

Spring 5-20-2023

Using Deep Neural Networks to Classify Astronomical Images

Andrew D. Macpherson
Seattle Pacific University

Follow this and additional works at: <https://digitalcommons.spu.edu/honorsprojects>



Part of the [Analysis Commons](#), [Artificial Intelligence and Robotics Commons](#), [External Galaxies Commons](#), [Other Astrophysics and Astronomy Commons](#), and the [Other Physics Commons](#)

Recommended Citation

Macpherson, Andrew D., "Using Deep Neural Networks to Classify Astronomical Images" (2023). *Honors Projects*. 181.

<https://digitalcommons.spu.edu/honorsprojects/181>

This Honors Project is brought to you for free and open access by the University Scholars at Digital Commons @ SPU. It has been accepted for inclusion in Honors Projects by an authorized administrator of Digital Commons @ SPU.

USING DEEP NEURAL NETWORKS TO CLASSIFY ASTRONOMICAL IMAGES

by

ANDREW MACPEHRSON

FACULTY MENTORS:

DR. LISA GOODHEW, DR. JOHN LINDBERG, DR. DENNIS VICKERS

HONORS PROGRAM DIRECTOR:

DR. CHRISTINE CHANEY

A project submitted in partial fulfillment of the requirements
for the Bachelor of Arts degree in Honors Liberal Arts
Seattle Pacific University
2023

Presented at the SPU Honors Research Symposium
Date May 20, 2023

Using Deep Neural Networks to Classify Astronomical Images

Andrew Macpherson
Departments of Physics, Computer Science
Seattle Pacific University
Seattle, WA
macphersona@spu.edu

Abstract—As the quantity of astronomical data available continues to exceed the resources available for analysis, recent advances in artificial intelligence encourage the development of automated classification tools. This paper lays out a framework for constructing a deep neural network capable of classifying individual astronomical images by describing techniques to extract and label these objects from large images.

Keywords—astronomy, deep neural networks, artificial intelligence, machine learning, classification

I. INTRODUCTION

In the present age of Big Data, the quantity of information available has surpassed an amount even the most dedicated analysts can comprehend.¹ The challenge of data analysis is no longer grappling with too little data but has transformed into the problem of sense-making amidst a sea of information. As the field of astrophysics continues to grow – and the technologies it uses with it – so too has a chasm between the quantity of data produced and the amount scientists can process. Zooniverse, an astronomical citizen science site, describes “the major challenge of the 21st century” as “dealing with the flood of information we can now collect about the world around us [2].”

We are taking more photos of the sky than ever before, yet we no longer have the resources to always know what’s in them. Amidst a backdrop of thousands of stars, only careful analysis may be able to distinguish a similar object from those around it. Challenges such as these, amidst an ever-growing influx of data, implore the development of more powerful image processing tools for the field of astrophysics.

Astrophysics research has a rich history of collaboration and thrives when its dedicated community of amateurs contributes to outstanding tasks. Citizen science sites have filled large gaps in knowledge (particularly in classification) and accelerate the understanding of the field as a whole. They allow researchers to “analyze their information

more quickly and accurately than would otherwise be possible, saving time and resources...leading to faster progress and understanding of the world, getting to exciting results more quickly [2].”

The rapidly expanding field of artificial intelligence (AI) provides an exciting new frontier for professional and citizen scientists alike. New technologies like GPT-4 [3] are ushering in a new wave of AI technologies and their ever-growing roles in everyday life demonstrate unexplored scientific potential for astrophysics.

Presently, astronomical machine learning models do exist but have seen limited capabilities. Existing tools are predominantly only available to professional research scientists at large institutions and their limited deployments have not yet caught up with the new wave of AI technologies.

This gap in research tools – especially those available to the amateur astronomer – and the limited capabilities available even to the most resourceful implore the development of modern classification AIs in the astronomical disciplines. In this paper, I have striven to create a framework for machine learning algorithms capable of handling dense astronomical image data and categorizing the objects within them.

II. THEORETICAL BACKGROUND

A. What are Astronomical Images?

When analyzing astronomical images (any image of outer space), it is useful for scientists to have more data available to them than a photo from a typical camera would provide. Everyday images are often stored in formats like JPEG and PNG, but astronomers prefer their own file type: the Flexible Image Transport System (aka a FITS file) [4]. Now the standard for astronomical imaging, the FITS format has been endorsed by NASA and the International Astronomical Union [5] and can store large amounts of data in various formats, including multi-dimensional arrays, tables, and header information, all

¹ Statista, a leading consumer data site, estimates that by the end of 2023, the world will have “captured, copied, and consumed” over

120 zettabytes (120 billion trillion bytes, or 1.2×10^{21} bytes) of information [1].

of which can combine to give astronomers a better idea of what they are looking at.

However, that additional data makes FITS files hard to visualize and requires more work than a typical photo. For example, FITS images often contain objects “photographed” with non-visible light – i.e., colors outside what we can see,² which scientists need to turn into an image we can comprehend. Nonetheless, FITS files are incredibly useful because a single image can contain hundreds or thousands of objects in its field of view. Distinguishing these objects from one another and extracting them individually is at the heart of this project and of immense value to automating any classification process.

B. What is AI?

The term artificial intelligence (AI) is becoming increasingly common in everyday life and yet can mean different things depending on the context.

François Chollet succinctly defines artificial intelligence as “the effort to automate individual tasks normally performed by humans [7].” Though AI fears are on the rise as we enter a new wave of AI development, this is not an inherently bad thing and is actually responsible for many technologies we rely on today. AI technologies are also responsible for contributing to major world history events (such as the British breaking Nazi codes to gain a tactical advantage [8]) and are used for everything from media content curation to detecting credit card fraud.

As with any technology, alongside every beneficial application, there is real cause for concern. Yet some AI technologies ought to elicit more (or less) concern than others. There are many different types of AI, and so it is often best to refer to AIs by what sort of *intelligence* they possess. This paper is far from a treatise on all the different types of AI, but it is worth briefly exploring what certain AIs are and are not.

C. Generative AI

The popular image of a self-aware AI often found in futuristic doomsday scenarios is referred to by scientists as Artificial General Intelligence (AGI). Shevlin et al. astutely noted in 2019 that “at a time when headline-making AI breakthroughs are an

almost daily occurrence, it might seem that we are on the cusp of living with artificial systems that match or exceed human intelligence [9].” Recent advances in AI technology like ChatGPT [10] and Dall·E Mini [11] may make this feel like more of a reality than ever, but it is still a long way from true. AGI does not exist and likely will not for quite some time, if at all.³ These programs are instead classified as generative AI, a type of artificial intelligence capable of producing original⁴ content.

Generative AI technology certainly has many risks associated with it, and the dangers of technologies capable of presenting information as true and producing unique images based on user input are not to be underestimated. They can hurt people, either through intentional misuse (e.g., deepfakes, which can produce audio or video “recordings” of someone saying or doing things that never happened⁵), or unintentional failings (e.g., ChatGPT returning information that is untrue or that infringes on somebody’s intellectual property).

Nonetheless, these programs are not capable of *learning* in the sense that humans are [9]. They are entirely reliant on their input (which is exactly why they are prone to misuse) and require training for specific tasks on enormous datasets. Even broad tools like ChatGPT do not *know* things in the same way humans do; it is essentially a fancy search engine that can recognize and output natural language. These technologies do not have thoughts or desires, even if they know how to say that they do [12].

D. Classification AI

This project focuses on the other primary type of AI in use today: classification AI.⁶ This technology is exactly what it sounds like, a type of artificial intelligence capable of recognizing images and putting them into a defined category. It should be apparent that there are significantly fewer risks associated with this type of technology and though misuses of this technology are certainly possible they are less apparent and of minor concern here where the worst-case scenario is an incorrect scientific label and systemic bias discriminates against groups of inanimate objects not whole groups of people.

² What we call visible light is an electromagnetic wave but most of these waves are at frequencies the human eye cannot detect. For more, see [6].

³ It is rarely a good idea to make predictions about how technologies will (or more often - will not) develop but it is a widespread consensus and the author’s personal belief that AGI is a long, long ways off.

⁴ How original this content truly is has sparked numerous ethical and artistic debates but it is sufficient to say here that much of the content has at least never been made in the format the AI produce.

⁵ Deepfakes are certainly cause for concern and presently one of the main dangers in AI usage. However, unless you are a politician or have an excessive amount of images of yourself on the Internet, you likely need not be personally worried just yet, though it may be worth revisiting old Facebook content.

⁶ Various authors will define differing numbers of types of artificial intelligence. I have chosen this dichotomy based on presently available technologies.

E. What is machine learning?

In its broader definition, image classification is not inherently a technology-dependent process. Each of us classifies thousands of things every single day as we receive images from our brains and put objects into categories that we recognize. For example, if your neighbor introduces you to their dog, you do not need them to tell you that it is a dog. Even if you haven't seen your neighbor's dog before (or even that dog's breed), you can pretty confidently recognize what a dog is, so much so that it is not something we are often aware of (think of the last time you had to ponder what species of pet your neighbor has). However, sometimes we also see things that don't fit into a box in our heads – perhaps on a vacation you've encountered native wildlife and wondered what species it belonged to. Eventually, you likely learned its name, saw other examples, and became capable of recognizing other animals like it.

Essentially, this is what we are asking a computer to do when we talk about automated image classification. The computer has never *seen* a dog before, but by showing it pictures of dogs, it eventually starts to recognize what a dog looks like and, like you, will know that the neighbor's dog is a dog without needing to be told explicitly. What you did on your vacation when you saw a new animal, the computer did as well. The difference lies in how you and the computer both went about the process of learning what a new animal looks like. For the computer, this process is called machine learning, a sub-field of artificial intelligence.

When learning what a new thing is, the human brain performs differently than computers. If asked to describe a dog, you might point out its paws, four legs, and tail, but you could also easily distinguish it from a cat with those same features. Evidently, there is something intrinsic about how we distinguish what makes a dog a dog.⁷ Describing exactly what a dog is proves more challenging than you may think. Any description needs to keep the description broad enough to include all sizes and types of dogs (a Chihuahua and a Husky certainly aren't very similar) but also distinct enough to distinguish non-dogs from dogs every time. It also needs to include dogs who may be missing characteristic features; a dog without a tail isn't any less of a dog after all! It may be easy for humans to tell a dog from a cat but for a computer without any prior

knowledge of these things, distinguishing the two may prove difficult.

