False Merger (Mergefrac)', 'False Non-merger (Mergefrac)'])
plt.savefig(path / 'PredictedClasses_Asym_m20', dpi=200)
# Write text file containing Problematic/Misclassified galaxies
filename = path / 'Problematic_galaxies.txt'
print(filename)

file = open(filename, 'w')
file.write("Run was completed on "+currentdate+" at "+currenttime+"\n\nID, RA, Dec, X
Position, Y Position\n\n")
for i in range(len(Incorrect_preds)):
   file.write(
str(Incorrect_preds[i,0].astype(int))+','+str(Incorrect_preds[i,8])+','+str(Incorrect_preds[i,9])+','+
str(round(Incorrect_preds[i,12]))+','+str(round(Incorrect_preds[i,13]))+'\n' )
   if i == len(Incorrect_preds)-1:
      print('File Saved!')
      file.close()
# Load CANDELS mosaics for galaxy cutouts
gdn_file = mos_path / 'f160w' / 'goodsn.fits'
gds_file = mos_path / 'f160w' / 'goodss.fits'
uds_file = mos_path / 'f160w' / 'uds.fits'
cos_file = mos_path / 'f160w' / 'cos.fits'
egs_file = mos_path / 'f160w' / 'egs.fits'
gdn_mos = fits.getdata(gdn_file, dtype='float64')
gds_mos = fits.getdata(gds_file, dtype='float64')
uds_mos = fits.getdata(uds_file, dtype='float64')
cos_mos = fits.getdata(cos_file, dtype='float64')
egs_mos = fits.getdata(egs_file, dtype='float64')
# Convert Pixels to WCS
w_gds = WCS(str(gds_file))
w_gdn = WCS(str(gds_file))
w_uds = WCS(str(uds_file))
w_cos = WCS(str(cos_file))
```

```
w_egs = WCS(str(egs_file))
# Create thumbnail cutouts of incorrectly classified galaxies
# First, we make a directory to save the thumbnail images to
figpath = path+'\IncorrectPreds'
try:
    os.mkdir(figpath)
except OSError:
    print('Creation of directory %s failed' % figpath)
else:
    print('Creation of directory %s successful!' % figpath)
for i in range (len(Incorrect_preds)):
    RA = Incorrect_preds[i,8]
    Dec = Incorrect_preds[i,9]
    n_gal = str(Incorrect_preds[i,0])
    n_gal = n_gal[:-2]
    size = (150,150)
    if Incorrect_preds[i,14]==1:
        field = 'GOODS-S'
        mosaic = gds_mos
        pos = w_gds.wcs_world2pix(RA, Dec, 0)
    elif Incorrect_preds[i,14] == 2:
        field = 'UDS'
        mosaic = uds_mos
        pos = w_uds.wcs_world2pix(RA, Dec, 0)
    elif Incorrect_preds[i,14] == 3:
        field = 'GOODS-N'
        mosaic = gdn_mos
        pos = w_gdn.wcs_world2pix(RA, Dec, 0)
    elif Incorrect_preds[i,14] == 4:
        field = 'COSMOS'
        mosaic = cos_mos
        pos = w_cos.wcs_world2pix(RA, Dec, 0)
    else:
        field = 'EGS'
        mosaic = egs_mos
        pos = w_egs.wcs_world2pix(RA, Dec, 0)
    cutout = Cutout2D(mosaic, pos, size)
    plt.figure()
    im, norm = imshow_norm(cutout.data, cmap='gray', origin='lower', vmin=0.000001,
vmax=10, stretch=LogStretch())
    plt.title('Galaxy '+n_gal)
    if Incorrect_preds[i,16] == 0:
        trueclass = 'Nonmerger (Mergefrac)'
    else:
        trueclass = 'Merger (Mergefrac)'
    if Incorrect_preds[i,17] == 0:
```

```
        predictedclass = 'Nonmerger (Mergefrac)'
    else:
        predictedclass = 'Merger (Mergefrac)'
    plt.figtext(0.8, 0.5, 'Field: '+field+'\nRA: '+str(RA)+'\nDec: '+str(Dec)+'\nTrue Class:
'+trueclass+'\nPredicted Class: '+predictedclass)
    plt.savefig(figpath+'/Galaxy'+n_gal+'.png', dpi=200, bbox_inches='tight')

# Create thumbnail cutouts of Training Data
# First, we make a directory to save the training nonmerger thumbnail images to
trainingnonmergerpath = path+'\TrainingNonmergers'
trainingmergerpath = path+'\TrainingMergers'
try:
    os.mkdir(trainingnonmergerpath)
except OSError:
    print('Creation of directory %s failed' % figpath)
else:
    print('Creation of directory %s successful!' % figpath)
# Make a directory to save the training merger thumbnail images to
try:
    os.mkdir(trainingmergerpath)
except OSError:
    print('Creation of directory %s failed' % figpath)
else:
    print('Creation of directory %s successful!' % figpath)
merger_count = 0
nonmerger_count = 0
for i in range(len(train)):
    if merger_count + nonmerger_count == 40:
        print('Merger Count = ', merger_count)
        print('Nonmerger Count = ', nonmerger_count)
        break
    else:
        RA = train[i,8]
        Dec = train[i,9]
        n_gal = str(train[i,0])
        n_gal = n_gal[:-2]
        size = (150,150)
        if train[i,14]==1:
            field = 'GOODS-S'
            mosaic = gds_mos
            pos = w_gds.wcs_world2pix(RA, Dec, 0)
        elif train[i,14] == 2:
            field = 'UDS'
            mosaic = uds_mos
            pos = w_uds.wcs_world2pix(RA, Dec, 0)
        elif train[i,14] == 3:
```

