

Karakterizacija profila sjajnosti meteorskih tragova i usporedba s drugim linijskim fenomenima

Cerovec, Sven

Master's thesis / Diplomski rad

2025

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:211:698042>

Rights / Prava: [Attribution-NonCommercial-NoDerivs 3.0 Unported / Imenovanje-Nekomercijalno-Bez prerađivanja 3.0](#)

Download date / Datum preuzimanja: **2025-04-01**



Repository / Repozitorij:

[Faculty of Organization and Informatics - Digital Repository](#)



**UNIVERSITY OF ZAGREB
FACULTY OF ORGANIZATION AND INFORMATICS
VARAŽDIN**

Sven Cerovec

**CHARACTERIZING METEOR BRIGHTNESS
PROFILES AND COMPARISON WITH
OTHER LINEAR PHENOMENA**

MASTER'S THESIS

Varaždin, 2025

UNIVERSITY OF ZAGREB
FACULTY OF ORGANIZATION AND INFORMATICS
V A R A Ź D I N

Sven Cerovec

Student ID: 16136668

Programme: Information and Software Engineering

**CHARACTERIZING METEOR BRIGHTNESS PROFILES AND
COMPARISON WITH OTHER LINEAR PHENOMENA**

MASTER'S THESIS

Mentor:

Asst. Prof. Bogdan Okreša Đurić, PhD

Varaždin, Veljača 2025

Sven Cerovec

Statement of Authenticity

Hereby I state that this document, my Master's Thesis, is authentic, authored by me, and that, for the purposes of writing it, I have not used any sources other than those stated in this thesis. Ethically adequate and acceptable methods and techniques were used while preparing and writing this thesis.

The author acknowledges the above by accepting the statement in FOI Radovi online system.

Abstract

This thesis presents the analysis of meteor brightness profiles using more than 26,000 images from within the database maintained by the Sloan Digital Sky Survey (SDSS). In this work, FITS images that contain linear features are studied to determine the differences representative of meteor trails, along with other celestial and atmospheric phenomena, most commonly satellites, to increase the knowledge of their unique lighting characteristics. Methods and tools used in this thesis include Python, Astropy, and Matplotlib and it lays the foundation for automatic classification of these linear features according to their brightness profiles.

Keywords: astrophysics; astrometry; meteor trails; image processing; linear feature detection; brightness profiling

Table of Contents

1. Introduction	1
2. Work Methods and Techniques	2
3. Central Thesis	3
3.1. The History of Celestial Cartography	3
3.1.1. Early Conceptions of the Celestial Sphere	3
3.1.2. Modern Age Star Catalogs and International Efforts	6
3.1.3. Computer Technology and Current Day Projects	8
3.2. Linear Artifacts in Astronomical Images	12
3.2.1. Meteors and Meteor Trails	12
3.2.2. SDSS Data Releases and FITS Files	16
3.2.3. Linear Artifacts in FITS Images	21
3.3. Identification and Classification of Linear Artifacts	28
3.3.1. Automated Linear Artifact Detection	28
3.3.2. Linear Artifact Profiling	37
3.3.3. Classification of Linear Artifacts	42
3.4. Findings and Interpretations	47
3.4.1. Evaluation Metrics for Classification of Linear Artifacts	48
3.4.2. Graphical User Interface for Trail Examination	53
3.4.3. Technical Obstacles and Improvement Possibilities	58
4. Conclusion	59
5. Acknowledgments	60
Bibliography	63
List of Figures	65
List of Tables	66
List of Listings	67

1. Introduction

Meteors have long captivated scientific and public interest, but their transient nature and momentary appearance challenge their observation and analysis. By extracting brightness profiles of trails from images and feeding them into a neural network, this thesis applies deep learning to classify genuine meteor trails from those of satellites or other noise.

The work presented in the following pages explores the identification, profiling, and classification of linear artifacts in astronomical images, with a particular focus on meteors. Detection of linear features is the starting point for a process that then continues with brightness profile analysis and ends with classification of the feature. The methods discussed throughout are rooted in image processing strategies and fortified through a graphical user interface that enables manual evaluation. This work attempts to advance the analysis of transient linear features.

The refinement of this approach has been facilitated considerably by modern surveys—most notably the Sloan Digital Sky Survey—and by the work of Bektešević and Vinković [1] on the automatic detection of elongated streaks in FITS images. The relevant procedures are described, such as detecting, profiling, and classifying meteor trails, in a manner so that specialized knowledge is not strictly required.

2. Work Methods and Techniques

This chapter lists the tools, work methods, and techniques employed throughout this study.

SAOImageDS9 (a widely used astronomical imaging and data visualization application) was used to inspect astronomical images and distinguish meteor trails from non-meteor trails for the neural network classifier's training data.

The images were drawn from the Sloan Digital Sky Survey (SDSS), specifically Data Release 9 (DR9).

Most of the data processing was done in Python and a wide range of its libraries. For linear artifact detection and clustering, OpenCV (for edge detection and image transformation), NumPy (for array operations), and scikit-learn (for any auxiliary clustering or distance metrics) were employed. The profiling of linear artifacts relied again on a combination of OpenCV, NumPy, and astropy (for reading and normalizing FITS data). Finally, the classification neural network centered on scikit-learn's MLPClassifier.

GitHub served as the repository for the project's source code and documentation. The project can be found on the following link:

`https://github.com/svencerovec/meteor-profile`

ChatGPT models *GPT-4o*, *o1*, and *o3-mini-high* were used for research purposes, development suggestions, and code comments.

3. Central Thesis

This chapter provides historical context and investigates solutions for brightness profiling of linear artifacts in astronomical images. It begins with a survey of celestial cartography, tracing early conceptions of the celestial sphere through modern star catalogs and culminating in the role of computer technology in current-day projects. Attention then shifts to linear artifacts, examining meteors and meteor trails, the Sloan Digital Sky Survey (SDSS) data releases, and the specifics of handling linear artifacts within FITS files. Following this overview, the chapter outlines the classification procedures that form the core of the thesis, expanding on how automated detection is combined with profiling and classification. The chapter concludes with findings of the work, including evaluation metrics of the classification, a practical graphical user interface, and a discussion of technical obstacles and possible improvements for future research.

3.1. The History of Celestial Cartography

Wikipedia [2] defines celestial cartography as the aspect of astronomy and branch of cartography concerned with mapping stars, galaxies, and other astronomical objects in the celestial sphere. Merriam-Webster [3] defines uranography as the construction of celestial representations (such as maps), derived from the Greek words *ouranos* meaning sky and *graphia* meaning literature or writing. Encyclopedia Britannica [4] defines cosmology as a field of study that combines the natural sciences, particularly astronomy and physics, in a joint effort to understand the physical universe as a unified whole. The effort to map the universe in the field of cosmology is called cosmography. Celestial cartography, uranography, and cosmography are some of the words one could use to describe the area of astronomy that aims to map the universe. The language used throughout this work is celestial cartography.

Celestial cartography is a core tool in modern astronomy, but its inception is far more mystical. To early man, the sky must have seemed like an unchanging sea of tiny bright specks, and just as we charted the land and the sea we stand and sail on, so did ideas arise to map the untouchable sea that is our night sky. Technological advancement throughout the ages has revolutionized this field, due to the integration of computer technologies with data collection. This section explores the journey of celestial cartography from its beginnings, through its scientific metamorphosis in the modern era, to its current synergy with computer technologies. This section's first and second subsections cover the history of celestial cartography. The section draws upon the work of Nick Kanas in his book titled *Star Maps: History, Artistry, and Cartography* [5].

3.1.1. Early Conceptions of the Celestial Sphere

The history of celestial cartography is a fascinating tale that spans several thousands of years. Early civilizations gave constellations mythological meanings and believed that they were images that represented different people, animals, or objects. Although the mythological

context of the sky was prominent throughout history and the same mythological meanings of constellations have remained to this day, throughout time, it began to fade with the discovery of more practical uses of celestial information. Driven by needs like timekeeping and navigation, humanity began recording the positions of celestial bodies in detail and creating what is commonly referred to as celestial maps. The interest of ancient civilizations in celestial phenomena has laid the foundation for a modern understanding of the cosmos.

Dating back 3,600 years to the late Bronze Age [6], the Nebra sky disc (shown in Figure 1) provides a unique glimpse into humanity's early knowledge of celestial objects. Although the disc is not quite what one would describe as an accurate star map, measuring about 30 cm in diameter, it features a blue-green patina adorned with gold symbols representing the most important celestial figures: the Sun, the Moon, and the stars. Whether the Pleiades, a cluster of seven stars (known in mythology as *The Seven Sisters*), are also depicted on the disc is debatable. If so, the disc would be one of human history's first known representations of a constellation. [7]



Figure 1: Photograph of the Nebra sky disc by Frank Vincentz [7]

The Nebra sky disc is more of an artistic depiction of the sky than a practical one. But an example of a practical tool turned decorative is an object called the armillary sphere. It was used to visualize the celestial sphere and calculate the positions of stars in the 3rd century BC by scholars like Eratosthenes in Greece and China during the Han dynasty. Later, its use spread to Islamic countries and continued into the Renaissance for education and navigation training. Today, the armillary sphere can be found in libraries and households as decoration, with modern technology having rendered it obsolete. [5]

A more practical example of an astronomical tool would be celestial maps, which were historically found in atlases specifically dedicated to celestial cartography. However, they were included in geographical atlases to complement terrestrial maps. These maps were used for celestial navigation, which involves using stars and other celestial bodies to determine one's position on Earth, but also as educational tools. Early celestial maps serve as cultural indicators that illustrate the state of knowledge and politics of their times and reflecting the cosmological beliefs of early cultures such as the ancient Greeks. Celestial mapping uses coordinate systems projected onto the celestial sphere, akin to the latitude and longitude we use for planet Earth, which change with the Earth's movements. [5]

The connection between celestial and geographical cartography owes much of its prominence to the contributions of the Greco-Egyptian astronomer and geographer Ptolemy. Ptolemy authored two influential works in the scientific area of cartography: the *Almagest* and the *Geographica*. While his *Almagest* documents planetary positions in the sky and lunar cycles, its counterpart, the *Geographica*, charts cities and landmarks on what was considered the Earth at the time. [5] [6]

Geographica, representing a significant milestone in the history of geographical cartography, also yielded advances in the astronomical field, most notably within Ptolemy's world map. Although the map showed locations on the known Earth during its time (around 150 CE), the methods used were enriched by astronomical observations. [8] Through the map, Ptolemy introduced latitude and longitude as part of a planar coordinate system, which is now in commonplace use not only for planet Earth but for all spherical surfaces, including the celestial sphere. [6]



Figure 2: Ptolemy's 150 CE World Map (redrawn in the 15th century) [8]

As detailed in his *Almagest*, Ptolemy's star catalog is a very influential astronomical work that lists 1022 stars in 48 constellations. It includes each star's location within its constellation,

its longitude and latitude on a similar coordinate system used in Ptolemy's world map, and its brightness on a scale from 1 to 6. [5] The star catalog influenced subsequent astronomical studies and inspired similar catalogs, an example of which can be found in the following chapter (Figure 3).