Further, the odds are good that you've met at least a few dogs in your life. Does the task of describing a dog become easier or more difficult if you've only ever seen photos? The question brings up another limitation of computer recognition and classification: inferences. Humans are capable of making various assumptions when looking at things that we can take for granted. If in a photo, you can't see all 4 of a dog's legs (perhaps because they are covered up or out of frame), you don't immediately wonder what happened to them; your brain can typically infer that the dog still has 4 legs and is capable of recognizing the animal as a dog. A computer though may have been taught that dogs have 4 legs and upon seeing only 2, concludes that the animal must not be a dog.



Fig. 1: A two-legged animal, according to an untrained computer.

Used with permission of Grace Macpherson.

Thankfully, computers use different techniques to figure out what's an image (described later in the Machine Learning section). It may seem frustrating at first that a computer is not capable of distinguishing objects in the same fashion humans are, but it quickly becomes helpful to use quantifiable methods rather than trying to describe what exactly constitutes a dog.

F. Understanding Deep Learning

Image classification falls under the broad field of computer vision, a subset of machine learning that is rapidly expanding and which has applications in everything from searching Google Images to self-driving cars and facial recognition systems.

⁷ This is meant in the context of the difficulty of describing our implicit ability to recognize, not the essence of a Platonic form or other philosophical construct.

Nearly all computer vision models now use a technique called deep learning, which utilizes a structure modeled after the human brain: a neural network. A neural network consists of various layers, each of which acts as a sort of filter to distill the input into something more useful to the task at hand. Layers are made up of a series of neurons which each perform individual operations that combine to complete the larger task of the layer. The name deep learning refers to a model’s layer depth, i.e., how many layers there are [7, p. 7] in the neural network, and models that utilize this technique (including the one presented here) are known as deep neural networks (DNNs).

III. ASTROMETRY

A. About

For training data, I used labeled images from astrometry.net (Astrometry), an online platform for users to share astronomical images that contains millions of high-quality, labeled images available in FITS format [13].

B. Catalogs & Object Annotations

All labels correspond to objects found within the Abell, Henry Draper, Tycho-2, Hipparcos, NGC/IC, and Yale Bright star catalogs [14], which contain an approximate 2.9 million combined objects [15]–[19], each consisting of different types of objects. Astrometry’s annotation feature checks each catalog based on the radius of the object in the image as described in Table 1.

Catalog	Object Type (Size)	Radius value (pixels)
Abell	Galaxy clusters (4073)	< 1
Henry Draper	Stars (359,083)	< 1
Tycho-2/Hipparcos ⁸	Stars (2.5 million)	< 0.25
NGC/IC	Varies (13,226) ⁹	< 10
Yale Bright Star	Stars ¹⁰ (9,110)	< 10

Table 1: Astrometry catalog parameterization

⁸ Note that Tycho-2 is an extension of the Hipparcos catalog and so they have been treated as one catalog here.

⁹ To be specific, the NGC/IC breaks categorizes every object as a Diffuse Nebula or Supernova Remnant, Planetary Nebula, Open

Since most objects in an image are not found in a catalog, each astronomical image only contains a handful of objects identifiable to the machine for testing and training purposes.

The discrepancy between the number of recognizable and labeled objects is easily made apparent by comparing Astrometry’s multiple views for an image. Figures 3 and 4 show the same image in two different views, the red-green view – which circles all prominent objects in each photo – and the annotated view – which circles recognized catalog objects.

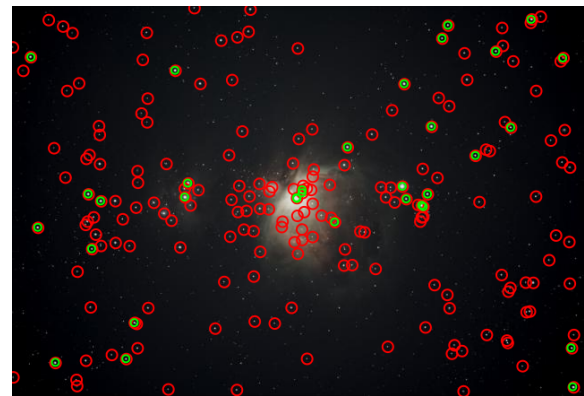


Fig. 2: Astrometry’s red-green view [21]

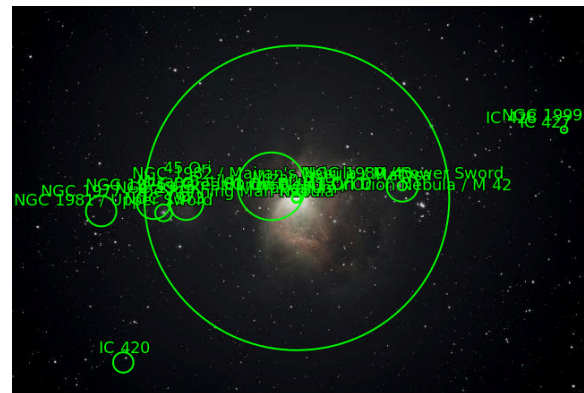


Fig. 3: Astrometry’s annotated view [22]

C. Utilization Percentage

To capture an idea of the percentage of objects Astrometry has annotations for, I asked OpenAI’s ChatGPT to write me a program that could identify the

Cluster, Globular Cluster, Part of other object, Duplication, or Not Found as described by Frommert [20].

¹⁰ The YBS actually only contains 9,095 stars with the remaining 15 objects consisting of novae/supernovae (11), globular clusters (2), and open clusters (2).

number of red circles (since green circles double count) in the red-green view of the image and divided it by the number of objects returned by the call to the API. I refer to this here as the utilization percentage. In this particular example, the programs identified 124 red circles and 18 catalog images, a utilization percentage of 15%. However, these values should be treated as a rough estimate and an upper bound since the calculation assumes that all catalog images are identified in the red-green view, which we can see in this example is not always the case.

In fact, there are even more objects visible that are not distinguished in either view. However, these objects tend to be very faint and thus not particularly useful for machine learning purposes. A view of all these objects can be seen in Figure 5, the image seen in Astronomy’s extraction view.

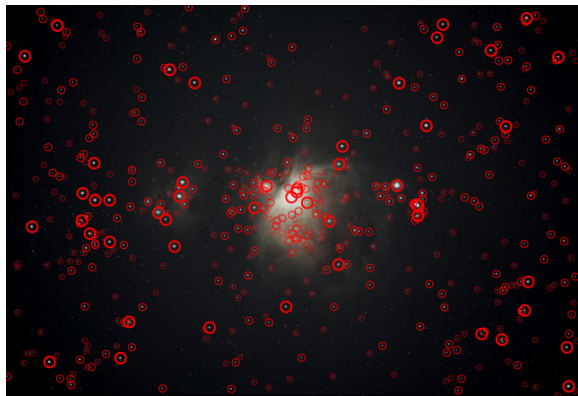


Fig. 4: Astrometry’s extraction view [23]

Identifying the commonalities between the two (useful) sets can prove challenging, especially visually as labels overlap (try and count the green circles) and in turn makes finding the exact percentage of cataloged vs identifiable images difficult to calculate. That said the average utilization percentage can be used to find a lower limit on the number of source images needed and should be considered in any further exploration of model limitations. The full program for finding the utilization percentage is shown in Appendix A as well as my conversation with ChatGPT to generate and modify it in Appendix B.

IV. SOURCE EXTRACTOR

A. About

The other tool used for preprocessing was Source Extractor. Source Extractor (often abbreviated to SExtractor) is an incredibly powerful tool for processing astronomical data. It is capable of identifying and extracting individual objects in a large astronomical image and outputting specific data

corresponding to each of them, which I used in combination with Astrometry data to produce training images. For a more comprehensive guide to Source Extractor’s capabilities, I highly recommend Dr. Benne Holwerda’s *Source Extractor for Dummies* [24].

B. Installation

Since Source Extractor lacks any sort of GUI, it is more complex to install than a normal program that is usually downloadable from the Internet. Detailed steps for installing and using Source Extractor are found elsewhere, but since I worked on a MacBook, I installed Source Extractor using Homebrew [25] a powerful package manager for MacOS. The full software can be found on GitHub [26] and I imported the configuration folder directly from the repository to ensure its accuracy.

C. Configuration

Within the configuration files, there are two that required modification, and despite their unconventional file extensions, they are easily modified in a basic text editor.

In `default.sex`, the `CATALOG_NAME` parameter was set to `outputs.fits`. Note that the actual file name is irrelevant so long as it ends with a `.fits` extension. The `CATALOG_TYPE` parameter should then be set to `FITS_1.0`.

The `default.param` file contains all of the parameter options for running SExtractor, but by default, they are all commented out using the `#` symbol. `X_MIN`, `X_MAX`, `Y_MIN`, and `Y_MAX` should all be uncommented.

To run Source Extractor on a given image, navigate in the terminal to the configuration folder. From there, simply run `sex filename.fits` in Bash, substituting as appropriate. I stored my data in a folder residing in the same directory as the configuration folder, and thus ran the command as `sex ../Data/filename.fits`.

V. COMBINING MATCHES

A. Astrometry API

With Source Extractor installed, I return momentarily to the Astrometry images. Astrometry provides an API [14], [27] capable of returning JSON data (quickly converted with a simple parse [28]) with various information about known objects in the image, including each object’s name(s), relevant catalog, location in the image (in x- and y-coordinates of pixels), and radius (size within the image).


```

{"annotations": [{"type": "bright",
"names": ["15 Mon"], "pixelx":
2055.177912132401, "pixely":
2618.5783134857497, "radius": 0.0,
"vmag": 4.659999847412109},
{"type": "ngc", "names": ["NGC
2259"], "pixelx": 449.805278967448,
"pixely": 4818.564763317862,
"radius": 115.41987644560295},
{"type": "ngc", "names": ["NGC
2264"], "pixelx":
2051.9800344569444, "pixely":
2617.751152553472, "radius":
219.29775922703007}]}

```

Submission ID: 7301665

Job ID: 8042724

Figure 5: Sample Astrometry API output

However, these data do not contain photometric information, which of course is the ultimate goal of the program.