```
            field = 'GOODS-N'
            mosaic = gdn_mos
            pos = w_gdn.wcs_world2pix(RA, Dec, 0)
        elif train[i,14] == 4:
            field = 'COSMOS'
            mosaic = cos_mos
            pos = w_cos.wcs_world2pix(RA, Dec, 0)
        else:
            field = 'EGS'
            mosaic = egs_mos
            pos = w_egs.wcs_world2pix(RA, Dec, 0)
        cutout = Cutout2D(mosaic, pos, size)
        plt.figure()
        im, norm = imshow_norm(cutout.data, cmap='gray', origin='lower', vmin=0.000001,
vmax=10, stretch=LogStretch())
        plt.title('Galaxy '+n_gal)
        if train[i,16] == 0:
            if nonmerger_count == 20:
                continue
            else:
                plt.figtext(0.8, 0.5, 'True Class: Non-merger (Mergefrac)\n'+'Merger Fraction =
'+str(train[i,15]))
                plt.savefig(trainingnonmergerpath+'/Galaxy'+n_gal+'.png', dpi=200,
bbox_inches='tight')
                nonmerger_count += 1
        if train[i,16] == 1:
            if merger_count == 20:
                continue
            else:
                plt.figtext(0.8, 0.5, 'True Class: Merger (Mergefrac)\n'+'Merger Fraction =
'+str(train[i,15]))
                plt.savefig(trainingmergerpath+'/Galaxy'+n_gal+'.png', dpi=200, bbox_inches='tight')
                merger_count += 1
# Create thumbnail cutouts of Training Data
# First, we make a directory to save the training nonmerger thumbnail images to
testingnonmergerpath = path+'\TestingNonmergers'
testingmergerpath = path+'\TestingMergers'
try:
    os.mkdir(testingnonmergerpath)
except OSError:
    print('Creation of directory %s failed' % figpath)
else:
    print('Creation of directory %s successful!' % figpath)
# Make a directory to save the training merger thumbnail images to
try:
    os.mkdir(testingmergerpath)
```

```
except OSError:
    print('Creation of directory %s failed' % figpath)
else:
    print('Creation of directory %s successful!' % figpath)
merger_count = 0
nonmerger_count = 0
for i in range(len(test)):
    if merger_count + nonmerger_count == 40:
        print('Merger Count = ', merger_count)
        print('Nonmerger Count = ', nonmerger_count)
        break
    else:
        RA = test[i,8]
        Dec = test[i,9]
        n_gal = str(test[i,0])
        n_gal = n_gal[:-2]
        size = (150,150)
        if test[i,14]==1:
            field = 'GOODS-S'
            mosaic = gds_mos
            pos = w_gds.wcs_world2pix(RA, Dec, 0)
        elif test[i,14] == 2:
            field = 'UDS'
            mosaic = uds_mos
            pos = w_uds.wcs_world2pix(RA, Dec, 0)
        elif test[i,14] == 3:
            field = 'GOODS-N'
            mosaic = gdn_mos
            pos = w_gdn.wcs_world2pix(RA, Dec, 0)
        elif test[i,14] == 4:
            field = 'COSMOS'
            mosaic = cos_mos
            pos = w_cos.wcs_world2pix(RA, Dec, 0)
        else:
            field = 'EGS'
            mosaic = egs_mos
            pos = w_egs.wcs_world2pix(RA, Dec, 0)
        cutout = Cutout2D(mosaic, pos, size)
        plt.figure()
        im, norm = imshow_norm(cutout.data, cmap='gray', origin='lower', vmin=0.000001,
vmax=10, stretch=LogStretch())
        plt.title('Galaxy '+n_gal)
        if test[i,16] == 0:
            if nonmerger_count == 20:
                continue
            else:
```

```
            plt.figtext(0.8, 0.5, 'True Class: Non-merger (Mergefrac)\n'+'Merger Fraction =
'+str(test[i,15]))
            plt.savefig(testingnonmergerpath+'/Galaxy'+n_gal+'.png', dpi=200,
bbox_inches='tight')
            nonmerger_count += 1
        if test[i,16] == 1:
            if merger_count == 20:
                continue
            else:
                plt.figtext(0.8, 0.5, 'True Class: Merger (Mergefrac)\n'+'Merger Fraction =
'+str(test[i,15]))
                plt.savefig(testingmergerpath+'/Galaxy'+n_gal+'.png', dpi=200, bbox_inches='tight')
                merger_count += 1
plt.close('all')
print('Finished!')
```

## APPENDIX B: COMPUTER VISION DATASET GENERATOR PYTHON CODE