The fall of Rome greatly affected the area of astronomy. The work of the Greeks was taken over by the Romans, who made few new contributions. In a way, progress in celestial cartography had been halted. Throughout the Middle Ages, astrology was still used for practical purposes, such as determining the appropriate time for medical operations. It was not until the Renaissance period that astrology and astronomy were recognized as two different fields, astrology taking a more pseudoscientific and mystical role and astronomy a scientific and serious one. [5]

The era between 1600 and 1800 is considered The Golden Age of pictorial celestial cartography in Europe. Astronomers like Tycho Brahe and Johannes Hevelius could use improved instruments to collect observational data. The data collected by Tycho Brahe and his team was used by Johannes Kepler in 1627 in his publication named the *Rudolphine Tables*, a star table consisting of 1005 stars (this was after Brahe died in 1601). *Prodromus Astronomiae* by Johannes Hevelius, published posthumously by his wife, contained the *Catalogus Stellarum*, a table of data collected from 1564 stars. [5]

The introduction of telescopes and micrometers further increased the accuracy of the collected data. Printing technology evolved from woodblocks to intaglio processes, allowing detailed images on copper and steel plates to be reproduced on paper. These technical advances in astronomical tools and printing technology created aesthetically pleasing and very technically accurate celestial maps during this era. [5]

3.1.2. Modern Age Star Catalogs and International Efforts

Contemporary star atlases depict stars and other celestial objects with great precision, aided by computer technology. Although modern atlases lack the historical and mythological context of ancient celestial maps, the fundamental purpose of a star atlas remains unchanged: to accurately represent the celestial sphere.

The 19th-century astronomer Friedrich Wilhelm August Argelander was a significant contributor to the area of celestial cartography. One of Argelander's most notable achievements is in his star catalog, in which, following the footsteps of Ptolemy, Brahe, Hevelius, and other astronomers, the attributes of each celestial object were represented through raw data in a table. [5] The technique used by Argelander in his work is an improvement on the work of his predecessors and a precursor to modern star mapping techniques using computer technology. Figure 3 shows an example of his star catalog.

Photography had been integrated into astronomy by the early 19th century. J.W. Draper's 1840 daguerreotypes of the Moon and Edmond Becquerel's daguerreotype of the Sun's spectrum authored some of the earliest images of celestial phenomena. [5]

The following quote from the Library of Congress describes the definition of a da-

Zone 40. 1841 September 21.							Reductionstafel. D = 59° 30'												
Nr.	Gr.	Feld.	Beob. Durchg.	Stw.	Microscop.	Corr.	Beob. Decl.	Bemerkungen und Reductionstafel.		Nr.	Gr.	Feld.	Beob. Durchg.	Stw.	Microscop.	Corr.	Beob. Decl.	Bemerkungen und Reductionstafel.	
136	9	0	5	33	34	40.54	+43	02	4.340	+10.3	51	25	21.4						
137	8	5	0	30	5.69	+45	03	1.620	+9.6	101	22	23.5							
<p>Zone 40. 1841 September 21.</p> <p>1 8.0 3 20 0 54.20 — 66 0.036 + 8.5 57 9 45.7 2 9.0 3 3 17.20 — 43 5.345 + 1.7 59 0 31.0 3 9.0 3 4 0.42 +15 33 6.200 + 3.1 59 49 52.3 4 0.0 4 4 44.12 — 34 3.318 + 3.0 59 47 6.5 5 9 3.4 6 18.28 — 32 4.418 — 0.9 59 56 12.3 6 9 1.3 7 37.33 — 36 38 1.830 + 0.5 59 28 14.4 7 9 1 7 37.33 — 36 38 1.830 + 0.5 59 28 14.4 8 9 4 7 16.34 +23 49 1.745 + 3.2 58 33 21.4 9 9.0 4 6 35.25 +66 48 5.418 + 1.7 58 35 27.7 10 9.0 4 8 35.29 +10 48 0.920 + 1.3 58 39 3.0 11 8 1 8 33.77 +100 68 6.315 + 9.6 56 64 53.5 12 9.0 4 9 2.66 +97 68 0.700 + 9.8 56 59 17.0 13 9.0 4 11 22.50 +21 57 4.715 +11.3 57 51 10.2 14 9.0 4 19 43.35 — 10 55 5.500 + 3.4 59 0 30.8 15 9.0 e.2 14 31.06 — 76 59 0.830 — 0.4 58 19 0.7 16 9.0 5 14 35.89 +27 64 2.010 + 9.3 57 18 15.0 17 9 1 16 12.03 — 45 63 3.780 + 8.7 57 21 51.4 18 9 1 17 5.59 + 46 0.840 + 1.4 58 49 16.1 19 9.0 5 17 43.91 +39 31 4.690 + 4.2 60 1 4.2 20 9 1 17 14.39 +101 32 5.470 — 0.6 59 55 22.9 21 9 0 1 18 23.93 +63 33 5.840 + 3.7 60 0 33.3 22 9 1 21 0.37 — 40 34 4.100 + 1.1 59 46 32.8 23 9 1.2 21 30.44 — 34 33 5.500 + 2.6 59 50 24.7 24 9 2 21 40.08 — 28 32 5.740 — 0.4 59 55 10.4 25 9 3 21 44.80 — 32 33.220 + 0.6 59 57 9.6 26 9 5 21 41.30 +29 37 6.150 + 1.4 59 59 53.0 27 9 5 22 6.42 +30 36 0.190 + 1.1 59 59 35.6 28 9 3 22 26.68 +86 38 0.900 + 1.5 59 59 41.5 29 9 3 24 2.60 — 35 31.70 +9.6 59 42 13.6 30 9 2 24 47.47 — 22 35 0.995 + 1.6 59 43 54.9 31 9 1 25 44.41 — 49 33 1.305 + 1.9 59 45 40.7 32 9 5 25 15.00 +85 41 2.615 + 1.4 59 12 35.6 33 9 5 25 37.39 — 36 41 0.915 + 1.6 59 14 3.4 34 9 1 27 19.09 — 34 39 5.895 + 1.1 59 20 34.7 35 9 2 27 31.29 — 18 41 2.550 + 1.8 59 12 41.6 36 9 2 28 1.56 — 29 37 3.945 + 1.3 59 31 36.3 37 9.0 1.2 28 56.10 — 29 43 2.450 + 1.1 58 52 46.2 38 9 1.2 28 38.95 — 41 44 3.460 + 1.5 58 52 46.2 39 9 4 29 37.71 — 22 64 0.580 + 9.3 57 15 29.0 40 9.0 4 30 33.04 — 65 45.220 + 8.8 57 11 16.8 41 9.0 4 30 44.13 — 17 65 3.575 + 9.9 57 11 47.2 42 8.9 d 30 26.88 +63 63 2.130 + 9.6 57 53 9.7 43 9 5 31 27.42 +34 60 6.300 + 9.8 57 34 54.0 44 9 1 33 33.19 — 33 65 3.670 + 8.9 57 11 55.9 45 9 1 34 58.31 — 63 5.855 + 7.4 57 21 59.3 46 8.9 d 35 29.03 +69 52 1.510 + 2.3 58 18 31.6 47 8.9 f 35 34.32 +97 50 3.385 + 2.8 58 27 4.0 48 8 e 34 15.47 +83 51 1.400 + 3.0 58 23 37.2 49 9 d.8 34 05.49 +77 50 3.295 + 3.3 58 27 8.8</p>																			

Figure 3: A table of star positions from Volume I of Argelander's "Beobachtungen". This volume covers the area of the sky between +45 and +80 degrees declination, published in 1846. The information includes star numbers, magnitudes, right ascension, and declination. [5]

guerreotype and the process of creating one.

"The daguerreotype is a direct-positive process, creating a highly detailed image on a sheet of copper plated with a thin coat of silver without the use of a negative. The process required great care. The silver-plated copper plate had first to be cleaned and polished until the surface looked like a mirror. Next, the plate was sensitized in a closed box over iodine until it took on a yellow-rose appearance. The plate, held in a lightproof holder, was then transferred to the camera. After exposure to light, the plate was developed over hot mercury until an image appeared. To fix the image, the plate was immersed in a solution of sodium thiosulfate or salt and then toned with gold chloride." [9]

Due to the first daguerreotypes' exposure times ranging from three to fifteen minutes, they were unsuitable for portraits and other moving subjects. However, these extended exposures were sufficiently long to capture stationary celestial objects like the Sun and Moon. Advancements in telescopes, lenses, and photographic techniques further established photography as one of the primary astronomical instruments, now commonly called astrophotography. By the late 19th century, astronomers had already captured events such as Venus transiting the Sun, morphological details on Mars, newly discovered comets and asteroids, and deep-sky phenomena including nebulae and star clusters. [5] Ambitious projects surfaced with the aim of mapping celestial objects using photography. Among them, the project *Carte du Ciel* (eng.

Map of the Sky) aimed to map stars across the entire sky.

The *Carte du Ciel* project, initiated in 1887, was an international effort to catalog and map the positions of stars using photography. Observatories from around the world agreed to photograph sections of the sky assigned to them with the goal of capturing approximately 25 million stars in photographs. Due to its apparent ambition, the project faced significant challenges, and despite a decade of effort, the *Carte du Ciel* project remains incomplete. Several sections were never finished due to resource limitations and technical difficulties. [5] The project remains a testament to international scientific collaboration and the pursuit of understanding the cosmos, and it serves as a precursor to future projects such as the goals of the Sloan Digital Sky Survey (SDSS) and Large Synoptic Survey Telescope (LSST).

3.1.3. Computer Technology and Current Day Projects

Like many other scientific fields, celestial cartography has been revolutionized by the rapid advancements in computer technology. Modern telescopes generate enormous volumes of data, which in turn require powerful computational power. In 1998, the Sloan Digital Sky Survey surfaced, intending to answer fundamental questions about the universe by studying hundreds of millions of celestial objects.

Although less ambitious than the *Carte du Ciel* project, the initial aim of the Sloan Digital Sky Survey (SDSS) was to map a quarter of the sky in unprecedented detail. The SDSS telescopes are located at Apache Point Observatory in the Sacramento Mountains of New Mexico, where the atmosphere is exceptionally clear and dark. The main instrument, a 2.5-meter telescope, was specifically built to map large portions of the sky more efficiently than previous telescopes. The telescope employs two reflecting mirrors and a pair of corrective lenses to produce sharp and widefield images about three degrees across, roughly the size of 30 Moons. This design allows for collecting detailed data on a vast swath of the night sky at once. A separate 0.5-meter photometric telescope monitors subtle atmospheric changes to ensure that the proper brightness of celestial objects is recorded. [10]

The 2.5 m wide-angle optical telescope conducted imaging and spectroscopic observations from 1998 until 2009, after which it operated exclusively in spectroscopic mode. As stated in Encyclopedia Britannica [11], spectroscopy studies the absorption and emission of light and other radiation by matter. This emission of light enables precise measurements of the chemical composition, motion, and distance of celestial objects [11], which is why the resources for the operation of the telescope were shifted to the collection of spectroscopic data after the initial imaging phase had been completed.