This is where Source Extractor comes into play. As previously mentioned, Source Extractor has hundreds of programmable parameters but here I made use of 4 per object: X_MIN, X_MAX, Y_MIN, and Y_MAX, corresponding to the object’s positional data (within the image).

Source Extractor can be easily processed using the tools of the AstroPy Python library [29]–[31]. AstroPy can read the output file of Source Extractor and transform it into a table whose values can be easily read by Python.

At this point, I combine the Astrometry and Source Extractor data to isolate individual objects while retaining their classification information. Since both programs return the position and radius of each object in the image, the overlap between the two data sets should yield a collection of objects with two different types of information. Yet Astrometry and SExtractor define each object’s position and radius differently. This is why Source Extractor is used at all – because of its ability to precisely carve out objects and provide information on their photometric data. Astrometry defines each object with a one-dimensional radius (simply a circle) and may define the position of the object differently than SExtractor which bounds the object rectangularly and more precisely.

To find the overlap in the sets, I compared each possible pair of objects after filtering out all objects with a radius of 0. First I checked if the radius of the Astrometry object was roughly the same size as the “y-radius” ($\frac{1}{2}$ the length in the y-dimension) of the Source Extractor radius (I chose y because most images are oriented landscape and there is a slightly larger percentage of objects with larger x-dimensions; the two can be easily swapped however).

If any two objects are roughly the same size, then they must also be located close to one another in the image to be a match. Since Source Extractor and Astrometry often find the same object but give it different coordinates, I made sure to give variation in the position but also ensured there was still overlap between the two regions. Specifically, I defined that their distance must then be within 9/10ths of the length of the smaller radius.

Thus, the three conditions for two objects to be a match are:

$$\min(r_{SE}, r_A) > 0 \quad (1)$$

$$(0.5 * r_{y,SE}) \leq r_A \leq (1.5 * r_{y,SE}) \quad (2)$$

$$\sqrt{(x_{SE} - x_A)^2 + (y_{SE} - y_A)^2} < 0.9 * \min(r_{SE}, r_A) \quad (3)$$

Since varied radii yield a significant amount of overlap, it is a difficult problem to keep a high percentage of images while also ensuring near-perfect accuracy. A machine is only as good as the data it is provided and so these criteria are intentionally defined strictly, preferring far fewer objects but with higher accuracies since it is easier to add more data than to classify with inaccurate data.

B. Processing Matches

Once a match has been found, relevant data from both sources need to be cataloged. For Astrometry, this is a simple matter of pushing the relevant object to an array where each entry corresponds to an object.¹¹

For Source Extractor, we are going to use the data at hand straight away, rather than storing an index and traversing an incredibly large file again or storing whole Table components. Rather, Source Extractor locates and bounds the image, so it makes sense to store the cutout of the object directly.

Here, AstroPy’s Cutout2D method is useful. Given coordinates and dimensions from Source Extractor’s output and raw image data in the form of

¹¹ Note that this could be made more space-efficient by storing only the object’s label since this is the only thing we will extract from the data. However, in the interest of generality, I have kept all

the data since it is presumed that the number of objects identified as matches is relatively small and is therefore not particularly burdensome to the stack.

the original FITS file, it is capable of creating a cutout of the image as a 2-dimensional array [32].

This and the index of the corresponding Astrometry data are then stored in parallel arrays and returned so that each object and both sets of its corresponding data are now easily located.

C. Ascertaining Object Type

Of course, knowing the name of our object doesn't immediately tell us what type of object it is. "NGC 123" hardly provides any clues about whether we are looking at a galaxy or a star. However, this is easily searchable data, which I compiled into CSV files for quick lookup ability in Python (Appendixes C, D, E).¹²

These catalogs provide a fairly large number of object classifications, and for data sets of less than extraordinary volume, it is often helpful to classify more broadly than the catalogs' level of detail. For example, there are 38 different types of galaxy classifications, ranging from specific types of galaxies (*disk galaxy*, *Seyfert galaxy*, *barred lenticular LINER galaxy*) to galaxies whose appearances are obscured by other astronomical objects (*spiral galaxy occulted by another spiral galaxy*, *lenticular galaxy pair*). Referring to them simply as "galaxies" loses a degree of specificity but also makes each result more useful to a machine learning model and allows for more accurate high-level classifications. This is also a useful technique for invalid data types. *Unknown*, *nonexistent*, and *non-existent* all essentially say the same thing and are more usefully summarized with a simple *N/A*. The full subclassification list and their generalizations are found in Appendix F.

D. Combining Labels

At last, herein lies the goal of the exercise: creating a labeled image for preprocessing centered on the relevant object with the tightest dimensions possible. It is a simple matter to generate a plot using the 2D vector that displays the data as an ordinary image. Note that is necessary to choose a color map, which may affect the relative contrast between an object and its surroundings.¹³ I have found the `twilight` mapping to be most useful, but also particularly liked `gist_heat` (see [34] for a full list of mapping options). A quick lookup function will tell us the generalized classification of the object and combining it with the object's Astrometry ID provides an image name that clearly identifies the object and gives it a

unique ID, ensuring no name conflicts will occur. The named cutout image is then easily saved to the local machine and the process can be repeated for each object in the file.

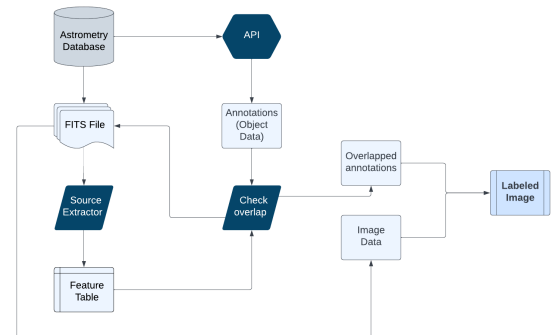


Fig. 5: Summary of the preprocessing dataflow

VI. MACHINE LEARNING

Having extracted and labeled astronomical objects from large-scale FITS files, it is now time to ask the computer to find patterns. As an undergraduate research project, the scope of this project is not to create an accurate model so much as to lay out the technical framework for creating a model capable of classifying images with a high degree of accuracy by developing an initial codebase.

To that end, the code provided for classifying images functions as a starter for those hoping to accomplish this feat. It is largely built off François Chollet's *Deep Learning with Python*, a text that has proved instrumental to this project and which any serious student of machine learning should have readily available.

While rather basic in its application of machine learning principles, the code here nonetheless requires a technical background and commensurate understanding of the field of AI. Chollet steps through his code better than I might, but I hope to outline the basic principles relevant to this application.

A. Process Overview

There are five main steps in any machine learning problem [35]:

1. Data collection
2. Data preparation
3. Training
4. Evaluation

wavelengths outside the visible light spectrum. Thus, there should be no worries about the loss of an object's "true color" when choosing a color mapping.

¹² Since these are CSV files they are not easily attached to this paper as a regular appendix might be. Instead, they can be found at [33].

¹³ It is commonplace within astronomy to recolor an image, even at the collection level since many photographs are mappings of

5. Tuning

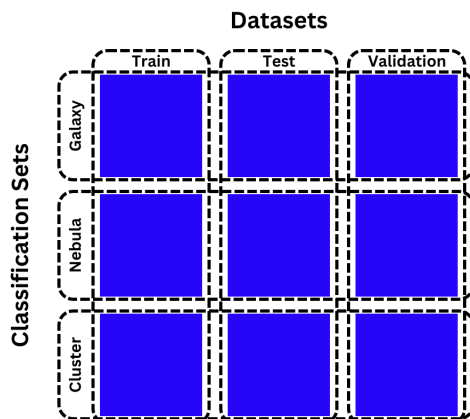
Perhaps the most difficult of these steps in our current application is the first step. Having accomplished this in the previous sections, we now want to prepare our data in a format optimized for deep learning.

B. Data Preparation

Our first step is telling the computer which things we want to classify, to tell it what its options are. Practically, this means sorting our images into folders corresponding to object type (these are our categories). Since we have already named our images accordingly, this should be a relatively easy exercise.

Machine learning models also require their data to be split into three categories: training, testing, and validation. The basic workflow of programming any machine learning algorithm (regardless of what type it is) is to train it, test it, and validate its accuracy. Training data is the most important of the sets and is used to teach the model the features of what it is trying to classify. The validation data is for evaluating the model's performance – giving it a test and checking its responses against an answer key [7, p. 133]. The test data is then used for one last evaluation of performance; it is separate from the validation data to identify potential overfitting (training the model to be too accurate for a specific dataset with a loss of generality) in the previous two sets.¹⁴ Research suggests that roughly 70-80% of data should be used as training data [36], with the rest being used to evaluate the model via testing and validation.

These splits can be modeled as cutting across the data both horizontally and vertically, as pictured in Figure 6.



¹⁴ The definition and usage of the validation set is not very consistent across machine learning applications (here I have used Chollet's definitions). However, the validation and testing sets are

Fig. 6: Splitting train/test/validation data by type of astronomical object

The other, larger, challenge in data preparation is converting the images to an input format the machine can actually use. We will accomplish both these tasks in the same command, using Keras' `image_dataset_from_directory()` function. Given properly sorted directories (train, test, validate folders each with a folder per classification set), this will apportion the data into the various sets for the model and convert the image into batched tensors of a common size based on RGB data from our standard image format [7, p. 217].