```
# This code will learn to classify images of galaxy mergers in the CANDELS fields
# Import modules
from astropy import units as u
from astropy.io import fits
from astropy.nddata import Cutout2D
from astropy.table import Table
from astropy.visualization import imshow_norm, SqrtStretch, LogStretch
from astropy.wcs import WCS
from datetime import datetime
import glob
import itertools
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from keras.models import Sequential, model_from_json, load_model
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array,
load_img
from keras.utils import to_categorical
import matplotlib as mpl
from matplotlib.colors import LogNorm
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np
from numpy import asarray, concatenate
import os
import pandas as pd
from pathlib import Path
from PIL import Image
from pprint import pprint
from sklearn.metrics import confusion_matrix, roc_curve
from sklearn.model_selection import train_test_split
# Define Paths
homepath = Path(str(os.path.normpath(os.getcwd() + os.sep + os.pardir)))
mos_path = homepath / 'Mosaics'
params_path = homepath / 'StructuralParameters'
indfeats_path = homepath / 'CV_Data' / 'Merger' / 'Ind_feats'
asymclass_path = indfeats_path / 'Asym'
tidalclass_path = indfeats_path / 'Tidal'
mergeclass_path = indfeats_path / 'Merge'
mergefracclass_path = indfeats_path / 'Mergefrac'
# First, we will generate training/validation data sets for Computer Vision classifier
# Cutouts will be generated in five different HST WFC3 filters: F105W, F125W, F140W, and
F160W
# Import CANDELS data tables
```

```
# These tables include measured galaxy parameters such as redshift, mass, size, field, etc.
# Read CANDELS .fits tables into Python
gds = fits.open(params_path / 'CANDELS.GDS.1018.wCAS_VC.fits')
uds = fits.open(params_path / 'CANDELS.UDS.1018.wCAS_VC.fits')
gdn = fits.open(params_path / 'CANDELS.GDN.1018.wCAS_VC.fits')
cos = fits.open(params_path / 'CANDELS.COS.1018.wCAS_VC.fits')
egs = fits.open(params_path / 'CANDELS.EGS.1018.wCAS_VC.fits')
# Extract data array from CANDELS .fits tables
gds_data = Table(gds[1].data)
uds_data = Table(uds[1].data)
gdn_data = Table(gdn[1].data)
cos_data = Table(cos[1].data)
egs_data = Table(egs[1].data)
# Remove unused variables
del gds, uds, gdn, cos, egs
# Assign numerical values for field
gds_data['Field'] = 1  # 1 = GOODS-S
uds_data['Field'] = 2  # 2 = UDS
gdn_data['Field'] = 3  # 3 = GOODS-N
cos_data['Field'] = 4  # 4 = COSMOS
egs_data['Field'] = 5  # 5 = EGS
# Create numpy arrays of features to be trained on
features_gds = np.array([gds_data['ID'], gds_data['Field'], gds_data['HMAG'],
gds_data['ZBEST'], gds_data['MASS'], gds_data['VC_F_MERGER'], gds_data['VC_F_ASYM'],
gds_data['VC_F_TIDAL'], gds_data['RA'], gds_data['DEC'], gds_data['X_IMAGE'],
gds_data['Y_IMAGE']])
features_uds = np.array([uds_data['ID'], uds_data['Field'], uds_data['HMAG'],
uds_data['ZBEST'], uds_data['MASS'], uds_data['VC_F_MERGER'], uds_data['VC_F_ASYM'],
uds_data['VC_F_TIDAL'], uds_data['RA'], uds_data['DEC'], uds_data['X_IMAGE'],
uds_data['Y_IMAGE']])
features_gdn = np.array([gdn_data['ID'], gdn_data['Field'], gdn_data['HMAG'],
gdn_data['ZBEST'], gdn_data['MASS'], gdn_data['VC_F_MERGER'],
gdn_data['VC_F_ASYM'], gdn_data['VC_F_TIDAL'], gdn_data['RA'], gdn_data['DEC'],
gdn_data['X_IMAGE'], gdn_data['Y_IMAGE']])
features_cos = np.array([cos_data['ID'], cos_data['Field'], cos_data['HMAG'],
cos_data['ZBEST'], cos_data['MASS'], cos_data['VC_F_MERGER'], cos_data['VC_F_ASYM'],
cos_data['VC_F_TIDAL'], cos_data['RA'], cos_data['DEC'], cos_data['X_IMAGE'],
cos_data['Y_IMAGE']])
features_egs = np.array([egs_data['ID'], egs_data['Field'], egs_data['HMAG'],
egs_data['ZBEST'], egs_data['MASS'], egs_data['VC_F_MERGER'], egs_data['VC_F_ASYM'],
egs_data['VC_F_TIDAL'], egs_data['RA'], egs_data['DEC'], egs_data['X_IMAGE'],
egs_data['Y_IMAGE']])
# Remove unused variables
del gds_data, uds_data, gdn_data, cos_data, egs_data
# Concatenate to create one features array
```