The operation of the SDSS project can be distinguished into several phases. From 2000 to 2005, SDSS-I produced widefield images and spectra spanning thousands of square degrees and repeated imaging of a stripe in the southern Galactic cap. Then, from 2005 to 2008, SDSS-II introduced expansions such as the Sloan Extension for Galactic Understanding and Exploration (SEGUE) to map the structure of the Milky Way and the Sloan Supernova Survey to study distant Type Ia supernovae. The Sloan Legacy Survey, primarily undertaken in

SDSS-I, covered over 7,500 square degrees of the Northern Galactic Cap and enabled analyses of large-scale cosmic structures. SEGUE obtained spectra for 240,000 stars to build a 3D map of the Galaxy, while the Sloan Supernova Survey discovered hundreds of new supernovae in a 300 square-degree stripe. SDSS-III (2008–2014) encompassed four projects: the Apache Point Observatory Galactic Evolution Experiment (APOGEE) for high-resolution infrared spectra of red giants, the Baryon Oscillation Spectroscopic Survey (BOSS) to measure the expansion rate of the universe via baryon acoustic oscillations, the Multiobject Apache Point Observatory Radial Velocity Exoplanet Large-area Survey (MARVELS) to monitor radial velocities and detect exoplanets, and SEGUE-2 which further examined the Galactic halo. SDSS-IV (2014–2020) continued with the Extended Baryon Oscillation Spectroscopic Survey (eBOSS) for cosmology, APOGEE-2 for more extensive infrared spectroscopy, and the Mapping Nearby Galaxies at Apache Point Observatory (MaNGA) for two-dimensional maps of galaxy interiors. Finally, SDSS-V (2020–present) employs automated fiber-positioning robots, targeting spectra of millions of stars, black hole hosts, and interstellar gas, thus expanding and refining the enduring SDSS legacy. [12]

The telescope imaging camera (shown in Figure 4), cooled to 190 K using liquid nitrogen, employs five photometric filters (see Table 1) and comprises 30 charge-coupled device (CCD) chips (each 2048 × 2048 pixels), yielding a 120-megapixel array with reliable detections of magnitudes of about 20-22 depending on the filter. [12] A CCD or a charge-coupled device is "an integrated circuit containing an array of linked, or coupled, capacitors, which under the control of an external circuit can transfer its electric charge to a neighboring capacitor." [13] Using a drift scan technique with choreographed adjustments in right ascension, declination, and tracking rate, the system continuously records narrow sky strips while synchronously shifting charges on the CCDs. [12]

Filter	Wavelength
u	Near-ultraviolet
g	Green
r	Red
i	Near-infrared
z	Longer-wavelength near-infrared

Table 1: SDSS Photometric Filters and Their Assigned Letter and Wavelength

This drift-scan method provided consistent astrometry across wide fields while minimizing detector readout overhead, although it also introduced slight distortions. After imaging, targets, including stars, galaxies, and quasars, were selected for spectroscopy by aligning optical fibers through holes drilled in an aluminum plate at precise coordinates. Initially, the telescope's spectrograph captured 640 spectra simultaneously, which later upgraded to 1000, allowing six to nine plates of data collection per night. During spectroscopic observations, the telescope follows the sky normally, keeping objects centered on their designated fiber tips for optimal data quality. [12]

In the paper *The Sloan Digital Sky Survey Monitor Telescope Pipeline*, Tucker et al. [14] introduce the following information about the inner workings of the SDSS. An east-west scan

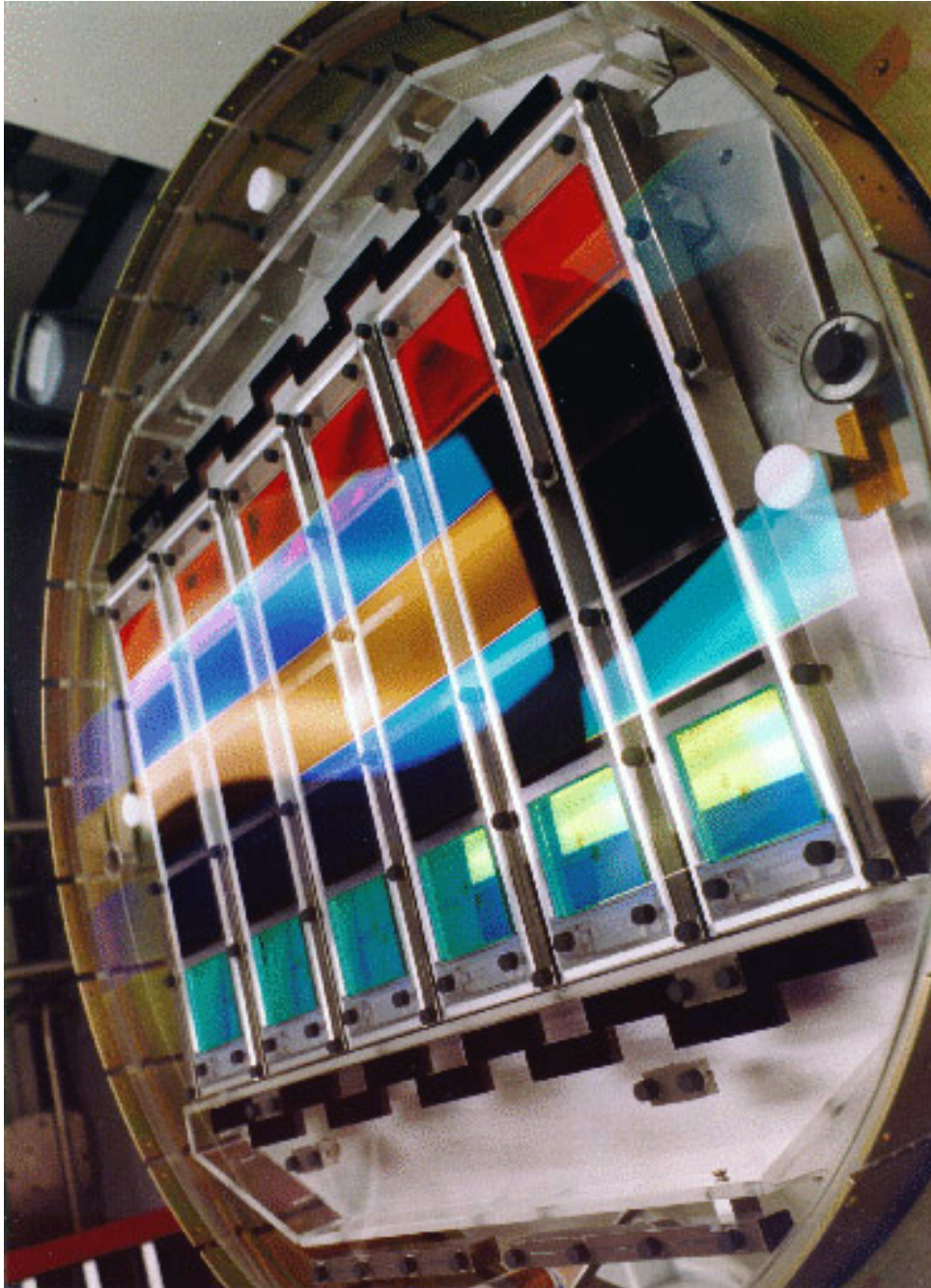


Figure 4: The imaging camera of the SDSS telescope showcases the 5 rows (one for each filter) of 6 charge-coupled devices per row. [12]

defines a 'strip,' and two offset strips make a 'stripe.' The imaging reaches $r = 22.2$ for 95% detection repeatability in point sources. Photometric calibrations achieve an accuracy of about 0.02 mag in g , r , i , and about 0.03 mag in u , z , using a combination of telescopes: the USNO (United States Naval Observatory) 1.0 m telescope for primary standard stars and the two SDSS telescopes: the 0.5 m photometric telescope for nightly zero points and extinction, and the 2.5 m telescope for scanning 'secondary patches.' Follow-up spectroscopy targets galaxies, quasars, and stars via a pair of 320-fiber multiobject spectrographs. Three main software pipelines, one for processing data from the USNO 1.0 m, another for producing astrometry and instrumental photometry from 2.5 m images, and a final calibration pipeline, work together to

compute and apply photometric zero points across the survey data.

The last of the software pipelines called the Monitor Telescope Pipeline (MTPIPE), is a suite of *Tcl* and *C* codes designed to process and calibrate data from the photometric telescope. It includes four main packages: `preMtFrames`, `mtFrames`, `excal`, and `kali`, run in that order. `preMtFrames` sets up the directory structure and tests the raw data quality. The package `mtFrames` creates master-bias, flat, and fringe frames, then applies them to target images before performing aperture photometry. The package `excal` calculates photometric solutions (zero-points, extinction) from standard star observations. At the same time, `kali` handles astrometric calibration and applies the derived photometry to 'secondary patches' used for transferring calibrations to the main SDSS 2.5 m telescope. [14]

According to Wikipedia [12], the SDSS is "a pioneering combination of novel instrumentation as well as data reduction and storage techniques that drove major advances in astronomical observations, discoveries, and theory." As stated in this chapter, there are many reasons why the SDSS should receive praise, a noteworthy one being the unprecedented amount of data generated each night and its storage and processing. It is estimated that the SDSS telescope generates about 200 gigabytes of data each night; during the early 2000s, this was a very large amount of data. [12] These data needed to be stored and accessed; this led to advances in massive database storage and access technology, with the SDSS being the first major astronomical project to make data available to the wider public in this form. [12]

In the paper *The Sloan Digital Sky Survey and its Archive*, Szalay et al. [15] describe the SDSS archiving process. The paper notes that astronomers have standardized the Flexible Image Transport System (FITS) as a self-descriptive data format. Still, efforts at the time also included ASCII and binary streaming solutions. The SDSS Operational Archive sends coherent chunks of data containing several sky segments to the Science Archive for efficient loading and clustering. About 20 GB arrive daily, so proper organization is crucial to avoid bottlenecks during inserts into the hierarchy of spatial containers. Few SDSS attributes interest most researchers, so a small 'tag' dataset (e.g., positions, colors, size) that can be indexed and searched quickly is isolated. A separate sample of 1% (around 10 GB) of the full database offers a lightweight resource for algorithm testing and debugging. Data movement occurs in a distributed system, with an analysis engine that processes large volumes in parallel and only sends the final, reduced results to the user. The engine is tightly integrated with the database server to avoid I/O and network bottlenecks. Vertical partitioning and sampling also improve performance by limiting queries to popular attributes or smaller subsets of the data. This scalable approach "rides Moore's law," growing alongside rapidly increasing CPU speeds and storage capacities. Ultimately, the goal had always been for astronomers to be able to pose complex queries on an evolving, high-dimensional sky atlas and get rapid, detailed answers. [15]

The spiritual successor of the SDSS is the Large Synoptic Survey Telescope (LSST) project of the Vera C. Rubin Observatory. The Rubin Observatory houses the Simonyi Survey Telescope. This telescope will capture images of the entire available sky every few nights, covering a 3.5-degree diameter field of view. The images will be recorded by a 3.2-gigapixel

CCD camera, the largest digital camera constructed yet. The three-mirror design of the LSST guarantees sharp images that contain those intricate details that astronomers are interested in. By observing the same regions repeatedly, a dynamic cosmic atlas will be constructed, enabling breakthroughs in fields such as supernova research, asteroid tracking, and cosmic structure. The Rubin Observatory achieved its first light at the beginning of 2025. [16]

Ivezić et al. describe the LSST in *The Astrophysical Journal*:

Motivated by the evident scientific progress enabled by large sky surveys, three nationally endorsed reports by The US National Academy of Sciences concluded that a dedicated ground-based widefield imaging telescope with an effective aperture of 6–8 m "is a high priority for planetary science, astronomy, and physics over the next decade." The Large Synoptic Survey Telescope (LSST) described here is such a system. Located on Cerro Pachón in northern Chile, the LSST will be a large, widefield, ground-based telescope designed to obtain multiband images over a substantial fraction of the sky every few nights. The survey will yield contiguous overlapping imaging of over half the sky in six optical bands, with each sky location visited close to 1000 times over 10 yr. [17]

From Ptolemy's ancient star catalog to the photographic endeavors of the *Carte du Ciel*, to the digital precision of the Sloan Digital Sky Survey and the current real-time data-driven approach of the Vera C. Rubin Observatory's LSST, these projects reflect humanity's quest to map the cosmos. The LSST represents the culmination of centuries of this cartographic tradition.

3.2. Linear Artifacts in Astronomical Images

This section covers information about transient streaks that appear in images taken for astronomical research, from the astrophysical processes that create them to data that capture the momentary appearance of such objects. It starts with a subsection on meteors and their influence on research. The second subsection goes further to the publicly available survey data of the Sloan Digital Sky Survey and the FITS file format at the heart of most modern astronomy imagery. Attention is finally turned to examples of meteor trails and some other line-like structures that may appear from other causes.