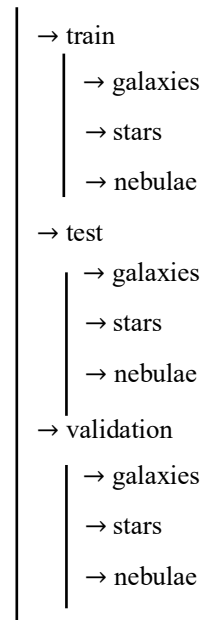


Fig. 7: Sample directory

1) *Tensor Conversion*: While the computer converts the images automatically, it is worth understanding what is happening under the hood. Every color can be modeled as a combination of red, green, and blue. The standard format is to specify the amount of each of these colors needed on a 0-255 scale. The appropriate combinations of each can then display any color (this is how your computer screen works). For each pixel then, `image_dataset_from_directory()` specifies the color as a combination of RGB values in a quantifiable form. These will each be stored as a vector with 3 channels (values) in the format `[R, G, B]`. The pixel's position in the image is then stored by

conceptually very similar, so data splitting is often referred to only as a dichotomy of testing and training.

row and column, where each pixel is stored as an entry in another vector representing the row. Each row is also a vector and so the image data can be easily accessed where the index of the pixel within the row corresponds to the column and the index of the vector containing the pixels corresponds to the row. Thus, each pixel can be accessed with two indices and individual channel values can be accessed as the internal values of a three-dimensional vector (for example, `pixel[0][0][0]` would access the value of red of the first pixel in the first row).

2) *Data Augmentation*: One of the biggest challenges in creating a machine learning model is supplying it with sufficient data. When working with small datasets, we can mitigate this problem by artificially inflating the number of images for the model to work on. Data augmentation can distort the original images and give them back to the model as new images to work on. This technique not only increases the size of your training set but also trains the model for non-standardized data. This is of less concern for astronomical images which do not necessarily have a proper orientation or zoom level but it is still useful.¹⁵ To accomplish this in this context, the images are flipped both horizontally and vertically, randomly rotated by a random value within $\pm 10\%$, and zoomed in or out by a random factor of $\pm 20\%$.

C. Training

Training the neural network is the step requiring the most amount of care from a technological perspective. Here we have followed Chollet's example of a very simple convolution network [7, p. 223]. After augmenting the data, the model first adjusts the RGB values by dividing each by 255, which results in every value being in the range [0, 1], a far more regular scale than [0, 255] which is derived from binary storage limitations and arbitrary to the task at hand.

Thus far we have only converted the format of our images and rescaled them to be more accessible to the computer. It is now time to manipulate them with the layers characteristic of a deep neural network.

1) *Convolution Layers*: The first key layer we apply is a Conv2D layer (aka a convolution layer). Each convolution layer learns local patterns derived from small subsets of the image. According to Chollet, convolution layers have two key characteristics: first, "the patterns they learn are translation invariant" – meaning they can recognize the pattern anywhere in

the image; and second, "they can learn spatial hierarchies of patterns" – meaning deeper convolution layers can recognize patterns of earlier layers [7, p. 205].

The importance of this second property is not to be understated. When I described earlier the different ways in which humans and computers recognize objects, I suggested that in describing what makes a dog a dog, you might point out features like its paws, legs, and tail. In fact, the way we recognize things may not be that different from computers as this is what hierarchical convolution layers are doing. Continuing with the example of recognizing a dog, shallow convolution layers (those coming first) might learn what a dog's nose looks like by recognizing patterns that appear consistently in dogs' noses. Over time, it learns all sorts of these features and recognizes that when put together they form a dog.

2) *Max Pooling*: The other primary type of layer we work with here is MaxPooling2D (aka max pooling). As Chollet describes, "max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel. [7, p. 209]" This results in "aggressive downsampling", i.e., reducing the number of parameters in a layer [7, p. 209]. This is important because as our neural network grows, the number of parameters the computer has to handle can quickly grow out of proportion. Hence, (almost) every time we apply a convolution layer, it is necessary to max pool the resulting feature map. By taking the maximum value of each channel, the max pooling carries only the most prominent features of a layer to the next level which functionally removes background noise. This has the double effect of widening the view of the convolution layers since they will now pay attention to wider patterns – particularly of lower convolution layers – while processing the same amount of data. Of course, any data loss process should be treated with caution, since perhaps the noise being removed is instead essential data, but this data removal is what allows for high-level pattern recognition.

Essentially, hierarchical convolution layers allow the computer to recognize patterns at the minute level while also looking at the image as a whole. Max pooling layers allow for that wider view of the picture by extracting the most important features from each layer and discarding the rest.

¹⁵ A more traditional use case for augmentation is trying to recognize objects where input images are always right side up...the dog is still a dog even if it's upside down.

3) Other Layers

a) *Flatten*: Before our model is ready for deployment there are a few layers left to configure. After our final convolution layer, we have an output with a shape of (7, 7, 256), a 7 x 7 feature map with 256 channels each. Flatten condenses this output to a 1-dimensional vector to pass to the classifier (remember: so far we have prepared the image for classification in the way we think is best for the computer, but we have not yet classified it) [7, p. 203].

b) *Dropout*: The penultimate step is the Dropout layer. This is our last line of defense against overfitting. Developed by Geoff Hinton at the University of Toronto, the idea of dropout “is to randomly drop units (along with their connections) from the neural network during training [37].” Despite being a fairly simple technique, it has shown incredible results and will make our model more accurate.

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[(None, 180, 180, 3)]	0
sequential_4 (Sequential)	(None, 180, 180, 3)	0
rescaling_6 (Rescaling)	(None, 180, 180, 3)	0
conv2d_31 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_24 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_32 (Conv2D)	(None, 87, 87, 64)	18496
max_pooling2d_25 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_33 (Conv2D)	(None, 41, 41, 128)	73856
max_pooling2d_26 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_34 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_27 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_35 (Conv2D)	(None, 7, 7, 256)	590080
flatten_6 (Flatten)	(None, 12544)	0
dropout_3 (Dropout)	(None, 12544)	0
dense_6 (Dense)	(None, 1)	12545

=====
 Total params: 991,041
 Trainable params: 991,041
 Non-trainable params: 0

Fig. 8: the model layers (as shown by model.summary())

c) *Dense*: The last layer in the DNN is where the magic happens: this is where our images are actually classified. The Dense layer is not just one layer but instead a series of layers that derive their name from being densely connected. It takes two arguments that are of note here: first the number of classification

categories we plan on having (3 in this example); and second, the activation function, a mathematical vector that tells the neural network what we’re asking it to do [38], [39]. We have actually been using activation functions all along with the activation=“relu” argument we applied to each of our convolution layers. Since this is a multi-class classification problem (determining which of multiple classes an object belongs to rather than sorting it into one of two categories), softmax is best suited for this task. The softmax activation asks the model to give a probability distribution describing the likelihood that the object in question belongs to each of the applicable categories [7, p. 108]. Mathematically, softmax is defined as\

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } (z_1, \dots, z_K) \in \mathbb{R}^K \quad (4)$$

for an input vector \mathbf{z} of K real numbers [40].

d) *Sequential*: You may also notice that at the top of our neural network (as seen in Fig. 8), there is a Sequential layer. This is essentially telling the DNN to chain the layers we give it together sequentially and is standard across deep neural networks [7, p. 63].

D. Evaluation

Of course, the reason we feed the model labeled data is so we can track how well it can come up with labels of its own; creating a model without knowing how well it performs is next to useless. Additionally, we’re going to be updating our model a lot in an attempt to make it better, but it’s not always going to work. To that end, we want to make sure we update our model only when it gets more accurate, which we define as having the least amount of loss in our validation set. To do so, we use the following piece of code from Chollet [7, p. 219-220]:¹⁶

```
callbacks = [
keras.callbacks.ModelCheckpoint(
    filepath="convnet_from_scratch.keras",
    save_best_only=True,
    monitor="val_loss")
]
```

¹⁶ As of writing this actually produces an error related to while loops. It appears this is a bug in Keras [41] and does not affect the functionality of the program.

```

history = model.fit(
    train_dataset,
    epochs=30,

validation_data=validation_dataset,
    callbacks=callbacks)

```

Finally, we can evaluate the accuracy of the model over time with our preferred metric (here we use validation set loss) and compare the accuracies at each stage to potentially alert us to overfitting.

E. Tuning

In all likelihood, the first version of a model that we try out is not going to be the best one. Different techniques work better for different datasets, and exploring the use of various layers is a great starting place for future researchers. We have already built in a functionality to keep only the best of our models, so it is never a bad idea to continuously improve. However, one may eventually reach a maximum generalizable accuracy with the data they have. It is worth mentioning yet again that overfitting can ruin a model's performance on further data and with a small dataset that threshold of generalizable accuracy may come sooner rather than later. If the threshold is too low, adding more data will raise it higher. Once the desired level of accuracy has been achieved, we are finished!

VII. CODING ENVIRONMENT

It is expected that all results and methodologies will be reproducible and are not dependent on operating system but in the interest of omitting unforeseen dependencies, all operating environments are provided below.

The entirety of this project was conducted on a 2019 MacBook Air running the most current version of MacOS (Ventura 13.0.1 as of writing). The processor is a 1.6 GHz Dual-Core Intel Core i5 and the memory is 8 GB (2133 MHz, LPDDR3).

All coding was done in a Jupyter Notebook environment and written in Python or Bash. Python was the primary language, but Source Extractor utilizes a command line interface and so operating it requires Bash scripting. However, Python allows for inline Bash calls which was how all Source Extractor commands after installation were executed. These are of course operable in the local shell (Command Prompt for Windows, Terminal for MacOS and Linux), but doing so requires running commands individually (one per image) and uploading results into Python before proceeding. This is an obvious

inefficiency and placing the execution commands into Python allows for easy looping and processing without exiting the kernel.

VIII. SOURCING IMAGES

In the interest of first developing terminology and an understanding of applicable tools, I have neglected to discuss how astronomical images were/ought to be sourced. As previously mentioned, all images were downloaded from Astrometry, which hosts large quantities of high-quality images. In truth, there is not much of a methodology for finding images on a systematic level. By default, Astrometry sorts images by the most recent date of upload and this ought to be sufficient in yielding quantities of images to comb through. I am unaware of any systematic bias recency may have on the model since images in FITS format are necessarily of high quality and there have been few major changes that would affect newer images.

However, there are best practices that may be of use in selecting images for model usage. Images that have prominent objects are easy for the model to identify and provide high-resolution data, which often makes a model more accurate.



Fig. 9: A prominent galaxy [42]

Conversely, it is advantageous to avoid overly crowded images. Small objects in large fields are difficult to distinguish from one another, especially when using two programs that may define coordinates and radii differently.