```
features = np.concatenate((features_gds, features_uds, features_cos, features_egs, features_gdn),
axis=1)
# Remove unused variables
del features_gds, features_uds, features_gdn, features_cos, features_egs
features = np.transpose(features)
print('Total number of galaxies before cuts: ', len(features))
# Create pandas dataframe, labelling columns appropriately
features = pd.DataFrame(features, columns = ['ID', 'Field', 'HMAG', 'ZBEST', 'MASS',
'VC_F_MERGER', 'VC_F_ASYM', 'VC_F_TIDAL', 'RA', 'Dec',  'X_IMAGE', 'Y_IMAGE'])
features['merge_frac'] =
(features['VC_F_MERGER']+features['VC_F_ASYM']+features['VC_F_TIDAL'])/3
mergers = features[features.VC_F_MERGER >= 0.6]
# Define magnitude, redshift, and mass cutoffs for data points
features = features[(features.HMAG <= 24.5) & (features.HMAG >= 0)]  # Magnitude cutoff of
24.5
features = features[(features.ZBEST > 0.5) & (features.ZBEST <= 2.5)]  # 0.5<z<2.5
features = features[features.MASS >= 9] # Mass minimum of 10^9 solar mass
# Create flags for Merge_frac, Asymmetry, Merger, and Tidal Arms
features['merger_flag'] = np.where(features['VC_F_MERGER'] >= 0.6, True, False)
features['asym_flag'] = np.where(features['VC_F_ASYM'] >= 0.8, True, False)
features['tidal_flag'] = np.where(features['VC_F_TIDAL'] >= 0.6, True, False)
features['mergefrac_flag'] = np.where(features['merge_frac'] >= 0.6, True, False)
features = features.sample(frac=1)  # Shuffle the data randomly
features = features.dropna()  # Drop rows with NaN values from data
# Prepare training/testing data arrays
n_training = len(features)*0.75  # Number of training data points (75% of all data points are
training points)
n_training = round(n_training)  # Number of training data points
n_testing = len(features) - n_training  # Number of testing data points
data_array = np.array(features, float)  # Create numpy array containing data features
training_data = features.iloc[:n_training]  # Create training data array
testing_data = features.iloc[n_training:]  # Create testing data array
del features, data_array, mergers
training_data = training_data.to_numpy()
testing_data = testing_data.to_numpy()
# Define number of training/testing mergers/nonmergers
n_train_mergers = np.sum(training_data[:,13])
print('Number of training mergers: ', n_train_mergers)
n_train_nonmergers = len(training_data)-n_train_mergers
print('Number of training nonmergers: ', n_train_nonmergers)
n_test_mergers = np.sum(testing_data[:,13])
print('Number of testing mergers: ', n_test_mergers)
n_test_nonmergers = len(testing_data)-n_test_mergers
print('Number of testing nonmergers: ', n_test_nonmergers)
# Determine how many of each class in Training Data
n_training = len(training_data)
```

```
n_train_asym = np.sum(training_data[:,14])
n_train_sym = n_training-n_train_asym
n_train_tidal = np.sum(training_data[:,15])
n_train_atidal = n_training-n_train_tidal
n_train_merger = np.sum(training_data[:,13])
n_train_nonmerger = n_training-n_train_merger
n_train_mergefrac = np.sum(training_data[:,16])
n_train_nonmergefrac = n_training-n_train_mergefrac
print("This is the makeup of the Training Data array")
print("Total number of galaxies: ", n_training)
print("Symmetric Galaxies: ", n_train_sym)
print("Asymmetric Galaxies: ", n_train_asym)
print("Galaxies with Tidal Arms: ", n_train_tidal)
print("Galaxies without Tidal Arms: ", n_train_atidal)
print("Mergers: ", n_train_merger)
print("Nonmergers: ", n_train_nonmerger)
print("Mergers (mergefrac): ", n_train_mergefrac)
print("Nonmergers (mergefrac): ", n_train_nonmergefrac)
# Determine how many of each class in Training Data
n_testing = len(testing_data)
n_test_asym = np.sum(testing_data[:,14])
n_test_sym = n_testing-n_test_asym
n_test_tidal = np.sum(testing_data[:,15])
n_test_atidal = n_testing-n_test_tidal
n_test_merger = np.sum(testing_data[:,13])
n_test_nonmerger = n_testing-n_test_merger
n_test_mergefrac = np.sum(testing_data[:,16])
n_test_nonmergefrac = n_testing-n_test_mergefrac
print("This is the makeup of the Testing Data array")
print("Total number of galaxies: ", n_testing)
print("Symmetric Galaxies: ", n_test_sym)
print("Asymmetric Galaxies: ", n_test_asym)
print("Galaxies with Tidal Arms: ", n_test_tidal)
print("Galaxies without Tidal Arms: ", n_test_atidal)
print("Mergers: ", n_test_merger)
print("Nonmergers: ", n_test_nonmerger)
print("Mergers (mergefrac): ", n_test_mergefrac)
print("Nonmergers (mergefrac): ", n_test_nonmergefrac)
filters = ['f105w', 'f125w', 'f140w', 'f160w']
# Create a function to generate cutout images of galaxies in each filter
def gal_cutout(filters, feat, traintest):
    j = 0  # Counter for asymmetric cutouts
    k = 0  # Counter for nonasymmetric cutouts


    if traintest == 'train':
```

```
    datatype = 'Training'
    dataarray = training_data
elif traintest == 'test':
    datatype = 'Testing'
    dataarray = testing_data
X_pos = np.empty((50,50), dtype=float)  # Create empty array for positive class training data
X_neg = np.empty((50,50), dtype=float)  # Create empty array for negative class training data
for h in range (len(filters)):

    currentfilter = filters[h]  # Define which filter the function is currently making cutouts for

    if feat == 1:
        feat_path = asymclass_path
        posclass = 'Asymmetric'
        negclass = 'Symmetric'
        featcol = 6
        featflagcol = 14
    elif feat == 2:
        feat_path = tidalclass_path
        posclass = 'Tidal'
        negclass = 'Nontidal'
        featcol = 7
        featflagcol = 15
    elif feat == 3:
        feat_path = mergeclass_path
        posclass = 'Merger'
        negclass = 'Nonmerger'
        featcol = 5
        featflagcol = 13
    elif feat == 4:
        feat_path = mergefracclass_path
        posclass = 'Merger (mergefrac)'
        negclass = 'Nonmerger (mergefrac)'
        featcol = 12
        featflagcol = 16

    posplot_path = feat_path / 'Plots' / datatype / posclass / currentfilter
    negplot_path = feat_path / 'Plots' / datatype / negclass / currentfilter

    filetype = '*png'