3.2.1. Meteors and Meteor Trails

According to the revised definitions of meteoroids and meteorites by Rubin and Grossman [18], meteoroids are "a 10 μm to 1-meter-size natural solid object moving in interplanetary space," and meteorites are "a natural solid object larger than 10 μm in size, derived from a celestial body, that was transported by natural means from the body on which it formed to a region outside the dominant gravitational influence of that body, and that later collided with a natural or artificial body larger than itself". Meteors [18] are streaks of light produced when meteoroids,

usually fragments of asteroids or comets, enter the Earth's atmosphere at high velocity and burn up because of frictional heating. Although meteoroids are fairly small compared to other celestial objects, traveling at tens of thousands of miles per hour, many meteoroids generate so much heat that, in addition to glowing bright, they sometimes vaporize in a flash. Most observers would usually experience a meteor as a bright streak of fast movement across the night sky. Though meteors exist only for fractions of seconds, they have held human fancy from auspicious portents to nightly wonders but are natural demonstrations of the processes shaping our solar system. Meteor showers or meteor storms are commonly known events that occur when Earth passes through streams of cosmic debris that burn up in the upper atmosphere. Two of the most well-known annual meteor showers, the Perseids and Leonids, reach maximum activity in August and November, respectively. Both events have been seen to regularly amaze observers with spectacular bursts of meteors.

The Leonids [19] are an annual prolific meteor shower from the comet Tempel–Tuttle, which has historically produced meteor storms every 33 years or so. They obtain their name from the apparent radiant of the constellation Leo, from which the meteors appear to emanate. As Earth passes through streams of solid particles that are propelled when the frozen gases of the comet boil off under solar heat, Leonid meteoroids enter our atmosphere. Larger fragments, about 10 mm in diameter and weighing about 0.5 g, can produce quite bright meteors, and the average Leonid shower delivers 12 to 13 [20] tons of material to Earth every year. Leonids can produce storms exceeding 1,000 meteors per hour, sometimes as many as 100,000 meteors per hour, in striking contrast to the few meteors per hour typically seen during everyday conditions.

The Perseid [22] meteor shower originates from what is known as the "Perseid cloud," which is a stream of cometary debris aligned with the orbit of Comet Swift-Tuttle, which has an orbital period of 133 years. Most of the particles in this stream have been there for about a thousand years. Still, a fresher dust filament ejected in 1865 can produce an early secondary peak in activity about one day before the main shower maximum. In the region intersecting Earth's orbit, the cloud extends approximately 0.1 astronomical units (AU) in cross-sectional width and approximately 0.8 AU along Earth's orbital path, shaped over time by recurrent gravitational interactions with the planet. Typically visible from mid-July, Perseids reach peak rates between 9th and 14th of August, depending on Earth's relative position within the debris stream. The hourly meteor counts can reach more than 60 at maximum under favorable conditions. Although many meteoroids reach after dawn, they cannot be seen in daylight. Those observed before midnight tend to travel at shallow angles through the upper atmosphere and so may produce longer trails akin to bright fireballs; most Perseids vaporize at altitudes above about 80 kilometers.

In some contexts, meteor occurrences can become serious challenges, such as creating a nuisance for scientific and technological endeavors. Meteor showers can fill the atmosphere with huge volumes of cosmic debris, causing bursts of bright light to flash across the sky and interfere with long-exposure astronomical imaging. Scientists have to go through contaminated data to isolate legitimate signals, which may disrupt scientific research when the cosmic events under study are faint or transient. In addition, high-velocity meteoroids can po-



Figure 5: Image of a Leonid Meteor by Navicore [21]



Figure 6: Image of Perseids by Ahmed abd Elkader Mohamed [22]

tentially threaten orbiting satellites and the International Space Station since small fragments, at a speed of tens of kilometers per second, might damage sensitive instrumentation or degrade its surface.

A notable example, launched on 12 July 1989 by the European Space Agency [23], Olympus-1 was among the largest civilian communications satellites ever put into geostationary orbit. It was used as a testbed for communications technology and broadcasting services. Onboard the spacecraft were several payloads that tested transmission and directed broadcasting. On 11 August 1993, during the peak of the Perseid meteor shower, Olympus-1 suffered an unexpected control anomaly attributed to a meteoroid impact. The incident caused the satellite to lose stabilization and enter an uncontrolled spin, eventually decommissioning later that year. This incident then underlined how vulnerable spacecraft were to meteoroid collision and brought new interest in protective measures and mitigation strategies for space assets operating in orbit exposed to a meteoroid and debris environment.

Severe meteor showers can also, for brief periods, interfere with radio communication systems because the ionized trails left by burning meteors alter the signal's propagation. The observations of Price and Blum [24] in 1998 using the ELF/VLF (Extremely Low Frequency/Very Low Frequency) method provide evidence that a considerable fraction of meteors, usually not observable optically, contribute to ionospheric disturbances capable of affecting radio communications. These scientists recorded a peak flux of 15,000 meteors per hour in the ELF/VLF band, while only 350 per hour were measured optically; it had been learned that smaller and dimmer meteors generate detectable electromagnetic pulses. Knowing this, the temporal coincidence between the peak ELF/VLF counts and the time of the optical maximum, five to ten minutes apart in this case, confirms that these signals are indeed of meteoric origin, as does the fact that no such pulses were detected on nights with no heightened optical activity. It also detects a secondary peak approximately 90 minutes before the optical maximum, which may suggest that bursts of "subvisible" meteors often precede brighter, more easily observable events. These findings showed that even meteors below the standard optical limiting magnitude can contribute to ionospheric ionization, showing that severe meteor showers can interfere with radio communication by changing its propagation paths.

Although meteors are generally considered natural marvels, these examples show that they can personify a nuisance in exact observational and technological efforts. Because these large-scale surveys, like the SDSS, aim to achieve clear, uncontaminated observations of stars, galaxies, and other sources, images that contain meteor trails are generally excluded from further analysis. Meteor trails have appeared in CCD frames, causing transient streaks considerably reducing various artifact photometric and morphological measurements. However, SDSS flagging (discarding) some frames with apparent meteor trails enhances confidence that their data results are reliable or consistent for future surveys in mapping space without interference with transient events of meteors. Although they might be less spectacular than other more massive bodies and complex phenomena in space, meteors nonetheless receive scientific attention. They represent the direct representatives of the more limited mass of debris that circulates our solar system and, as such, offer concrete evidence about the compositional and dynamical processes by which the Sun has formed. Chemical analyses of meteorites inform

us about primordial materials that predate the Earth and are carriers of important clues to the nature of the early solar nebula.

3.2.2. SDSS Data Releases and FITS Files

The SDSS Science Archive Server (SAS) provides access to astronomical data products, including images, spectroscopy, and calibrated catalogs. The releases of its data products are made in a series of incremental Data Releases (DRs), each building upon previous ones with new observations, expanded sky coverage, additional spectroscopic targets, and improved data-processing pipelines. Below is a brief overview of how SDSS data releases have evolved and what each of them included according to a series of articles published in *The Astronomical Journal* [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39].

- **Early Releases (DR1 to DR7)**

DR1 (2003) was the first public data release that provided images and spectra for a substantial portion of the sky, along with catalogs of extracted objects. [25] DR2 through DR7 continued to expand sky coverage, adding more imaging data in the u, g, r, i, and z filters and a collection of spectroscopic measurements of galaxies, quasars, and stars. These releases also incorporated improvements to the data reduction and calibration pipelines. [26] [27] [28] [29]

- **SDSS-III Era (DR8 to DR12)**

DR8 introduced a recalibration of all previously released imaging data, significantly improving photometric accuracy across the survey footprint. [30] DR9, DR10, and DR11 gradually released data from the Baryon Oscillation Spectroscopic Survey (BOSS), focusing on obtaining redshifts of galaxies and quasars to map the universe's large-scale structure more precisely. DR12 (2014) was the final release of SDSS-III, encompassing complete BOSS data, including spectra and improved catalogs, as well as ancillary science programs (e.g., SEGUE-2 for stellar spectroscopy). [31] [32] [33]

- **SDSS-IV Era (DR13 to DR17)**

DR13 (2016) began the SDSS-IV project, which included several major surveys: eBOSS (extended BOSS), MaNGA (Mapping Nearby Galaxies at APO), and APOGEE-2 (infrared stellar spectroscopy). DR13 also provided new reductions for previously collected data sets and improved data access tools. [34] DR14 (2017) and DR15 (2019) continued to release updated eBOSS, MaNGA, and APOGEE-2 data, offering additional functionality in their data-access portals and more advanced data products such as MaNGA data cubes (IFU spectroscopy of nearby galaxies). [35] [36] DR16 (2019) contained final data from the eBOSS survey and updated MaNGA and APOGEE-2 data products, including expanded sky coverage and deeper spectroscopic samples. [37] DR17 (2021) marked the final release of SDSS-IV. It contained complete data sets from eBOSS, MaNGA, APOGEE-2, various ancillary programs, refined calibration procedures, and final catalogs. [38]

- **Beyond DR17: SDSS-V**

With the close of SDSS-IV, the survey moved to SDSS-V, which has its own set of new and ongoing programs, including spectroscopic monitoring of various types of objects and infrared exploration, the first of its releases being the DR18 (2023) [39]. Future data releases under SDSS-V are expected to continue the tradition of incremental improvements, expansion of sky coverage, and new data products.

Each data release increases the volume of imaging and spectroscopic data and typically refines data processing and calibration approaches. These upgrades help ensure that scientists have access to the best possible measurements and catalogs for a wide range of astronomical research, from Galactic archaeology to cosmological structure formation. All past SDSS data releases remain publicly accessible through SAS and other portals; most of these releases contain some form of information stored in FITS files.

In a conference proceeding from 1979, presented in a volume titled *Image Processing in Astronomy* on page 445 lies an early discussion of the FITS image format in the form of paper called *FITS: A Flexible Image Transport System*. In the paper, Wells, Greisen, and Harten [40] introduce the Flexible Image Transport System (FITS) as a standard format for exchanging astronomical data on magnetic tapes among various observatories. Historically, institutions have been using unique software and data formats, making it challenging to share information between them without translating file structures and rewriting code. FITS was conceived as a self-documenting, human-readable, and machine-readable format to address this problem. It accommodates n-dimensional, regularly spaced data arrays (e.g., images, spectra, and other data products) and metadata describing the data. By defining a common and straightforward structure, FITS prevents the need to endlessly adapt software to the internal conventions of each institution.

The paper describes the organization of the FITS header. Each FITS file contains a header that follows a card-based system consisting of 80-character lines (referred to as 'cards'), each containing specific keywords and values. Essential metadata includes the dimensions of the data array (NAXIS), the number of bits per pixel (BITPIX), and whether the file conforms to the FITS standard (SIMPLE). Additional lines may follow for optional parameters: telescope names, coordinate system definitions, observation dates, scaling factors, etc. All of these components use ASCII text, so the headers are readable by humans and easily parsed by software.

A minimum set of header keywords (SIMPLE, BITPIX, NAXIS, NAXISn, and END) must appear in the first header record for any FITS file so that even the most basic programs can parse and process the data. Beyond these mandatory keywords, the authors also encourage the addition of several other keywords for use in advanced interpretations. For example, keywords such as CRVALn (reference coordinate value), CDELTn (coordinate increments), and CTYPEn (coordinate axes types) can be used to describe the spatial or spectral properties of the data. This balance between a core standard and the possibility to expand the data is what makes FITS flexible for different use cases, from radio arrays to spectral line observations.

The paper concludes that the FITS format provides a mechanism for sharing n-dimensional

array data among institutions without the loss of information or the need for custom adaptation. Its simple structural rules, particularly the header plus data approach, were designed to accommodate future advances in this field. Wells, Greisen, and Harten emphasize that FITS is not only for astronomers; the generality of the format means that it could be applied to other fields dealing with large data sets that require flexible and precise metadata.