Fig. 10: A crowded image [43]

There are no inherent limitations on which images the model can use – some are just more useful than others. Likewise, the model may not be able to identify less distinguishable objects with the same accuracies as more prominent bodies.

IX. CONCLUSION

We have now accomplished the task we set out to achieve: developed a framework for machine learning in an astronomical context and provided a comprehensive process that takes data from raw FITS format and turns it into classified individual images.

With limited data, there is still much to do and further inquiries to pursue before declaring broader success. Yet the results accomplished thus far are significant to the development of automated classification programs for astronomy & astrophysics. The techniques laid out herein demonstrate that developing a program capable of classifying individual astronomical objects is not only possible but achievable using tools that are freely available to any dedicated user with basic computer access.

X. LIMITATIONS AND FURTHER INQUIRIES

The astronomical community has a rich history of collaboration, and it is my hope that this paper provides a springboard for further inquiries and development. The opportunities for further exploration are numerous but likely begin with processing enough images using the aforementioned techniques to produce a large enough quantity of data to instantiate a working machine learning model.

With this in mind, it is worth reviewing limitations that may arise in future developments.

First, each astronomical image processed with this algorithm yields a comparatively small number of labeled object images, meaning a high quantity of

astronomical images is needed to produce a sufficient data set.

Second, while I have made a good faith attempt to build an accurate machine learning model, it is impossible to determine its usefulness until it has a substantial dataset to work on. I have attempted to explain clearly and in enough detail the tools and processes used to construct the model should further revisions be required after more data have been acquired.

It should also be noted that Source Extractor is an incredibly powerful tool with uses far beyond object recognition. (In fact, it has its own built-in classifier. However, it has not been updated in quite some time and faces significant limitations). It may be possible to use some of these tools to generate more useful data for the deep neural network. Again, I recommend Dr. Benne Holwerda's *Source Extractor for Dummies* [24].

Future inquirers may also wish to tinker with the mathematical conditions imposed to find matches between Astrometry and Source Extractor objects. These were found through repeated experimentation but may be better with different values or by being replaced altogether.

Despite these limitations, we have now produced a framework for

- 1) Extracting color-mapped cutouts of individual objects in FITS files
- 2) Labeling those cutouts by object types via API calls and common astronomical catalogs
- 3) Training a deep neural network to recognize types of objects.

These tools allow scientists both amateur and professional to contribute to the body of knowledge of astronomical images using publicly available data and tools and to advance the understanding of our neighbors near and far in the universe.

APPENDIX A: RED CIRCLE DETECTION – PYTHON IMPLEMENTATION

```
# Load the image
img =
cv2.imread('/Users/andrewmacpherson
/Downloads/7874916.png')

# Convert the image to HSV color
space
hsv = cv2.cvtColor(img,
cv2.COLOR_BGR2HSV)
```

```

import os

image_path =
'/Users/andrewmacpherson/Downloads/
7874916.png'

if os.path.exists(image_path):
    print('Image file found')
else:
    print('Image file not found')

# Display the HSV image
#cv2.imshow('HSV image', hsv)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

# Define the lower and upper bounds
for the red color in HSV
lower_red = np.array([0, 50, 50])
upper_red = np.array([10, 255,
255])
mask1 = cv2.inRange(hsv, lower_red,
upper_red)

lower_red = np.array([170, 50, 50])
upper_red = np.array([180, 255,
255])
mask2 = cv2.inRange(hsv, lower_red,
upper_red)

# Combine the masks to get the
final mask for red color
mask = mask1 + mask2

# Apply morphological operations to
the mask to remove noise
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(mask,
cv2.MORPH_OPEN, kernel)

```

```

# Detect circles in the image
circles = cv2.HoughCircles(opening,
cv2.HOUGH_GRADIENT, dp=1,
minDist=10, param1=5, param2=3,
minRadius=1, maxRadius=50)

# Count the number of circles
if circles is not None:
    num_circles = len(circles[0])
    print("Number of red circles
detected:", num_circles)
else:
    print("No circles detected in
the image.")

```

APPENDIX B: RED CIRCLE DETECTION – CONVERSATION WITH CHATGPT

The following is a conversation with ChatGPT asking it to generate code to detect the number of circles in an image. The only things I changed were adjusting param1 and param2 to values of 5 and 3 from 50 and 30, respectively after ChatGPT recommended that changes could improve results.

Andrew: How could I count the number of items in an image?

ChatGPT: Counting the number of items in an image can be a challenging task, and the best approach depends on the nature of the items you are trying to count. Here are a few possible ways to approach the problem:

1. **Manual counting:** If the items are easily distinguishable and the image is not too large, you can simply count them manually by visually inspecting the image.
2. **Object detection:** If the items are well-defined and distinct from the background, you could use object detection algorithms to identify and count the objects. Popular object detection frameworks include YOLO, Faster R-CNN, and RetinaNet.
3. **Image segmentation:** If the items are overlapping or have irregular shapes, you could use image segmentation algorithms to separate the objects from the background and count them. Popular image segmentation frameworks include U-Net, Mask R-CNN, and DeepLab.

4. Counting by color or texture: If the items have distinct colors or textures that differ from the background, you could use color or texture-based methods to count them. This approach may be less accurate than object detection or image segmentation, but it can be faster and simpler to implement.

It's worth noting that all of these approaches may require some level of training or customization to work well with your specific image dataset.

Andrew: Are you able to count objects in an image?

ChatGPT: As an AI language model, I don't have the capability to count objects in an image on my own. However, there are various computer vision techniques and tools that can be used for object counting in images. These methods typically involve deep learning-based object detection or segmentation models, which can be trained on large datasets of annotated images to learn how to accurately identify and count objects of interest in new images. There are also various pre-trained object detection and segmentation models available that can be fine-tuned or applied directly to new images for object counting.

Andrew: Write me code that will detect the number of red circles in an image of outer space

ChatGPT: Detecting red circles in an image of outer space can be a challenging task, especially since the presence of stars and other celestial objects in the image could make it difficult to distinguish the circles from the background. Nonetheless, here is some sample code that could help you get started with detecting red circles in an image using Python and OpenCV:

```
import cv2
import numpy as np

# Load the image
img = cv2.imread('space_image.jpg')

# Convert the image to HSV color space
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Define the lower and upper bounds
```

```
# Define the lower and upper bounds
for the red color in HSV
lower_red = np.array([0, 50, 50])
upper_red = np.array([10, 255, 255])
mask1 = cv2.inRange(hsv, lower_red,
upper_red)

lower_red = np.array([170, 50, 50])
upper_red = np.array([180, 255,
255])
mask2 = cv2.inRange(hsv, lower_red,
upper_red)

# Combine the masks to get the final
mask for red color
mask = mask1 + mask2

# Apply morphological operations to
the mask to remove noise
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(mask,
cv2.MORPH_OPEN, kernel)

# Detect circles in the image
circles = cv2.HoughCircles(opening,
cv2.HOUGH_GRADIENT, dp=1,
minDist=50, param1=50, param2=30,
minRadius=0, maxRadius=0)

# Count the number of circles
num_circles = len(circles[0])

print("Number of red circles
detected:", num_circles)
```

This code first loads the image and converts it to the HSV color space. It then defines the lower and upper bounds for the red color in HSV and applies a mask to the image to isolate the red regions. Morphological operations are applied to the mask to remove noise, and circles are detected in the resulting image using the Hough circle transform. Finally, the number of circles detected is printed to the console. Note that the parameters used for Hough circle detection may need to be adjusted depending on the specifics of your image.

Andrew: Debug this code for me:

```
import cv2
```

```

import numpy as np

# Load the image
img = cv2.imread('space_image.jpg')

# Convert the image to HSV color space
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Define the lower and upper bounds for the red color in HSV
lower_red = np.array([0, 50, 50])
upper_red = np.array([10, 255, 255])
mask1 = cv2.inRange(hsv, lower_red, upper_red)

lower_red = np.array([170, 50, 50])
upper_red = np.array([180, 255, 255])
mask2 = cv2.inRange(hsv, lower_red, upper_red)

# Combine the masks to get the final mask for red color
mask = mask1 + mask2

# Apply morphological operations to the mask to remove noise
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

# Detect circles in the image
circles = cv2.HoughCircles(opening, cv2.HOUGH_GRADIENT, dp=1, minDist=50, param1=50, param2=30, minRadius=0, maxRadius=0)

```

```

# Count the number of circles
num_circles = len(circles[0])

```

```
print("Number of red circles detected:", num_circles)
```

ChatGPT: There are a few things that could be causing issues with this code:

1. The image path: The image path specified in the **cv2.imread** function may not be correct. Make sure that the image is located in the same directory as the Python script or specify the full path to the image.
2. No circles detected: It's possible that the Hough circle transform is not detecting any circles in the image. This could be due to factors such as insufficient contrast or too much noise in the image. You can try adjusting the parameters for Hough circle detection or applying additional preprocessing steps to improve circle detection.
3. Error with accessing **circles[0]**: If no circles are detected in the image, **circles** will be **None** and trying to access **circles[0]** will result in an error. To avoid this, you can add a check to see if **circles** is not **None** before attempting to access **circles[0]**.

Here's an updated version of the code that addresses these issues:

```

import cv2
import numpy as np

# Load the image
img = cv2.imread('space_image.jpg')

# Convert the image to HSV color space
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Define the lower and upper bounds for the red color in HSV
lower_red = np.array([0, 50, 50])
upper_red = np.array([10, 255, 255])
mask1 = cv2.inRange(hsv, lower_red, upper_red)
lower_red = np.array([170, 50, 50])
upper_red = np.array([180, 255, 255])

```

```

mask2 = cv2.inRange(hsv, lower_red,
upper_red)

# Combine the masks to get the final
mask for red color
# Apply morphological operations to
the mask to remove noise
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(mask,
cv2.MORPH_OPEN, kernel)

# Detect circles in the image
circles = cv2.HoughCircles(opening,
cv2.HOUGH_GRADIENT, dp=1,
minDist=50, param1=50, param2=30,
minRadius=0, maxRadius=0)

# Count the number of circles
if circles is not None:
    num_circles = len(circles[0])
    print("Number of red circles
detected:", num_circles)
else:
    print("No circles detected in
the image.")

# Apply morphological operations to
the mask to remove noise
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(mask,
cv2.MORPH_OPEN, kernel)

# Detect circles in the image
circles = cv2.HoughCircles(opening,
cv2.HOUGH_GRADIENT, dp=1,
minDist=50, param1=50, param2=30,
minRadius=0, maxRadius=0)

# Count the number of circles
if circles is not None:
    num_circles = len(circles[0])
    print("Number of red circles
detected:", num_circles)
else:
    print("No circles detected in
the image.")