    # Define paths for CANDELS mosaics
    cos_file = mos_path / currentfilter / 'cos.fits'
    egs_file = mos_path / currentfilter / 'egs.fits'
    gdn_file = mos_path / currentfilter / 'goodsn.fits'
```

```
    gds_file = mos_path / currentfilter / 'goodss.fits'
    uds_file = mos_path / currentfilter / 'uds.fits'
    # Load data for each mosaic
    cos_mos = fits.getdata(cos_file, dtype='float64')
    egs_mos = fits.getdata(egs_file, dtype='float64')
    gdn_mos = fits.getdata(gdn_file, dtype='float64')
    gds_mos = fits.getdata(gds_file, dtype='float64')
    uds_mos = fits.getdata(uds_file, dtype='float64')
    # Extract WCS coordinates for each image
    w_cos = WCS(str(cos_file))
    w_egs = WCS(str(egs_file))
    w_gdn = WCS(str(gdn_file))
    w_gds = WCS(str(gds_file))
    w_uds = WCS(str(uds_file))

    # Remove previous images from Positive and Negative Class directories

    for f in glob.glob(str(posplot_path / filetype)):
        try:
            os.remove(f)
        except OSError as e:
            print("Error: %s : %s" % (f, e.strerror))

    for f in glob.glob(str(negplot_path / filetype)):
        try:
            os.remove(f)
        except OSError as e:
            print("Error: %s : %s" % (f, e.strerror))
    for i in range (len(dataarray)):
        if i%1000 == 0:
            print('i = ',i)
        if dataarray[i,featflagcol] == True or k <= j:
            field = 'FIELD ERROR'
            RA = dataarray[i,8]  # Determine RA for galaxy i
            Dec = dataarray[i,9] # Determine Declination for galaxy i
            z = dataarray[i,3]
            n_gal = str(dataarray[i,0]) # Determine ID of galaxi i
            n_gal = n_gal[:-2]
            size = (50,50)  # Define image size: 50x50 pixels
            if dataarray[i,1]==1:  # Determine Field of galaxy i
                field = 'GOODS-S'
                mosaic = gds_mos  # Determine which field mosaic to cutout from
                pos = w_gds.wcs_world2pix(RA, Dec, 0)  # Determine WCS coordinates of galaxy
within the mosaic
            elif dataarray[i,1] == 2:  # Repeat for other CANDELS fields
                field = 'UDS'
```

```
            mosaic = uds_mos
            pos = w_uds.wcs_world2pix(RA, Dec, 0)
        if dataarray[i,1] == 3:
            field = 'GOODS-N'
            mosaic = gdn_mos
            pos = w_gdn.wcs_world2pix(RA, Dec, 0)
        elif dataarray[i,1] == 4:
            field = 'COSMOS'
            mosaic = cos_mos
            pos = w_cos.wcs_world2pix(RA, Dec, 0)
        elif dataarray[i,1] == 5:
            field = 'EGS'
            mosaic = egs_mos
            pos = w_egs.wcs_world2pix(RA, Dec, 0)
        cutout = Cutout2D(mosaic, pos, size)  # Generate cutout
# Skip cutout if image is empty (this occurs due to filters not completely overlapping)
        if np.max(cutout.data) == 0:
            print(field+' Galaxy '+str(dataarray[i,0])+' image is empty')
            continue
        plt.figure()  # Plot cutout image for user
# Plot a normalized representation (NOTE: data has NOT been normalized, just visually
presented as such)
        im, norm = imshow_norm(cutout.data, origin='lower', cmap='gray')
        plt.title('Galaxy '+n_gal)
        if dataarray[i,featcol] == 0:  # Determine whether galaxy i is negative (class 0) or
positive (class 1)
            trueclass = negclass
            path = negplot_path
            label = 0
            k += 1
            print('k = '+ str(k))
        elif dataarray[i,featcol] >= 0.6:
            trueclass = posclass
            label = 1
            path = posplot_path
            j += 1
            print('j = '+ str(j))
        else:
            continue
        plt.figtext(0.8, 0.5, 'Field: '+field+'\nFilter: '+currentfilter+'\nRA: '+str(RA)+'\nDec:
'+str(Dec)+'\nTrue Class: '+trueclass+'\nRedshift: '+str(z))  # Add galaxy info to the plot
        plt.subplots_adjust(right=0.8)
        figname = 'Galaxy'+n_gal+'.png'
        plt.savefig(path / figname, dpi=200)  # Save the figure as .png file
        plt.show()
```

```
            plt.figure().clear()  # Clear plot from memory after it has been saved to prevent
reaching memory limit
            plt.close()  # Close plot after being cleared
            cutout_data = cutout.data  # Create numpy array of cutout image data
            if label == 0:
                X_neg = np.dstack((X_neg, cutout_data))  # Add cutout image data to trainX array
                print('Negative Class Array Shape: '+str(X_neg.shape))
                if k == 1:
                    np.delete(X_neg, 0, 2)
                    print('Deleted empty image in Negative Class array')
            elif label == 1:
                X_pos = np.dstack((X_pos, cutout_data))  # Add cutout image data to trainX array
                print('Positive Class Array Shape: '+str(X_pos.shape))
                if j == 1:
                    np.delete(X_pos, 0, 2)
                    print('Deleted empty image in Positive Class array')
            del field, RA, Dec, n_gal, size, mosaic, pos, cutout, im, norm, trueclass, path, label,
cutout_data
        del cos_mos, gdn_mos, gds_mos, egs_mos, uds_mos
    print('Final X_pos shape: ', X_pos.shape)
    print('Final X_neg shape: ', X_neg.shape)