The products among the SDSS data releases relevant to this study are the FITS files that store imaging frames from individual observations conducted during SDSS runs, specifically the frames collected in DR12. These frames capture data in one of the five broad photometric filters. Each frame file follows a structured naming convention that includes the specific run identification, camera column, filter band, and field number. These elements identify the region of the sky recorded in any given exposure. The files use bz2 compression and, once uncompressed, can be opened with software such as SAOImage DS9 or Python's Astropy package.

The structure of each file comprises of the pixel data, stored as intensity counts, and an accompanying header containing metadata. The header encodes information such as the World Coordinate System, exposure time, air mass, and timestamps. This approach to data storage allows for different image-processing and photometric tasks, tying each frame to precise celestial coordinates. Through SAS, various users can construct their own surveys or focus on individual targets.

One example illustrating these conventions is a frame file named *frame-g-004822-5-0011.fits.bz2*. In this case, the prefix "frame" means a raw CCD exposure rather than a combined or processed image. The letter g indicates that the data were collected through the g-band filter, which captured a specific portion of the optical spectrum. The six-digit number '004822' refers to the identification of the run, essentially the session in which the data were collected, while the single-digit '5' signifies the camera column. Finally, '0011' denotes the field number within that run and camera column. Figure 7 shows the contents of the frame file exported from SAOImage DS9 after applying a filter to improve star visibility.

Listing 1 is a short Python session demonstrating how to open a FITS file and inspect its structure using the *astropy.io.fits* module. First, we import the module and open the file, creating an HDUList object. We then call *hdul.info()*, which displays a summary of each Header Data Unit (HDU) within the file *frame-g-004822-5-0011.fits*. The HDUList object of this specific file shows that the primary HDU is a 2D image (2048 × 1489) stored as 32-bit floats, the second HDU is a 1D image extension of length 2048, also stored as 32-bit floats, and the next two are binary tables containing various columns and their corresponding data types. This metadata provides a quick overview of the arrangement of the different components within the FITS file.

Listing 2 shows a continuation of the Python session by examining the primary HDU. The information that is given in *primary_hdu.header* specifies that the HDU contains a two-dimensional image of dimensions 2048 × 1489 pixels (NAXIS1 = 2048, NAXIS2 = 1489) stored in 32-bit floats (BITPIX = -32). The flux units are given in nanomaggies (BUNIT = 'nanomaggy'), and the header includes observational information such as the observation date (DATE-OBS = 2004-09-12) and the telescope pointing coordinates (RA = 305.79314, DEC = -22.074210).

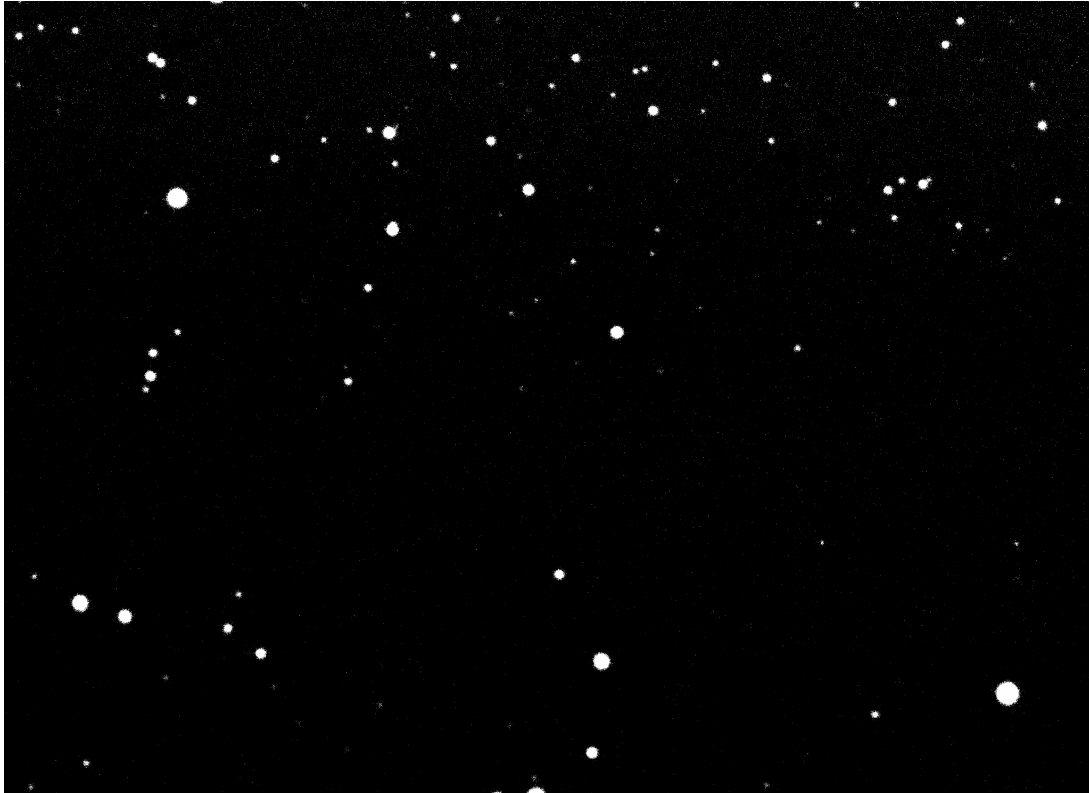


Figure 7: frame-g-004822-5-0011.fits adjusted using Z-scale, SDSS Data Release 12 [41]

World Coordinate System keywords (CRVAL1, CRVAL2, CD1_1, CD1_2, etc.) indicate a fully defined astrometric solution. Calibration parameters such as NMGY and NMGYIVAR enable the conversion between pixel counts and physical flux. Inspection of *primary_hdu.data* confirms that the HDU data is a 1489×2048 NumPy array containing float32 values, including small negative numbers indicative of background subtraction or other reduction steps.

```

1 In [1]: import astropy.io.fits as fits
2 In [2]: hdul = fits.open('frame-g-004822-5-0011.fits')
3 In [3]: hdul.info()
4 Filename: frame-g-004822-5-0011.fits
5 No.      Name      Ver   Type      Cards  Dimensions  Format
6  0  PRIMARY      1  PrimaryHDU    96  (2048, 1489)  float32
7  1                1  ImageHDU      6  (2048,)      float32
8  2                1  BinTableHDU   27  1R x 3C      [48896E, 2048E, 1489E]
9  3                1  BinTableHDU   79  1R x 31C     [J, 3A, J, A, D, D, 2J, J, D,
  ↵  D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, E, E]

```

Listing 1: Opening a FITS file and displaying the header data information

```

1 In [4]: primary_hdu = hdul[0]

2 In [5]: primary_hdu.header
3 SIMPLE = T /
4 BITPIX = -32 / 32 bit floating point
5 NAXIS = 2
6 NAXIS1 = 2048
7 NAXIS2 = 1489
8 EXTEND = T /Extensions may be present
9 BZERO = 0.00000 /Set by MRD_SCALE
10 BSCALE = 1.00000 /Set by MRD_SCALE
11 TAI = 4601680593.08 / 1st row - Number of seconds since Nov 17 1858
12 RA = 305.79314 / 1st row - Right ascension of telescope boresigh
13 DEC = -22.074210 / 1st row - Declination of telescope boresight (d
14 ...
15 DATE-OBS= '2004-09-12' / 1st row - TAI date
16 TAIHMS = '04:36:33.07' / 1st row - TAI time (HH:MM:SS.SS) (TAI-UT = appr
17 COMMENT TAI,RA,DEC, SPA,IPA,IPARATE,AZ,ALT,FOCUS at reading of col 0, row 0
18 ORIGIN = 'SDSS '
19 TELESCOP= '2.5m '
20 TIMESYS = 'TAI '
21 RUN = 4822 / Run number
22 FRAME = 19 / Frame sequence number within the run
23 CCDLOC = 55 / Survey location of CCD (e.g., rowCol)
24 STRIPE = 291 / Stripe index number (23 <--> eta=0)
25 STRIP = 'N ' / Strip in the stripe being tracked.
26 ...
27 CUNIT1 = 'deg ' /Units
28 CUNIT2 = 'deg ' /Units
29 CRPIX1 = 1025.00000000 /X of reference pixel
30 CRPIX2 = 745.000000000 /Y of reference pixel
31 CRVAL1 = 304.748702242 /RA of reference pixel (deg)
32 CRVAL2 = -22.0319479594 /Dec of reference pixel (deg)
33 CD1_1 = -7.98875046723E-05 /RA deg per column pixel
34 CD1_2 = 7.55601500338E-05 /RA deg per row pixel
35 CD2_1 = 7.55953774176E-05 /Dec deg per column pixel
36 CD2_2 = 7.99170527941E-05 /Dec deg per row pixel
37 ...
38 RERUN = '301 ' / rerun
39 HISTORY SDSS_FRAME_Astrom: Astrometry fixed for dr9 Thu Jun 21 07:49:33 2012

40 In [6]: primary_hdu.data
41 Out [6]:
42 array([[ -0.01843262, -0.03240967, 0.01426697, ..., 0.01654053,
43         0.00730896, -0.02041626],
44        [ -0.01374817, 0.00493622, -0.04174805, ..., -0.00655365,
45         0.01193237, -0.00193596],
46        [ 0.02828979, 0.03295898, 0.00492859, ..., -0.01118469,
47        -0.04815674, 0.04888916],
48        ...,
49        [ 0.00999451, 0.0333252 , 0.00532532, ..., -0.00146103,
50        -0.0199585 , -0.03845215],
51        [ -0.02737427, -0.03668213, 0.02865601, ..., -0.01531982,
52        -0.0199585 , -0.00608063],
53        [ -0.0133667 , -0.00868225, -0.02734375, ..., -0.01069641,
54        -0.03845215, -0.03381348]], dtype='>f4')

```

Listing 2: Displaying the header data information of the primary HDU

3.2.3. Linear Artifacts in FITS Images

There are several astrophysical, and sometimes artificial, reasons why linear features might occur in a FITS image taken by the SDSS. A meteor creates streaks across the field of view during exposure because of rapid motion across the sky. As noted by Bektešević and Vinković [1], satellites that cross the field of view appear to produce similar types of linear artifacts. These features often have uniform brightness or periodic changes in brightness if the object is rotating. Comets could also show trails if their apparent motion through space in a given exposure time is large enough.

Non-astronomical sources of linear features include cosmic rays that strike the telescope. Depending on the telescope's angle, these energy particles can yield bright streaks across the image or other point-like features. Internal reflections within the telescope's optical system can also produce linear features, which may be recognizable by their tendency to lie parallel or perpendicular to bright objects in the field; in imaging, this phenomenon is also known as ghosting.

If the data contamination is too significant, FITS files containing linear features become unusable for research regarding the intended astronomical targets. Such linear features can distort the light profiles of stars, galaxies, or other objects, making it hard to extract valid photometric or spectroscopic measurements. The features also introduce signal interference that complicates image analysis; where a linear feature intersects key regions of interest, the affected data may need to be excluded altogether.

The following section presents some examples of trails captured in FITS images of DR12 of the SDSS [41], both of meteors and satellites crossing the field of view during the exposures. Depending on the object's speed, trajectory, and reflectivity, they can appear as linear streaks of various widths, brightnesses, and uniformities.

Figure 8 shows a narrow diagonal streak across the image from the top left to the bottom right. This streak is reasonably constant in brightness and width, a typical signature of an object in fast motion during exposure. Given that it is fairly even in illumination, it would suggest a steady release of light over the exposure. Still, slight variations along the trail reflect minor fluctuations in the object's luminosity as it passed through the camera's field of view. These trail characteristics are indicative of those produced by meteors.

Figure 9 shows the same diagonal streak, now plotted using a reduced intensity that highlights the apparent faintness of the streak. By reducing the overall brightness, the translucent nature of this specific streak becomes more evident, as do the minor variations in its apparent width or brightness. These two figures display the minutely varying brightness in the meteor profile, although the illumination during its passage is reasonably consistent.