```

This updated version of the code includes a check to see if **circles** is **None** before attempting to access **circles[0]**. It also includes an additional print statement in case no circles are detected in the image.

Andrew: What do the masks in this code do?

ChatGPT: The masks in this code are used to filter out all pixels in the image that are not within a specified range of red colors in the HSV color space.

First, the image is converted to HSV color space using the **cv2.cvtColor** function. This is because HSV color space separates color information into three components: hue, saturation, and value. In this case, we are interested in the hue component, which represents the color itself.

Next, two masks are created using the **cv2.inRange** function. These masks are binary images that are white wherever the original image falls within the specified lower and upper bounds, and black everywhere else. The **mask1** and **mask2** in the code are used to detect red colors that fall within the lower and upper hue values of 0-10 and 170-180, respectively. These values were chosen because the hue values for red colors fall within these ranges in the HSV color space.

Finally, the two masks are combined using the **+** operator to get the final mask for red color. This mask is a binary image that is white wherever the original image falls within either of the specified hue ranges, and black everywhere else. This final mask is then used for further processing, such as circle detection using the Hough transform.

Andrew: How can I view the HSV color space in this code?

ChatGPT: You can view the HSV color space in this code using the **cv2.imshow** function to display the image after it has been converted to the HSV color space. Here's an example of how you can modify the code to display the HSV image:

```

import cv2
import numpy as np

# Load the image
img = cv2.imread('space_image.jpg')

```

```

import cv2
import numpy as np

# Load the image
img = cv2.imread('space_image.jpg')

# Convert the image to HSV color space
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Display the HSV image
cv2.imshow('HSV image', hsv)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Define the lower and upper bounds for the red color in HSV
lower_red = np.array([0, 50, 50])
upper_red = np.array([10, 255, 255])
mask1 = cv2.inRange(hsv, lower_red, upper_red)

lower_red = np.array([170, 50, 50])
upper_red = np.array([180, 255, 255])
mask2 = cv2.inRange(hsv, lower_red, upper_red)

# Combine the masks to get the final mask
mask = mask1 + mask2

# Apply morphological operations to the mask to remove noise
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

# Detect circles in the image
circles = cv2.HoughCircles(opening, cv2.HOUGH_GRADIENT, dp=1,
minDist=50, param1=50, param2=30, minRadius=0, maxRadius=0)

# Count the number of circles
num_circles = len(circles[0])

```

```

print("Number of red circles detected:", num_circles)

```

This will display a window showing the HSV color space representation of the image. You can close the window by pressing any key. Note that the HSV image will look different from the original RGB image because it separates the color information into three components, as mentioned in my previous answer.

Andrew: Running this code gives me the error message: Assertion failed) !_src.empty() in function 'cvtColor'

ChatGPT: The error message "Assertion failed) !_src.empty() in function 'cvtColor'" means that the image you are trying to convert to the HSV color space is empty or does not exist in the specified file path.

You should check the following:

1. Make sure that the file name and file path are correct and that the image is in the specified location. You can check the file path by using the **os.path.exists** function:

```

import os

image_path = 'path/to/image.jpg'

if os.path.exists(image_path):
    print('Image file found')
else:
    print('Image file not found')

```

2. Make sure that the image file is in a format that OpenCV can read. OpenCV can read most common image formats, such as JPEG, PNG, and BMP.
3. Make sure that the image file is not empty. You can check the size of the image file by using the **os.path.getsize** function:

```

import os

image_path = 'path/to/image.jpg'

if os.path.getsize(image_path) > 0:
    print('Image file not empty')

```

```
else:  
    print('Image file empty')
```

If you have confirmed that the image file exists, is in a supported format, and is not empty, you may need to check the syntax of your code to make sure that the image is being read and loaded correctly.

Andrew: How fast should the program you wrote run?

ChatGPT: The speed of the program I wrote for detecting the number of red circles in an image can vary depending on several factors, including the size of the image, the number of circles in the image, and the performance of the computer running the program.

In general, however, the program should run relatively quickly for typical image sizes and numbers of circles. The most time-consuming parts of the program are usually the morphological operations and the circle detection algorithm. The Hough circle detection algorithm has a time complexity of $O(n^2)$, which means that the processing time increases quadratically with the number of pixels in the image. However, OpenCV's implementation of the algorithm is optimized for speed and can handle images with large numbers of pixels relatively quickly.

In most cases, the program should be able to detect the number of red circles in an image within a few seconds to a few minutes, depending on the factors mentioned above. If the program is taking too long to run, you may want to consider optimizing the parameters of the circle detection algorithm or using a faster computer or graphics processing unit (GPU) for processing the image.

Andrew: I ran the code you gave me and it found the image file but found no circles

ChatGPT: If the program was not able to detect any circles in the image, there are several possible reasons why this might have happened:

1. The red circles in the image may be too small or too large to be detected by the circle detection algorithm. You can try adjusting the **minRadius** and **maxRadius** parameters in the **cv2.HoughCircles** function to account for this.
2. The red circles in the image may be too faint or too bright to be detected by the color thresholding

process. You can try adjusting the **lower_red** and **upper_red** bounds in the HSV color space to account for this.

3. There may be other objects or features in the image that are being detected as circles, which are not actually red circles. You can try adjusting the morphological operations to remove noise and non-circular features from the image.
4. The red circles in the image may be partially obscured or hidden from view by other objects or features in the image. In this case, it may not be possible to detect all of the circles in the image using the circle detection algorithm alone.

To troubleshoot the issue, you can try visualizing the intermediate steps of the image processing pipeline, such as the thresholded mask and the morphological operations, to see if there are any issues with these steps that are preventing the circles from being detected. Additionally, you can try using a different image or a subset of the image to test the program and see if it is able to detect circles in that case.

Andrew: What size of circle did you ask the program to detect?

ChatGPT: In the code I provided, the **minRadius** and **maxRadius** parameters in the **cv2.HoughCircles** function were set to **0**, which means that the algorithm will try to detect circles of any size.

If you want to detect circles of a specific size range, you can adjust these parameters accordingly. For example, setting **minRadius** and **maxRadius** to **10** and **50**, respectively, would tell the algorithm to only look for circles with radii between 10 and 50 pixels. Note that the appropriate range of radii depends on the size of the circles in the image you are trying to detect, so you may need to experiment with different values to get the best results.

Andrew: What does minDist represent?

ChatGPT: The **minDist** parameter in the **cv2.HoughCircles** function represents the minimum distance between the centers of detected circles.

When the Hough Circle Transform algorithm is applied to an image, it can detect multiple circles that overlap or are located close to each other. To avoid detecting the same circle multiple times, the algorithm

enforces a minimum distance between the centers of the detected circles.

The **minDist** parameter specifies the minimum distance between the centers of the detected circles, in pixels. If two circles are detected with centers closer than this distance, only the circle with the stronger edge response will be returned.

The appropriate value for **minDist** depends on the size of the circles you are trying to detect and the amount of overlap between circles in the image. Setting a small value for **minDist** will allow the algorithm to detect circles that are close together, but may result in duplicate detections. Setting a larger value for **minDist** will reduce the likelihood of duplicate detections, but may cause some circles to be missed.

Andrew: What do param1 and param2 represent?

ChatGPT: The **param1** and **param2** parameters in the **cv2.HoughCircles** function control the sensitivity and accuracy of the circle detection algorithm.

- **param1** represents the threshold for the edge detection stage of the algorithm. This value determines how strong the edges in the image need to be in order to be considered for circle detection. Increasing **param1** will result in the algorithm being more selective about which edges are considered, which can improve accuracy but may also result in some circles being missed.
- **param2** represents the threshold for the circle detection stage of the algorithm. This value determines how strong the accumulation of edge points needs to be in order for a candidate circle to be considered a valid detection. Decreasing **param2** will result in the algorithm being more lenient about which edge accumulations are considered, which can improve sensitivity but may also result in false detections.

The appropriate values for **param1** and **param2** depend on the characteristics of the image you are trying to detect circles in, such as the amount of noise and the contrast between the circles and the background. You may need to experiment with different values to find the best combination for your particular image.

Andrew: Can I export our conversation?

ChatGPT: I'm sorry, but as an AI language model, I don't have the capability to export our conversation. However, you can select and copy the text from this chat and paste it into a text file or any other application that you prefer.

Andrew: How should I cite our conversation?

ChatGPT: As an AI language model, I don't have an official citation format, but you can cite our conversation by including the following information:

- Name of the AI model: ChatGPT
- Developer: OpenAI
- Date of the conversation: [Insert the date of the conversation here]
- URL or other identifying information, if applicable: [Insert any relevant URL or other identifying information, such as the platform or application where the conversation took place]

Here's an example citation in APA style:

OpenAI. (n.d.). ChatGPT. Retrieved [insert date of conversation here], from [insert platform or application where conversation took place]

Alternatively, you can check with the guidelines of the specific style guide you are using to determine the appropriate citation format.