    # Save numpy arrays to respective folders
    posfilename = datatype+posclass+'AllFilters_data.npy'
    negfilename = datatype+negclass+'AllFilters_data.npy'
    filepath = feat_path / 'Plots' / datatype / posclass / posfilename
    with open(filepath, 'wb') as f:
        np.save(filepath, X_pos)
        print('Positive Class Array Saved!')
    filepath = feat_path / 'Plots' / datatype / negclass / negfilename
    with open(filepath, 'wb') as f:
        np.save(filepath, X_neg)
        print('Negative Class Array Saved!')

# Define a function that will augment datasets based on feature, class and datatype (training vs
testing)
def im_augment(feat, traintest):
    if feat == 1:
        feat_path = asymclass_path
        posclass = 'Asymmetric'
        negclass = 'Symmetric'
        if traintest == 'train':
            n_class = n_train_asym
        else:
            n_class = n_test_asym
    elif feat == 2:
```

```
    feat_path = tidalclass_path
    posclass = 'Tidal'
    negclass = 'Nontidal'
    if traintest == 'train':
        n_class = n_train_tidal
    else:
        n_class = n_test_tidal
elif feat == 3:
    feat_path = mergeclass_path
    posclass = 'Merger'
    negclass = 'Nonmerger'
    if traintest == 'train':
        n_class = n_train_merger
    else:
        n_class = n_test_merger
elif feat == 4:
    feat_path = mergefracclass_path
    posclass = 'Merger (mergefrac)'
    negclass = 'Nonmerger (mergefrac)'
    if traintest == 'train':
        n_class = n_train_mergefrac
    else:
        n_class = n_test_mergefrac
if traintest == 'train':
    datatype = 'Training'
elif traintest == 'test':
    datatype = 'Testing'

posfile_name = datatype+posclass+'AllFilters_data.npy'
negfile_name = datatype+negclass+'AllFilters_data.npy'

posdata_path = feat_path / 'Plots' / datatype / posclass / posfile_name
negdata_path = feat_path / 'Plots' / datatype / negclass / negfile_name
posim_path = feat_path / 'Images' / datatype / posclass
negim_path = feat_path / 'Images' / datatype / negclass

filetype = '*png'

# Delete old Positive and Negative Class images

for f in glob.glob(str(posim_path / filetype)):
    try:
        os.remove(f)
    except OSError as e:
        print("Error: %s : %s" % (f, e.strerror))
for f in glob.glob(str(negim_path / filetype)):
```

```
        try:
            os.remove(f)
        except OSError as e:
            print("Error: %s : %s" % (f, e.strerror))

    # Define augmentation function

    datagen = ImageDataGenerator(
        rotation_range=180,  # Rotate up to 180 degrees in either direction
        horizontal_flip=True,  # Randomly flip image horizontally
        vertical_flip=True,  # Randomly flip image vertically
        brightness_range=[0.8,1.2],  # Randomly adjust brightness by +/- 20%
        fill_mode='constant',  # Fill negative image space with constant value
        cval=0)

    # Load positive class array and apply augmentation function

    X_pos = np.load(posdata_path)
    X_pos = np.moveaxis(X_pos, 2, 0)  # Rearrange axes for augmentation function

# Reshape array for augmentation function
    X_pos = X_pos.reshape(X_pos.shape[0],X_pos.shape[1],X_pos.shape[2],1,1)

    i = 0
    for batch in datagen.flow(X_pos[:,:,:,0], batch_size=1,
                              save_to_dir=posim_path, save_prefix='1_Galaxy', save_format='png'):
        i += 1
        if i > len(X_pos)*8:  # Define how many images to generate
            break  # otherwise the generator would loop indefinitely

    # Load negative class array and apply augmentation function

    X_neg = np.load(negdata_path)
    X_neg = np.moveaxis(X_neg, 2, 0)  # Rearrange axes for augmentation function
    X_neg = X_neg.reshape(X_neg.shape[0],X_neg.shape[1],X_neg.shape[2],1,1)  # Reshape
array for augmentation function

    i = 0
    for batch in datagen.flow(X_neg[:,:,:,0], batch_size=1,
                              save_to_dir=negim_path, save_prefix='1_Galaxy', save_format='png'):
        i += 1
        if i > len(X_neg)*8:  # Define how many images to generate
            break  # otherwise the generator would loop indefinitely