The bright, nearly vertical linear feature slightly to the right of the center in Figure 10 is a satellite trail produced by a satellite passing through the field during exposure. A satellite trail is typically a smooth, unbroken streak since the satellite is in sunlight at a more or less constant level as it moves across the frame.

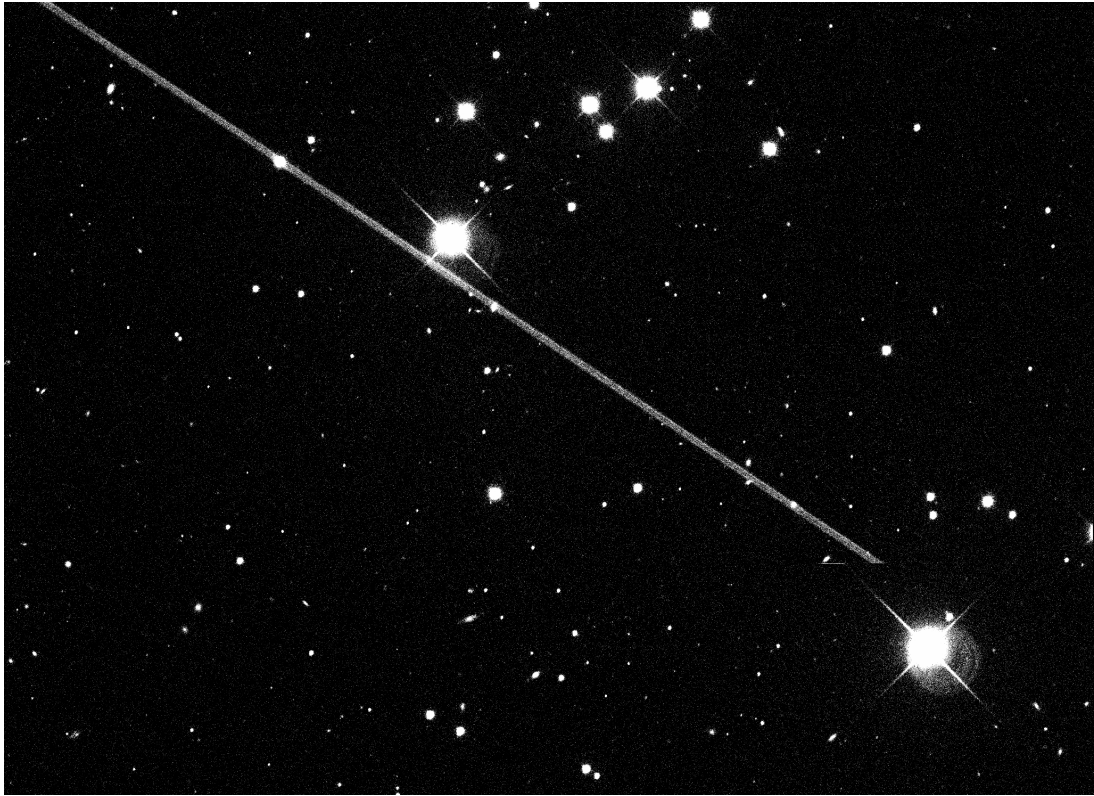


Figure 8: frame-g-000109-5-0090.fits, showing a meteor trail [41]



Figure 9: frame-g-000109-5-0090.fits with reduced intensity [41]



Figure 10: frame-g-000109-5-0125.fits, showing a satellite trail [41]

Figure 11 shows the same streak but with a reduced brightness intensity. This indicates that the streak has an even width and brightness along its length, and its linear profile is uniform, a feature of satellite trails that passed through during the exposure. Compared to the meteor trail in Figure 9, this trail does not show any translucency, which is shown to be the main difference between the meteor trail and the satellite trail.



Figure 11: frame-g-000109-5-0090.fits with reduced intensity [41]

Figure 12 presents a thin diagonally oriented streak with a very low brightness of what appears to be a meteor that is either further away or smaller in size. The narrow width of the trail shows only a short or relatively dim emission of light, but the consistent alignment and continuity would point out the steady motion of the object. This modest linear feature is relatively thin, a typical feature of satellite trails. However, this trail has slight variations in brightness along its path, and its width varies ever so slightly; these are indicative features of meteor trails.

Figure 13 shows a noticeably wide streak oriented diagonally, resulting from a bright meteor that passed within the camera's field of view. This width suggests that the camera experienced partial saturation, which made the trail appear broader compared to other fainter events. The uniform brightness along the band demonstrates the presence of a steadily radiating meteor during an intense flash as it traversed the captured sky. The steady decrease in brightness along the trail profile, as opposed to a flat and sudden drop in the brightness that satellites demonstrate, shows that this particular trail is a meteor.

Figure 14 shows a diagonally oriented streak, which, while quite faint, occupies a vast band throughout the frame. This broad, diffuse appearance would suggest that the meteor was bright enough to have spread out or saturated the imaging system, yet not as bright as some

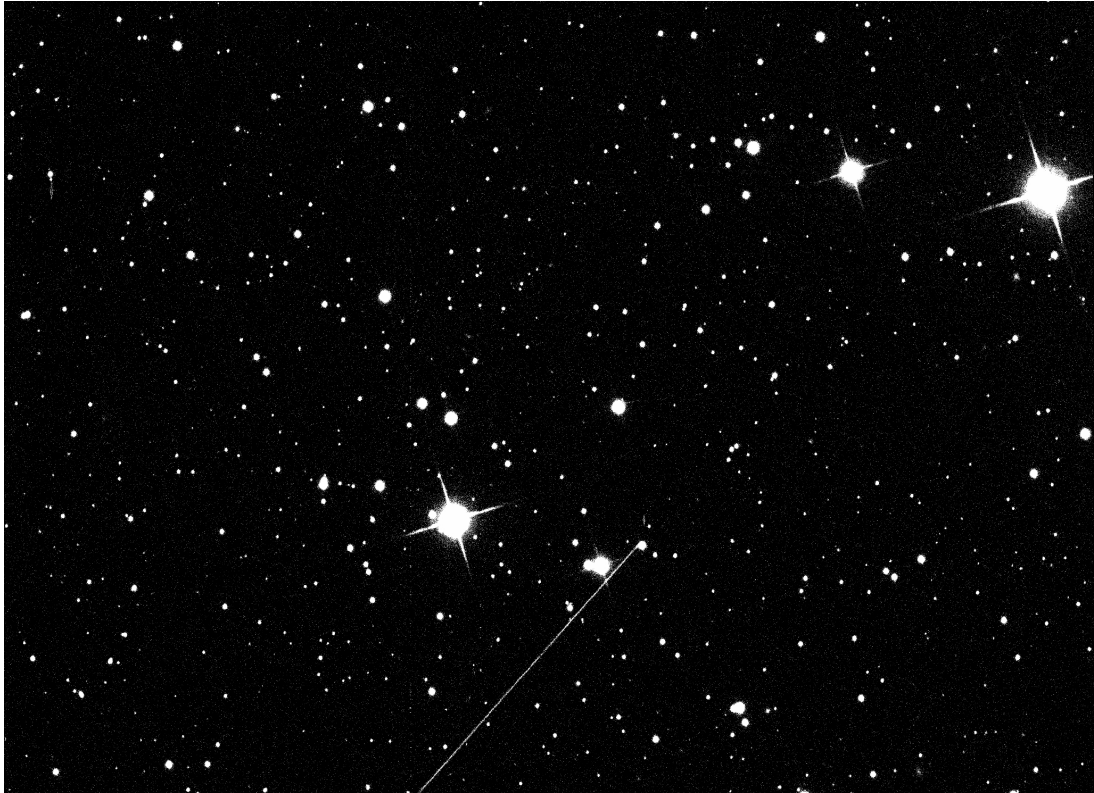


Figure 12: frame-z-008149-5-0096.fits, containing a thin, faint meteor trail [41]

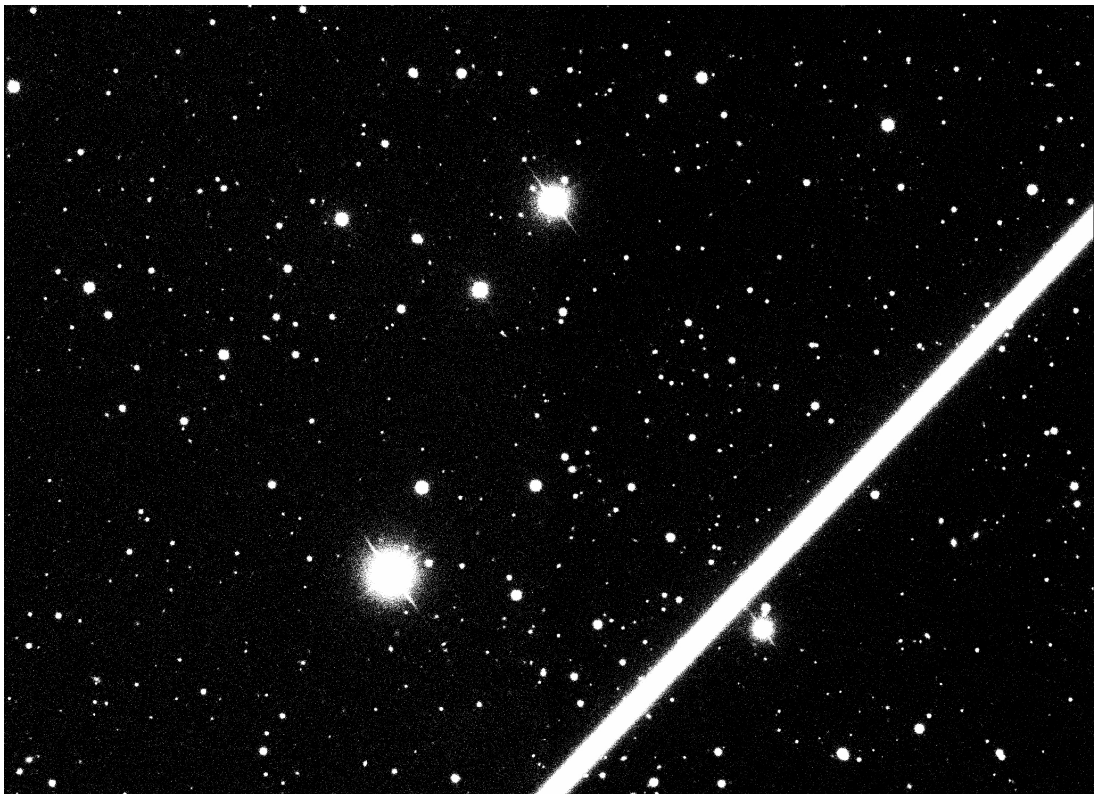


Figure 13: frame-i-006177-3-0331.fits, containing a wide, bright meteor trail [41]

other examples. Its consistent thickness and uninterrupted continuity point to a smooth and uniform passage across the field during exposure. This example shows that although a meteor

trail can be extensive, it does not mean it must also be bright and luminous.