APPENDIX F: OBJECT SUBCLASSIFICATIONS AND GENERALIZATIONS

Galaxy

Spiral galaxy

Lenticular galaxy

Elliptical galaxy

Barred spiral galaxy

Galaxy

Irregular galaxy

Interacting galaxy

Barred lenticular galaxy

Interacting galaxies

Intermediate spiral galaxy

Unbarred spiral galaxy

Seyfert galaxy

Unbarred lenticular galaxy

Lenticular

Intermediate lenticular galaxy
 Barred Magellanic spiral galaxy
 Barred irregular galaxy
 Ring galaxy
 Spiral galaxy occulted by another spiral galaxy
 Barred spiral
 Barred Magellanic irregular galaxy
 Grand design spiral galaxy
 Dwarf galaxy
 Elliptical
 Disk galaxy
 Spiral galaxy, peculiar
 Lenticular galaxy pair
 Lenticular galaxy/Spiral galaxy
 Dumbbell galaxy
 Peculiar galaxy/Lenticular galaxy
 Radio galaxy
 Dwarf elliptical galaxy
 Barred lenticular LINER galaxy
 Starburst galaxy
 Magellanic spiral
 Intermediate lenticular galaxy
 Dwarf spheroidal galaxy
 Spiral arm

Asterism

Asterism
 Asterism of four stars, mistaken for a galaxy
 Asterism, mistaken for an open cluster
 Unknown, possibly an asterism

Cluster

Globular cluster
 Open cluster
 Star cluster
 Star cluster candidate
 Diffuse nebula and star cluster

Nebula

Planetary nebula
 Diffuse nebula
 Emission nebula
 Reflection nebula
 Nebula

Star

Star
 Double star
 Association of stars
 Triple star
 Multiple star
 Possible double star
 Binary star
 Stellar association

Other

Interstellar matter
 Supernova remnant
 Star forming region
 Variable star and diffuse nebula
 Star cloud

N/A

Unknown
 non-existent
 Doesn't exist
 Nonexistent
 (Identification uncertain)

ACKNOWLEDGEMENT

I owe a big thanks to a great many people, starting first and foremost fellow researchers Susan Nash and Cadence Moore whose support and feedback were invaluable to my research.

I'm also very grateful to each of my advisors for all the time and effort they each volunteered to help me. Drs. Lisa Goodhew, John Lindberg, Dennis Vickers, and Christine Chaney were all instrumental in helping me navigate professional research and writing and in providing insight when I needed a sounding board.

I am also thankful to Emmanuel Byers (creator of Source Extractor), Dustin Lang (Astrometry), and Dr. Douglas Downing (SPU) for their knowledge, to Tayler Quist of Blue Origin for her mentorship in this project and my career, and to Anthony Rinaldi for his technical abilities and the inspiration for the degrees I am in. Thank you also to my editors for helping turn my thoughts into a polished academic paper. Lastly, I am eternally grateful for each of my friends and my family for their support throughout this process

REFERENCES

- [1] Petroc Taylor, "Volume of Data/Information Created, Captured, Copied, and Consumed Worldwide from 2010 to 2020, with Forecasts from 2021 to 2025," IDC & Statista, Jun. 2021. [Online]. Available: <https://www.statista.com/statistics/871513/worldwide-data-created/>
- [2] Zooniverse, "About," *Zooniverse*. <https://www.zooniverse.org/about>
- [3] OpenAI, "GPT-4." OpenAI, Mar. 14, 2023. [Online]. Available: <https://openai.com/research/gpt-4>
- [4] High Energy Astrophysics Science Archive Research Center, "The FITS Support Office at NASA/GSFC." National Aeronautics and Space Administration. [Online]. Available: <https://fits.gsfc.nasa.gov>
- [5] IAU FITS Working Group, "IAU." International Astronomical Union. [Online]. Available: <https://fits.gsfc.nasa.gov/iaufwg/iaufwg.html>
- [6] Science Mission Directorate, "Visible Light." National Aeronautics and Space Administration, 2010. [Online]. Available: https://science.nasa.gov/ems/09_visiblelight
- [7] François Chollet, *Deep Learning with Python*, Second Edition. Shelter Island, NY: Manning Publications, 2021.
- [8] A. Turing, "Memorandum to OP-20-G on Naval Enigma (c.1941)," in *The Essential Turing*, Oxford University Press, 2004. doi: 10.1093/oso/9780198250791.003.0013.
- [9] H. Shevlin, K. Vold, M. Crosby, and M. Halina, "The Limits of Machine Intelligence: Despite Progress In Machine Intelligence, Artificial General Intelligence is Still a Major Challenge," *EMBO Rep.*, vol. 20, no. 10, Oct. 2019, doi: 10.15252/embr.201949177.
- [10] OpenAI, "Generative Pre-trained Transformer." OpenAI, 2021. [Online]. Available: <https://openai.com/blog/gpt-3-apps/>
- [11] B. Dayma *et al.*, "Borisdayma/Dalle-Mini: Initial Release." Zenodo, Jul. 29, 2021. doi: 10.5281/ZENODO.5146400.
- [12] Kevin Roose, "A Conversation With Bing's Chatbot Left Me Deeply Unsettled," *The New York Times*, Feb. 17, 2023. [Online]. Available: <https://www.nytimes.com/2023/02/16/technology/bing-chatbot-microsoft-chatgpt.html>
- [13] "Astrometry User Images," *Astrometry*. https://nova.astrometry.net/user_images
- [14] Dustin Lang, "astrometry/net/api.py." Astrometry, Sep. 07, 2022. [Online]. Available: <https://github.com/dstndstn/astrometry.net/blob/main/net/api.py#L462>
- [15] "Yale Bright Star Catalog." Harvard University. [Online]. Available: <http://tdc-www.harvard.edu/catalogs/bsc5.html>
- [16] George O. Abell, Harold G. Corwin Jr., and Ronald P. Olowin, "A Catalog of Rich Clusters of Galaxies," *Astrophys. J. Suppl.*, vol. 70, May 1989, doi: 10.1086/191333.
- [17] Browse Software Development Team, "HDEC - Henry Draper Extension Charts Catalog." NASA Goddard Space Flight Center, Nov. 23, 2022. [Online]. Available: <https://heasarc.gsfc.nasa.gov/W3Browse/star-catalog/hdec.html>
- [18] E. Høg *et al.*, "The Tycho-2 catalogue of the 2.5 million brightest stars," *Astron. Astrophys.*, vol. 355, pp. L27-L30 (2000), Mar. 2000.
- [19] Wolfgang Steinicke, "Revised New General Catalogue and Index Catalogue." NGC/IC Project, Mar. 20, 2023. [Online]. Available: <http://www.klima-luft.de/steinicke/ngcic/rev2000/Explan.htm>
- [20] Hartmut Frommert, "The Interactive NGC Catalog Online." Students for the Exploration and Development of Space. [Online]. Available: <http://spider.seds.org/ngc/ngc.html>
- [21] AstroBin, "tmpdygg3587 " (*Submission 7133065*). 2023. [Photograph]. Available:

- https://nova.astrometry.net/user_images/7304470#redgreen
- [22] AstroBin, "*tmpdygg3587* " (Submission 7133065). 2023. [Photograph]. Available: https://nova.astrometry.net/user_images/7304470#annotated
- [23] AstroBin, "*tmpdygg3587* " (Submission 7133065). 2023. [Photograph]. Available: https://nova.astrometry.net/user_images/7304470#extraction
- [24] Benne W. Holwerda, "Source Extractor for Dummies." Space Telescope Science Institute. [Online]. Available: http://astroa.physics.metu.edu.tr/MANUALS/seextractor/Guide2source_extractor.pdf
- [25] Max Howell, "Homebrew." [Online]. Available: <https://brew.sh>
- [26] Emmanuel Bertin, "Source Extractor." GitHub. [Online]. Available: <https://github.com/astromatic/seextractor>
- [27] Dustin Lang, David W. Hogg, Keir Mierle, Michael Blanton, and Sam Roweis, "Nova.astrometry.net: API." [Online]. Available: <http://astrometry.net/doc/net/api.html>
- [28] Vishal Hule, "Parse a JSON Response Using Python Requests Library." PyNative.com, May 14, 2021. Accessed: Feb. 13, 2023. [Online]. Available: <https://pynative.com/parse-json-response-using-python-requests-library/>
- [29] The Astropy Collaboration *et al.*, "The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package," 2022, doi: 10.48550/ARXIV.2206.14220.
- [30] The Astropy Collaboration *et al.*, "The Astropy Project: Building an inclusive, open-science project and status of the v2.0 core package," 2018, doi: 10.48550/ARXIV.1801.02634.
- [31] The Astropy Collaboration *et al.*, "Astropy: A Community Python Package for Astronomy," 2013, doi: 10.48550/ARXIV.1307.6212.
- [32] The Astropy Developers, "Cutout2D." Astropy. [Online]. Available: <https://docs.astropy.org/en/stable/api/astropy.nddata.Cutout2D.html>
- [33] Andrew Macpherson, "Astronomical Identification ML Appendixes." GitHub, May 01, 2023. [Online]. Available: <https://github.com/admacpherson/Astronomical-Identification-ML/tree/main/Appendixes>
- [34] John Hunter and Darren Dale, "Choosing Colormaps in Matplotlib." The Matplotlib Development Team. [Online]. Available: <https://matplotlib.org/stable/tutorials/colors/colormaps.html>
- [35] Gavin Edwards, "Machine Learning | An Introduction," *Towards Data Science*, Nov. 18, 2018. <https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0>
- [36] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva, "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation." University of Texas - El Paso, Feb. 2018. [Online]. Available: https://scholarworks.utep.edu/cs_techrep/1209/
- [37] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving Neural Networks By Preventing Co-Adaptation of Feature Detectors." arXiv, University of Toronto, Toronto, ON, Jul. 03, 2012. Accessed: Apr. 22, 2023. [Online]. Available: <http://arxiv.org/abs/1207.0580>
- [38] Christophe Pere, "What is activation function?," *Towards Data Science*, Jun. 09, 2020. <https://towardsdatascience.com/what-is-activation-function-1464a629cdca>
- [39] François Chollet, Keras, 2015. [Online]. Available: https://keras.io/api/layers/core_layers/dense/
- [40] Wikipedia contributors, "Softmax function --- Wikipedia, The Free Encyclopedia." Wikipedia, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Softmax_function
- [41] Willy Lutz and AndrzejO, "WARNING:tensorflow:Using a while loop for Converting Cause There Is No Registered Converter for This Op," *stack overflow*. <https://stackoverflow.com/questions/73770146/warningtensorflowusing-a-while-loop-for-converting-cause-there-is-no-registered>
- [42] AstroBin, "*tmphs_u_rtl* " (Submission 7580801). 2023. [Photograph]. Available: https://nova.astrometry.net/user_images/7749608#original
- [43] anonymous, "*00_dovesono.fit* " (Submission 7611192). 2023. [Photograph]. Available: https://nova.astrometry.net/user_images/7779806#original

APPENDIX G: SEATTLE PACIFIC UNIVERSITY HONORS SYMPOSIUM PRESENTATION

Presented orally on May 20, 2023 at Seattle Pacific University to the 3rd Annual Honors Symposium as part of the panel "How Does Truth Grow?: Evolution and Adaptations of Knowledge"

Good morning, thank you so much for being here.