    # Validate testing images before converting to numpy array
    while True:
```

```
        try:
            confirmation = input('WAIT! Please confirm testing images are not corrupt. Press y when
complete: ')
            if confirmation == 'y':
                break
            print('Please confirm images and press Y')
        except Exception as e:
            print(e)
    # Load positive class images and convert to numpy array
    X_pos = np.empty((50,50,1,1), dtype=float)  # Create empty array for training data

    i=0

    imtype = '*png'
    for filename in glob.glob(str(posim_path / imtype)): # For all .png files in the directory
        im=Image.open(filename)
        data = asarray(im)
        data = data.reshape(data.shape[0], data.shape[1], 1, 1)
        label = 0
        X_pos = np.append(X_pos, data, axis=2)  # Add cutout image data to trainX array
        i+=1
        if i%1000 == 0:
            print(i)
    print('X_pos shape: '+str(X_pos.shape))
    # Rearrange trainX axes for training
    X_pos = np.moveaxis(X_pos,2,0)  # Move index column to be the first column
    X_pos = np.delete(X_pos, 0,0)  # Delete first "image," which is the empty image when the
array was created
    Y_pos = np.ones(len(X_pos))
    print('X_pos shape: '+str(X_pos.shape))
    print('X_pos shape: '+str(X_pos.shape))
    # Save validation merger image data array
    X_neg = np.empty((50,50,1,1), dtype=float)  # Create empty array for training data
    i=0
    for filename in glob.glob(str(negim_path / imtype)):
        im=Image.open(filename)
        data = asarray(im)
        data = data.reshape(data.shape[0], data.shape[1], 1, 1)
        label = 0
        X_neg = np.append(X_neg, data, axis=2)  # Add cutout image data to trainX array
        i+=1
        if i%1000 == 0:
            print(i)
    print('X_neg shape: '+str(X_neg.shape))
    # Rearrange trainX axes for training
    X_neg = np.moveaxis(X_neg,2,0)  # Move index column to be the first column
```

```
    X_neg = np.delete(X_neg, 0,0)  # Delete first "image," which is the empty image when the
array was created
    Y_neg = np.zeros(len(X_neg))
    print('X_neg shape: '+str(X_neg.shape))
    print('X_neg shape: '+str(X_neg.shape))

    # We create "master" data arrays for data which include image data AND labels
    # Establish trainX nonmerger and merger image and label arrays
    # Append image and label arrays
    f = np.append(X_neg,X_pos, axis=0)
    g = np.append(Y_neg,Y_pos, axis=0)
    print(f.shape)
    # Add new dimension to data arrays to accomodate for label indice
    h = np.expand_dims(f,4)
    print(h.shape)
    # Write in labels for each image
    for i in range(len(g)):
        h[i,0,0,0,0] = g[i]
    np.random.shuffle(h)  # Shuffle data array randomly (this only shuffles the first axis, so pixel
data and labels stay together)
    masterfile_name = datatype+'_data_master.npy'
    mastertestpath = feat_path / 'Images' / datatype
    # Save positive class image data array
    filename = mastertestpath / masterfile_name
    with open(filename, 'wb') as f:
        np.save(filename, h)

    del X_pos, Y_pos, X_neg, Y_neg, f, g, h
# Features
# 1 = Asymmetry
# 2 = Tidal Arms
# 3 = Merger
# 4 = Merge Fraction
len(testing_data)
gal_cutout(filters, 1, 'test')
del testing_data
gal_cutout(filters, 1, 'train')
del training_data
im_augment(1, 'test')
im_augment(1, 'train')
print('Finished!')
```

## APPENDIX C: COMPUTER VISION MERGER CLASSIFIER PYTHON CODE

```
# Import packages
import numpy as np
import os.path
from numpy import concatenate
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from datetime import datetime
from astropy.visualization import imshow_norm, SqrtStretch, LogStretch
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from sklearn.metrics import roc_curve
from keras.models import load_model
from keras.callbacks import EarlyStopping, ModelCheckpoint
# Load training and testing datasets
trainX_master = np.load(r'C:\Users\Alex Koch\Documents\Python\Research\Machine
Learning\CV_Data\Merger\Ind_feats\Tidal\Images\Training\Training_data_master.npy')
trainX = trainX_master[:,:,:,0] # The first three dimensions of trainX_master contain image data
trainY = trainX_master[:,0,0,0] # The final column on trainX_master contains labels
print('trainX shape: '+str(trainX.shape))
print('trainY shape: '+str(trainY.shape))
testX_master = np.load(r'C:\Users\Alex Koch\Documents\Python\Research\Machine
Learning\CV_Data\Merger\Ind_feats\Tidal\Images\Testing\Testing_data_master.npy')
testX = testX_master[:,:,:,0]
testY = testX_master[:,0,0,0]
print('testX shape: '+str(testX.shape))
print('testY shape: '+str(testY.shape))
# Reshape training and testing image data to be 1 channel deep
trainX = trainX.reshape((trainX.shape[0],50,50,1))
testX = testX.reshape((testX.shape[0],50,50,1))
# One hot encode target labels
trainY = to_categorical(trainY)
testY = to_categorical(testY)
# Normalize training/testing image data (range 0-1)
trainX = trainX - trainX.min()
```

```
trainX = trainX/trainX.max()
testX = testX - testX.min()
testX = testX/testX.max()
# Define Model (Dieleman et al 2015)
model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', kernel_initializer='he_uniform',
input_shape=(50,50,1)))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(64, (5,5), activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())
model.add(Dense(2048, activation='sigmoid'))  # Sigmoid or Softmax
model.add(Dense(2048, activation='relu'))
model.add(Dense(2, activation='sigmoid'))  # Sigmoid or Softmax
model.summary()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Create directory for Results
X = str(datetime.now())
X = X.replace(' ', '_')
X = X.replace(':','_')
X = X[:-7]
path = r'C:\Users\Alex Koch\Documents\Python\Research\Machine
Learning\CV_Data\Merger\Ind_feats\Merge\Results\^'+X
path = path.replace('^','')
print(path)
try:
    os.mkdir(path)
except OSError:
    print('Creation of directory %s failed' % path)
else:
    print('Creation of directory %s successful!' % path)
esm = ('val_loss', 'min')
cpm = ('val_accuracy', 'max')
es = EarlyStopping(monitor=esm[0], mode=esm[1], verbose=1, patience=10)
mc = ModelCheckpoint(path+'\\best_model.h5', monitor=cpm[0], mode=cpm[1],
save_best_only=True, verbose=1)
history = model.fit(trainX, trainY, validation_data=(testX,testY), epochs=4000, batch_size=32,
verbose=1, callbacks=[es,mc])
# Reload the best model for evaluation
saved_model = load_model(path+'\\best_model.h5')
loss, acc = saved_model.evaluate(testX, testY, verbose=0)
print('Loss: ',loss, '\nAccuracy: ', acc)
pred_probs = saved_model.predict(testX)
```