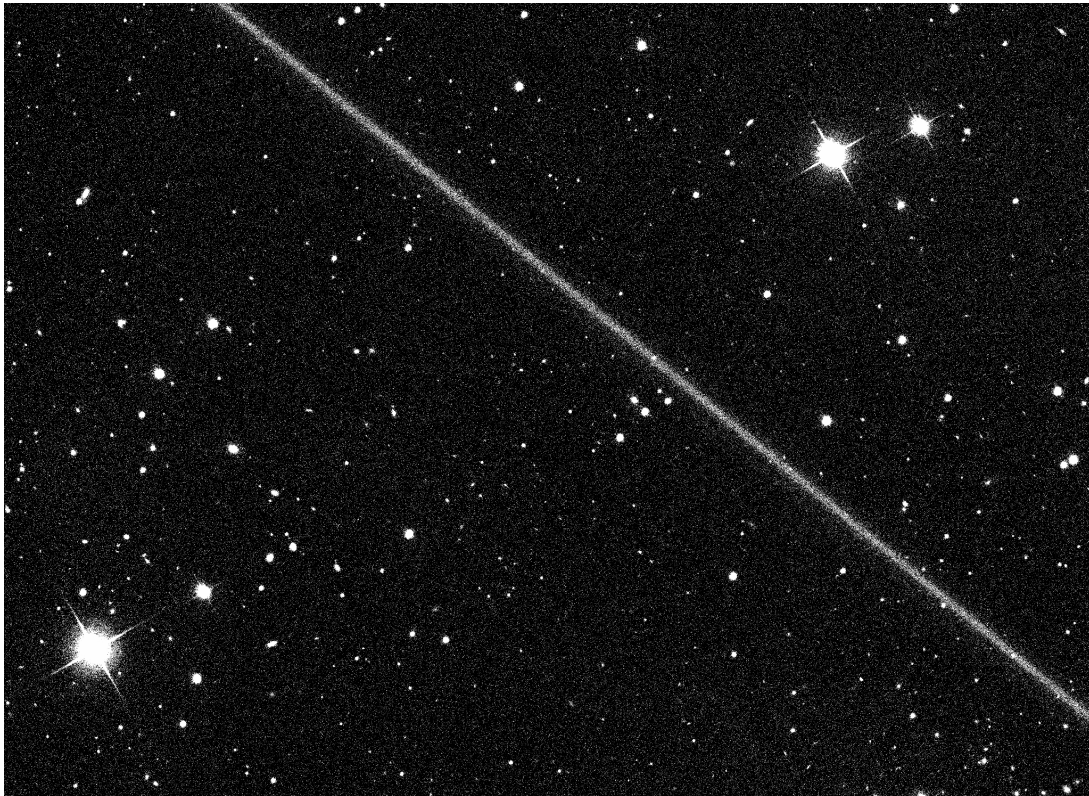


Figure 14: frame-r-002566-1-0334.fits, containing a faint although wide meteor trail [41]

Figure 15 diagonally across the center of the image shows a trail exhibiting subtle repeated variations in brightness along its length. This suggests that the object in question is rotating; the reflected sunlight from an elongated rotating satellite varies periodically, with its rotation causing brightness fluctuations as it passes over the camera exposure. The distinctly segmented appearance of the trail indicates a stable rotation rate, with fluctuation in brightness occurring regularly during the exposure. This example shows that although the trail seems translucent at points, it is still most likely a satellite that made it due to this apparent rotation and continuity. The trail suggests that the satellite that made it has one side that reflects light more brightly and another that does not.

The trail in Figure 16 extends diagonally across the field of view, displaying a unique streak contrasting sharply against the background. It appears to be the trail of a meteor burning up very close to the Earth's surface; this is indicated by the width of the trail and by its translucency. Its width appears consistent along its length, suggesting a uniform brightness distribution. The upper portion of the trail seems slightly more luminous, which may indicate variations in the meteor's intensity as it traversed the camera's field.

Figure 17 shows another meteor trail with a unique property. In several segments of the trail, faint extensions appear to branch away from the primary streak, creating the impression of small "tails" being shed from the main body.

Although the examples here illustrate some characteristic features that may help distinguish between meteors and satellites, such trails can frequently resist easy classification



Figure 15: frame-z-003355-6-0110.fits, containing what appears to be a trail of a rotating satellite [41]



Figure 16: frame-u-002243-5-0072.fits, containing a very wide meteor trail [41]

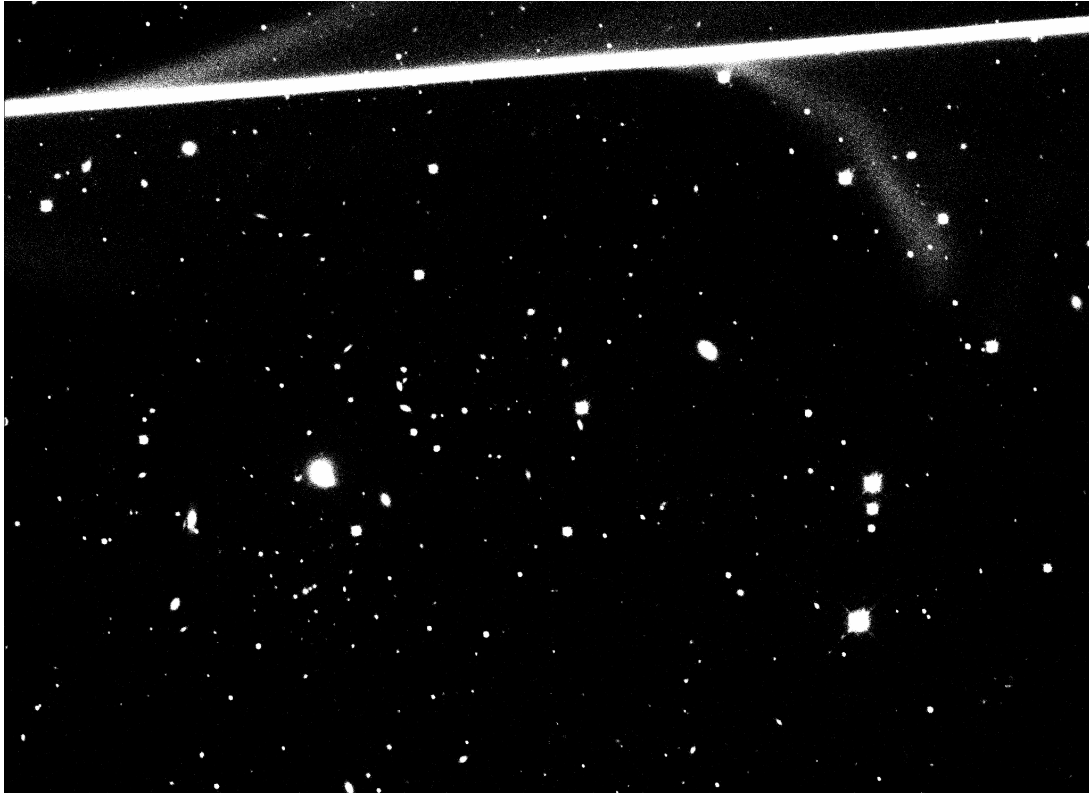


Figure 17: frame-r-000094-5-0168.fits, containing a meteor trail with branching tails [41]

through mathematical models. Differences in brightness, speed, and trajectory sometimes fall within an overlap where a classification may not be clear-cut. However, most trails show tell-tale features, such as consistent thickness or periodic fluctuations, which enable them to be correctly identified. These examples provide a basis for the trail classification process.

3.3. Identification and Classification of Linear Artifacts

This section presents a framework for analyzing linear artifacts in astronomical images, from their detection and brightness profiling to their classification as a meteor or non-meteor trail. It is organized into three subsection, each addressing a step in the analysis of linear artifacts. In the first subsection, an automated linear artifact detection methodology is introduced. The second subsection focuses on the profiling process that extracts brightness profiles and computes metrics for a given trail. The third and final subsection presents a neural network classification model that distinguishes genuine meteor trails from non-meteor trails. The processes in this section combine into a pipeline framework designed to automatically detect, profile, and classify trails as they appear in astronomical images.

3.3.1. Automated Linear Artifact Detection

The Linear Feature Detection Algorithm (LFDA) developed in the work of Bektešević and Vinković [1] addresses the challenge of automatically identifying linear artifacts in astronomical

images, which are generally dismissed as noise. Building on the Hough transform, which, according to Fischer [42] is a technique that can be used to isolate features of a particular shape within an image, the LFDA adds steps to remove bright sources (e.g., stars and galaxies) so that linear features dominate in Hough space, reducing false positives and making it possible to detect fainter streaks that standard methods would miss.

Automatic detection of linear features or artifacts may not be strictly necessary for data sets of fewer than 100,000 images due to manageable manual inspection times. Still, the value of such a tool as LFDA becomes evident when dealing with millions of images. In these scenarios, the algorithm must be highly efficient and should take less than a second per image to avoid a long processing time. Although the parallel nature of LFDA is a step toward achieving this speed, issues remain, such as data transfer, memory constraints, and false detections. For example, a 1% error rate could still leave tens of thousands of images requiring manual verification. In contrast, tightening the detection thresholds could miss faint features. LFDA was developed to incorporate it into a pipeline where strategies need to be employed for parallelization and data management to meet both the performance and accuracy demands in sky surveys of larger scales.

The LFDA workflow begins by stripping out known objects and then searching for bright and, if necessary, dim linear features. In each pass, morphological operations like erosion and dilation eliminate noise and accentuate dominant features, while histogram equalization and brightness enhancement boost faint trails. Edge detection and contour detection isolate candidate shapes, which are further analyzed using minimum-area rectangles. In LFDA, Canny edge detection is used, which, according to Fischer [43], takes a grayscale image as input and produces an image showing the positions of tracked intensity discontinuities as output. In the final steps of the LFDA, the Hough transform is used to identify linear structures, although it can produce false positives if discs of stars or galaxies remain in the image.

The first step implemented in the algorithm is to remove known objects in the image. Object removal can be almost trivial when object catalogs or image differencing products exist (as they do in LSST), leaving primarily linear features and other artifacts behind. The LFDA incorporates an object removal module, which removes objects based on survey data (e.g., SDSS photoObj files containing objects' data and their positions). In SDSS, the approach involves filtering out objects by magnitude caps and observed frequency, then masking them with simple squares scaled by each object's Petrosian radius, that is, the object's radius based on its brightness profile. Although this method is lightweight in computational terms, it still preserves transient linear features, avoiding the overmasking of potential trail detections but substantially cleaning up the image for subsequent steps.

Minimal image processing is used to detect bright linear features to avoid excessive contrast that could introduce noise. The images are initially converted to 8-bit depth using a floor function, then undergo histogram equalization followed by a dilation step. After that, Canny edge detection is applied with a weaker threshold to identify the objects' borders. The contours of these edges are fitted with minimum-area rectangles filtered by length and elongation criteria. Potential lines for both the processed image and its version with overlaid rectangles are then

derived via the Hough transform. The two sets of lines are compared, rejecting those with different slopes (θ) or positions (r), which helps eliminate common false positives such as star diffraction spikes.

When searching for dim linear features, the LFDA first raises faint pixel values above a minimum threshold and then adds a constant flux, making it so that these subtle trails persist after conversion to 8-bit and histogram equalization. This step accentuates noise, which is then reduced via an erosion-dilation sequence tuned to preserve barely visible lines. The subsequent steps, such as edge detection, contour analysis, and minimum-area rectangle fitting, mirror those used for bright lines. Line sets from the processed and reconstructed images are compared so that spurious detections are left out and the actual faint trails are confirmed.

In tests performed for the LFDA, comparing lines derived from minimum-area rectangles with those from the processed image significantly reduced false positives, especially those caused by partially removed or highly structured objects. Genuine, extended, bright features generally satisfy the constraints and appear consistently in both reconstructions, whereas spurious detections typically do not. Occasionally, an extremely bright, highly structured, or poorly masked object may still generate a false positive that must be manually checked.

False negatives can occur if the masking inadvertently removes too much of a faint trail, is too weak to survive morphological processing, or has brightness dips that break the expected linear profile. However, performance evaluations on various SDSS frames showed that the algorithm could detect lines with low error rates in 0.1 to 0.3 seconds per frame. It can also readily adapt to different observational data sets by adjusting relevant parameters. Including LFDA in survey pipelines, particularly those employing image differencing and object removal, can provide efficient tracking of linear features across vast data volumes and is a step toward the goal of fully automated detection, classification, and extraction of fundamental physical properties of these features.