I'm excited to have the opportunity to share with you all what I've been working on over the past year alongside these illustrious other scholars. While our topics are quite different and we occupy a wide range of disciplines, we all share a mind for the liberal arts and see common themes among our work. It is our hope that, through the presentation of our work, we can make these connections apparent and challenge your perceptions of discontinuity between disciplines.

As you may have seen in your programs, I have the pleasure of talking about Artificial Intelligence – AI – and the good fortune to have spent a year researching a topic that most people have only cared about for 6 months. You've probably heard a lot about AI in this last half-year or so but unless you're as exceptionally intrigued by it as I am, may not have spent time deciphering and unpacking every piece of it.

The term AI means a lot of things to a lot of people. When I asked my parents what they thought of when they heard the term, they said, "robots taking over" and "am I going to be able to trust anything I read ever again?" and I think this tracks with a lot of popular conceptions.

A lot of us picture sentient machines coming to eradicate humanity, or maybe you've heard about "the end of High School English" as students use tools like ChatGPT to cheat on their essays. Some of these concerns are certainly more warranted than others, but they all fall under the umbrella of AI.

But what is AI? Over my last year of research, I've come across all sorts of definitions that all draw different boxes around what's included and what's not. But I want to offer a definition of the term "AI" from François Chollet, the author of my favorite AI book and several AI tools, that I think captures our doomsday scenarios but also grounds it in more present applications.

Before I define it, I actually owe a bit of an apology to my mom. I cut off her definition a little bit to make my doom and gloom point, and because it was a little too similar to my own. Her full response to my question was "robots taking over human activities." What I came up with after a full year is "the effort to automate individual tasks normally performed by humans." ...So my mom may have a future as an AI researcher if she wants it.

But I think this definition captures a lot. Automating human tasks is not inherently a bad thing. So much of what we interact with on a daily basis is built on automating jobs people used to do or maybe never would have done in the first place.

Every time you swipe your credit card, your bank uses an AI to check for fraudulent activity instantly rather than having someone sort through paper logs all day.

Every time you open Facebook or Twitter, you're served a personalized feed based off what a machine thinks you would enjoy most. I wasn't in those social media boardrooms, but I don't think having a single guy behind the computer arranging all the tweets was ever on the table.

But think about the last time you bought a book...how did you hear about it? For some of us, it may have been a friend, or we stepped into a bookstore and picked something out from the staff recommendations shelf. Or maybe it was the Amazon recommendations or something you saw on social media. Whether we like it or not, whether we realize it or not, AI is already part of our lives, and its role will only continue to expand.

I want to talk more about what an AI revolution means in our modern age, but I think it may be helpful to examine a case study with ideas about the construction of knowledge in mind.

My research hasn't just covered AI generally but specifically expanding and applying this technology to my other area of focus: outer space.

As we enter into an age of Big Data, the field of analytics has seen its mission change: rather than dealing with the challenges of too little information, we're more often dealing with too much of it. When there are millions or billions of data points, how are we supposed to make sense of this sea of information? How do we determine which is useful and when?

The field of astrophysics is not unique in facing this problem. Amidst a flood of images, humans simply no longer possess the resources to catalog and analyze every one of them. Astronomers might be able to recognize a galaxy or a nebula in a single image but what if it's in a group of a thousand others? What about when there are ten thousand images?

This is the context I want to apply AI to. Classification is something computers are actually really good at.

But they don't learn in the same way as you or I. If I showed you a hundred photos that were all of either a dog or cat, you could pretty confidently separate them into two categories. But try describing exactly what makes them different from each other or from any and every other animal without leaving any actual dogs or cats out. Can you come up with a definition that includes both golden retrievers and chihuahuas? What about if the dog is missing something like a leg or tail?

I don't think many of us would mistake a three-legged dog for a cat but if you've told a computer that a dog always has four legs then showed it a three-legged dog, it would have no idea what it was looking at.

To solve this explicit definition problem, I used a technology called deep neural networks, which belong to a subfield of machine learning, a broad term referring to anything in AI where the program can receive feedback and improve itself from it. I won't delve into the technical details here – though if you're curious they're all laid out in my paper – but we are in essence giving the computer a quiz.

If we show it a hundred dogs and a hundred cats and ask it to categorize them, at first it will be randomly guessing which one each is. But if each time we tell it the correct answer afterwards, eventually it will start to pick up patterns. It will start learning features in images that give it clues to the right answer. And this is what I'm doing in the context of astronomy. We have lots of photos of things we know are galaxies and stars and nebulae that we can use to teach the computer. I spent a large part of my project figuring out how to extract photos of those individual bodies and label them in a way the neural network understands. But once we've prepared lots and lots of those images and given them to the machine, it will start to pick up distinguishing features of each object. The idea is to gradually filter each image from something you and I can look at to something the computer can recognize patterns from.

The term neural network actually comes from what this technique was modeled after: the human brain. The model is built of a series of "neurons" that each perform an individual operation that slightly changes the image. When series of neurons work together, they form layers that perform recognizable large-scale changes to the image that are useful for machine learning.

But though it's modeled after the human brain, how much is this neural network actually "learning?" Stepping back further, what does it mean to learn? Does the computer actually know what a galaxy or a star is, or did it just learn to pass the tests we gave it?

Without being an astrophysicist, you could still probably tell me stars and galaxies aren't the same thing and offer a cursory definition of each. But to the computer, the difference in a star and a galaxy is just a set of clues in the image data, not anything intrinsic that describes the real world. To a computer, both of these are just pixels on a screen but to you and I, they define our place in the universe.

With enough data, deep neural networks can get better at their tasks than any person ever could, but is it fair to say they know more?

Many of you may be familiar with ChatGPT, the program I mentioned earlier that you can ask questions of like a person and get really accurate and specific knowledge from. But even though we can learn from the information the program gives us, does it know it? How can we learn from a teacher that doesn't possess knowledge?

I think this reveals another scary feature of artificial intelligences: because they don't truly know anything, there's no guarantee the information is accurate. It's learned to pass enough tests that it can generally give you an answer that at least sounds right...and usually is. But if you ask it where it got its information from, it doesn't always know.

Neural networks are often called black boxes because, while we know what data we gave it and what we asked it to do, we don't know exactly how it does it.

I have up here a simplified diagram of a neural network but when there are thousands of data points and hundreds of layers, it becomes literally impossible to keep track of the flow of information.

Or when the input is the entirety of what's available online, we might not know what we even gave it. This is why chatbots often become quickly problematic. A few years ago, Microsoft unveiled a Twitter bot called Tay that learned from the speech patterns of users who interacted with it. But when users started posting hateful content, the bot quickly started echoing it, spouting Nazi propaganda and harmful conspiracy theories.

Chatbots aren't people. It may seem obvious to say that but what's stopping Tay from saying those things? As far as "she's" concerned, she's getting better. She is emulating not only speech from real Twitter users but actual viewpoints. She doesn't know that her words can cause real harm and that they correspond to things in the physical world that real people are affected by.

I don't mean to sound down about the future of artificial intelligence or scare you too much. But I do hope it scares you a little bit. AI, like any other technology, is an inherently neutral thing. Whether it becomes a tool for good or for evil depends entirely how we use it.

In this way, I like to compare it to the Internet. Many of us, including myself, cannot remember a time before Google, but if you do, think about whether you thought it was a good or a bad thing when it first came out. Think about how you would classify it now. I think the vast majority of us would say Google has made our lives orders of magnitude easier and given us access to information we would never have had at a speed the entirety of human history could not even dare to imagine.

But I don't have to tell you that Google comes with harm too. The people behind it try to moderate its content as best they can but they don't know everything that's on there. Pornography, not always consensual, is rampant, and cybercrime steals from millions of people and can cripple whole nations. Is the tradeoff worth it?

I don't think many of us would call it societal progress to go back to the pre-search engine days, but that progress doesn't come without concurrent harm. The same technology that could be the bedrock of clean energy is also responsible for nuclear destruction. The morality of technology is exactly what we make it.

I think AI is going to change the way we live in the same way the Internet did. Soon we'll be wondering how we ever went about life without it and its incredible utility. But my hope is that along the way, instead of merely asking what we *can* do next, that we'll ask what we should do as well.

I want to leave you with one final thought experiment that's sat with me for many years. There's a short story by Jorge Borges that was published over 75 years ago but still holds tremendous relevance today. It's called *The Library of Babel* and it imagines a library of almost infinite proportion that contains a book for every possible combination of letters.

Evidently, the vast majority of these books are pure gibberish, but the library must also contain, word for word, every book ever written, every sentence ever spoken, and every one that's not. Somewhere it describes every second of your life in perfect detail but also in complete error in many other copies. Your deepest desires and darkest secrets

are written down but so are the ones that don't belong to you at all. Shakespeare's *Hamlet* is there word for word along with copies that switch out single words, sentences, or scenes for ones that make no sense at all.

In this world where every novel statement has already been said, where no possible set of words and letters creates a unique permutation, what does it mean to create? To know? Does *Hamlet* mean anything if it was randomly generated?

To the library, as with our computers, none of these combinations mean any more than the others. The difference between *Hamlet* and utter gibberish is only in our interpretations of it. Borges wrote his original story in Spanish, and it didn't see an English translation until over 20 years later. If you presented that text to someone who spoke Spanish and to another who didn't, only one of them would be impacted by it. There is a framework of understanding necessary to making meaning of anything.

I invite you then to consider what it truly means to know and who or what is capable of it. Consider this question throughout the presentations of my colleagues and when you read the next headline proclaiming the newest skill AI is capable of perfecting.

Thank you so much for your time and I'd now like to turn it over to Mr. C.J. Wisely.