```
# sklearn.metrics.roc_curve() does is not compatible with One Hot Encoded probabilities
# We need to generate an array where each prediction is between 0 and 1
# Values closer to 0 indicate nonmerger prediction, values closer to 1 indicate merger prediction
preds = np.zeros(len(pred_probs))
for i in range(len(pred_probs)):
    if pred_probs[i,0] > pred_probs[i,1]:
        preds[i] = 1-pred_probs[i,0]
    else:
        preds[i] = pred_probs[i,1]
# Calculate False Positive Rate (fpr), True Positive Rate (tpr), and threshodl
fpr, tpr, thresholds = roc_curve(np.argmax(testY, axis=1), preds)
from sklearn.metrics import auc
auc = auc(fpr, tpr)
# Plot a Receiver Operating Characteristic (ROC) curve
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Keras (area = {:.3f})'.format(auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.savefig(path+'\\roc.png', dpi=200, bbox_inches = "tight")
plt.show()
# Zoom in view of the upper left corner.
plt.figure(2)
plt.xlim(0, 0.3)
plt.ylim(0.7, 1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Keras (area = {:.3f})'.format(auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve (zoomed in at top left)')
plt.legend(loc='best')
plt.savefig(path+'\\roc_zoom.png', dpi=200, bbox_inches = "tight")
plt.show()
z = np.argmax(pred_probs, axis=-1)
z.shape
cm = confusion_matrix(y_true=np.argmax(testY, axis=-1), y_pred=np.argmax(pred_probs,
axis=-1))

# Code adapted from https://deeplizard.com/learn/video/km7pxKy4UHU
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
```

```
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            verticalalignment="center",
            color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
cm_plot_labels = ['Nonmerger','Merger']
cm_plot = plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
plt.savefig(path+'\\confusionmatrix.png', dpi=200, bbox_inches = "tight")

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.vlines(11,0,0.5, linestyles='dashed')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.savefig(path+'\\loss.png', dpi=200, bbox_inches = "tight")
plt.show()

plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.vlines(11,0,1,linestyles='dashed')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig(path+'\\accuracy.png', dpi=200, bbox_inches = "tight")
plt.show()
```

```
test_labels = np.argmax(testY, axis=-1)
# Create directories in Results for correctly/incorrectly classified galaxy images
misclasspath = path+'\\Misclassified'
mergerpath = path+'\\Mergers'
nonmergerpath = path+'\\Nonmergers'
print('Misclassified Path: '+misclasspath)
try:
    os.mkdir(misclasspath)
except OSError:
    print('Creation of directory %s failed' % misclasspath)
else:
    print('Creation of directory %s successful!' % misclasspath)
try:
    os.mkdir(mergerpath)
except OSError:
    print('Creation of directory %s failed' % mergerpath)
else:
    print('Creation of directory %s successful!' % mergerpath)
try:
    os.mkdir(nonmergerpath)
except OSError:
    print('Creation of directory %s failed' % nonmergerpath)
else:
    print('Creation of directory %s successful!' % nonmergerpath)
# Create .png images of correctly/incorrectly classified galaxies
for i in range(len(test_labels)):
    if test_labels[i] != np.argmax(pred_probs, axis=-1)[i]:
        impath = misclasspath
        if test_labels[i] == 0:
            trueclass = 'Nonmerger'
            predictedclass = 'Merger'
        else:
            trueclass = 'Merger'
            predictedclass = 'Nonmerger'
    else:
        if test_labels[i] == 0:
            impath = nonmergerpath
            trueclass = 'Nonmerger'
            predictedclass = 'Nonmerger'
        else:
            impath = mergerpath
            trueclass = 'Merger'
            predictedclass = 'Merger'
    im, norm = imshow_norm(testX[i,:,:,0], origin='lower', cmap='gray')
    plt.title('Galaxy '+str(i))
    plt.figtext(0.8, 0.5,'\nTrue Class: '+trueclass+'\nPredicted Class: '+predictedclass)
```

```
    plt.subplots_adjust(right=0.8)
    plt.axis('off')
    plt.savefig(impath+'\\'+str(i)+'.png', dpi=200, bbox_inches = "tight")
    plt.show()
print('Finished!')
```

## BIOGRAPHY OF THE AUTHOR

Alex Koch was born and raised in Indianapolis, IN, and graduated from Bishop Chatard High School in 2010. He attended Purdue University, West Lafayette, and earned a Bachelor of Science in Physics and a Minor in Astronomy in 2014 and went on to work for as a contracted computer specialist for the U.S. Department of Defense until 2016. He is a candidate for a Doctor of Philosophy in Physics from the University of Maine in August 2021.