For this project, the *TrailDetector* class was created that takes conceptual inspiration from the LFDA project but forgoes its more complex steps; this suits the smaller data set of 26,600 FITS files that were noted from the resulting trail detections of the LFDA. Instead of removing all cataloged objects or applying elaborate morphological operations, this class relies on more direct image normalizations, such as ZScale or Percentile scaling, along with straightforward erosion and dilation. By limiting parameters (such as the thresholds for Canny edge detection or Hough transforms), this approach becomes inherently easier to tune for a narrower range of image types. Just as LFDA differentiates between bright and dim features with successive passes and extensive morphological cleanups, this class also performs two main detection modes, "bright" and "dim." It performs an essential step based on Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to handle overlapping detections. As a result, it provides enough functionality to detect trails reliably without the overhead of advanced masking, brightness modeling, or object removal that characterizes the LFDA.

The imports for the *TrailDetector* class found in Listing 3 provide a foundation for the class's capabilities. The standard library *os* is used for file handling, while *numpy* offers numerical operations and array manipulation. *OpenCV* (*cv2*) is central to operations like edge

```

1 import os
2 import numpy as np
3 import cv2
4 import matplotlib.pyplot as plt
5 from astropy.io import fits
6 from astropy.visualization import (
7     ImageNormalize,
8     PercentileInterval,
9     SqrtStretch,
10    ZScaleInterval,
11 )
12 from scipy.spatial.distance import pdist, squareform
13 from sklearn.cluster import DBSCAN

```

Listing 3: Imports for the *TrailDetector* class

```

1 class TrailDetector:
2     def __init__(
3         self,
4         canny_params=None,
5         hough_params=None,
6         dbscan_eps=150,
7         dbscan_min_samples=2,
8     ):
9         self.canny_params = canny_params or {"threshold1": 6, "threshold2": 18}
10        self.hough_params = hough_params or {
11            "rho": 1,
12            "theta": np.pi / 180,
13            "threshold": 250,
14            "minLineLength": 300,
15            "maxLineGap": 150,
16        }
17        self.dbscan_eps = dbscan_eps
18        self.dbscan_min_samples = dbscan_min_samples
19
20        self.merged_lines = []
21        self.best_line = None

```

Listing 4: Constructor of the *TrailDetector* class

detection (Canny) and line detection (Hough transform). *Matplotlib's pyplot* is used to visualize the results. *Astropy's* FITS utilities read astronomical data files, and its visualization tools (*ImageNormalize*, etc.) perform image stretching and normalization. Finally, *scipy* distance routines (*pdist*, *squareform*) and the DBSCAN algorithm of *scikit-learn* enable the class to group detected line segments, producing the final trail candidates of a given image.

The *TrailDetector* class constructor found in Listing 4 accepts various parameters that control the stages of trail detection. By default, it sets Canny edge detection thresholds (*canny_params*) and Hough transform parameters (*hough_params*) to reasonable starting values, ensuring edge and line detection work for typical astronomical images. The DBSCAN parameters (*dbscan_eps* and *dbscan_min_samples*) specify how loosely or strictly the line segments must cluster to be merged into a single trail. Internally, the constructor also initializes *merged_lines* and *best_line* as empty placeholders for the eventual aggregated line segments and the single best trail candidate, respectively.

Much like the LFDA, in the *detect_trails* method of the *TrailDetector* class (found in

```

1 def detect_trails(self, fits_file, save_processed=False,
2   ↪ processed_dir="processed_images"):
3     if save_processed:
4         os.makedirs(processed_dir, exist_ok=True)
5
6     lines_bright = self._attempt_detection(
7         fits_file=fits_file,
8         mode='bright',
9         save_processed=save_processed,
10        processed_dir=processed_dir
11    )
12    if lines_bright:
13        self.merged_lines = lines_bright
14        self.best_line = self._choose_best_line(lines_bright)
15        return [self.best_line]
16
17    lines_dim = self._attempt_detection(
18        fits_file=fits_file,
19        mode='dim',
20        save_processed=save_processed,
21        processed_dir=processed_dir
22    )
23    if lines_dim:
24        self.merged_lines = lines_dim
25        self.best_line = self._choose_best_line(lines_dim)
26        return [self.best_line]
27
28    print(f"No trails detected in {fits_file} after both attempts.")
29    return []

```

Listing 5: The *detect_trails* method of the *TrailDetector* class

Listing 5, two different detection modes (bright and dim) are being tested for a given FITS file to maximize the chance of identifying a potential linear artifact. If a bright linear feature is detected on the first pass, the algorithm selects the best candidate, defined here as the longest line, from the set of discovered lines and returns it immediately. If there are no detections in the bright mode, it proceeds to a second pass, which adjusts the parameters to look for subtler trails, again choosing the best line if multiple candidates are found.

The *_attempt_detection* method found in Listing 6 takes a FITS file and processes it according to a specified "mode," either "bright" or "dim." Depending on the mode, it first applies the relevant steps to emphasize potential trails. After normalizing and transforming the image, the method uses OpenCV's Canny edge detection and the Hough transform to identify line segments that could represent trails. These line segments are then grouped using a DBSCAN process that groups similar lines into a smaller set of merged lines. The final set of lines is returned for further evaluation.

The *_preprocess_bright* method found in Listing 7 enhances the image so that bright linear features are more easily detected. It begins by applying the ZScaleInterval algorithm through ImageNormalize, a technique that scales pixel intensities to balance out extreme brightness values. The image is then converted to an 8-bit format, which makes the image ready for further operations, such as histogram equalization. This equalization redistributes the pixel intensity range to heighten local contrast, helping features stand out. In the end, a 4×4 morphological kernel is used for erosion (which reduces noise and removes smaller objects) and

```

1 def _attempt_detection(
2     self,
3     fits_file,
4     mode,
5     save_processed=False,
6     processed_dir="processed_images"
7 ):
8     with fits.open(fits_file) as hdul:
9         image_data = hdul[0].data
10
11     if mode == 'bright':
12         processed = self._preprocess_bright(image_data)
13     else:
14         processed = self._preprocess_dim(image_data)
15
16     if save_processed:
17         base_no_ext = os.path.splitext(os.path.basename(fits_file))[0]
18         out_name = f"{base_no_ext}_{mode}_processed.png"
19         out_path = os.path.join(processed_dir, out_name)
20         cv2.imwrite(out_path, processed)
21         print(f"Saved pre-processed '{mode}' image to: {out_path}")
22
23     edges = cv2.Canny(processed, **self.canny_params)
24     lines = cv2.HoughLinesP(edges, **self.hough_params)
25     if lines is None:
26         return []
27
28     lines_data = self._process_lines(lines)
29     if len(lines_data) == 0:
30         return []
31
32     merged = self._merge_lines(lines_data)
33     return merged

```

Listing 6: The `_attempt_detection` method of the `TrailDetector` class

```

1 def _preprocess_bright(self, image_data):
2     norm = ImageNormalize(image_data, interval=ZScaleInterval())
3     float_img = norm(image_data)
4     gray_8u = cv2.convertScaleAbs(float_img)
5     equ = cv2.equalizeHist(gray_8u)
6     kernel = np.ones((4, 4), np.uint8)
7     equ = cv2.erode(equ, kernel, iterations=1)
8     equ = cv2.dilate(equ, kernel, iterations=1)
9     return equ

```

Listing 7: The `_preprocess_bright` method of the *TrailDetector* class

```

1 def _preprocess_dim(self, image_data):
2     image_data = np.nan_to_num(image_data, nan=0.0, posinf=0.0, neginf=0.0)
3     norm = ImageNormalize(
4         image_data,
5         interval=PercentileInterval(1, 99),
6         stretch=SqrtStretch()
7     )
8     float_img = norm(image_data)
9     gray_8u = cv2.convertScaleAbs(float_img)
10    equ = cv2.equalizeHist(gray_8u)
11    kernel = np.ones((3, 3), np.uint8)
12    equ = cv2.erode(equ, kernel, iterations=1)
13    kernel = np.ones((9, 9), np.uint8)
14    equ = cv2.dilate(equ, kernel, iterations=1)
15    return equ

```

Listing 8: The `_preprocess_dim` method of the *TrailDetector* class

dilation (which expands the core of the now dilated objects). The resulting image retains prominent bright linear artifacts while substantially reducing noise.

In the `_preprocess_dim` method shown in Listing 8, the image is prepared to highlight faint linear features without overwhelming the image with noise. First, all invalid pixel values are replaced with zeros to avoid skewing subsequent operations. The data is then normalized using a (1–99) percentile interval combined with a `SqrtStretch`, brightening dim signals while keeping the overall range balanced. After converting the resulting data to an 8-bit image, local contrast is enhanced by histogram equalization. Finally, morphological erosion (with a 3×3 kernel) is applied to remove minor artifacts, followed by dilation (with a 9×9 kernel) to restore the essential structure of any residual faint features.

The `_merge_lines` method found in Listing 9 combines lines that share similar positions, endpoints, and orientations into a single representative line. It begins by defining a distance measure that sums the spatial separation of the endpoints of the lines and the angular difference between them. These distances are passed to the DBSCAN algorithm, which groups segments with low mutual distances into clusters. For every identified cluster, the method calculates the mean coordinates of the endpoint, producing a single ‘merged’ line to represent all segments of that cluster.

In addition to the two main processing methods, the *TrailDetector* class also includes several helper methods. The `_process_lines` method takes raw line data from the Hough transform, calculates geometric properties like length and angle for each line, and returns them in a

```

1 def _merge_lines(self, lines_data):
2     def custom_distance(l1, l2):
3         pos_dist = np.hypot(l1[0] - l2[0], l1[1] - l2[1])
4         angle_dist = abs(l1[5] - l2[5])
5         return pos_dist + 50 * angle_dist
6
7     dist_matrix = squareform(pdist(lines_data, metric=custom_distance))
8     clustering = DBSCAN(
9         eps=self.dbscan_eps,
10        min_samples=self.dbscan_min_samples,
11        metric= "precomputed"
12    ).fit(dist_matrix)
13
14    merged_lines = []
15    for label in set(clustering.labels_):
16        if label == -1:
17            continue
18        cluster = lines_data[clustering.labels_ == label]
19        x1_mean = int(cluster[:, 0].mean())
20        y1_mean = int(cluster[:, 1].mean())
21        x2_mean = int(cluster[:, 2].mean())
22        y2_mean = int(cluster[:, 3].mean())
23        merged_lines.append([x1_mean, y1_mean, x2_mean, y2_mean])
24    return merged_lines

```

Listing 9: The `_merge_lines` method of the `TrailDetector` class

```

1 from trail_detector import TrailDetector
2 detector = TrailDetector()
3 detector.detect_trails("frame-g-000250-5-0189.fits")
4 [[1569, 11, 1810, 1478]]
5 detector.plot_trails("frame-g-000250-5-0189.fits")

```

Listing 10: Example usage of the `TrailDetector` class

convenient array. The `_choose_best_line` method selects the longest line from a list of merged candidates as the primary trail of interest. The `plot_trails` method overlays this chosen line on the original FITS image for visualization, and `save_merged_lines` writes the line's coordinates to a text file.

The Listing 10 shows an example of using the `TrailDetector` class. The example starts by importing the `TrailDetector` class and creating an instance without specifying any custom parameters. The `detect_trails` method is then called on the FITS file `"frame-g-000250-5-0189.fits"` and returns a list containing one detected line with coordinates: `[1569, 11, 1810, 1478]`. The list of numbers represents the pixel coordinates (x_1, y_1, x_2, y_2) of the start and end points of the most prominent linear trail found in the image. The original content of the file `"frame-g-000250-5-0189.fits"` can be viewed in Figure 18, the image was altered using ZScale so that the trail is visible. Using the `plot_trails` method with the same FITS file overlays this line on the image, visually representing the detection result shown in Figure 19.

The complete code of the `TrailDetector` class can be found in the appendices.

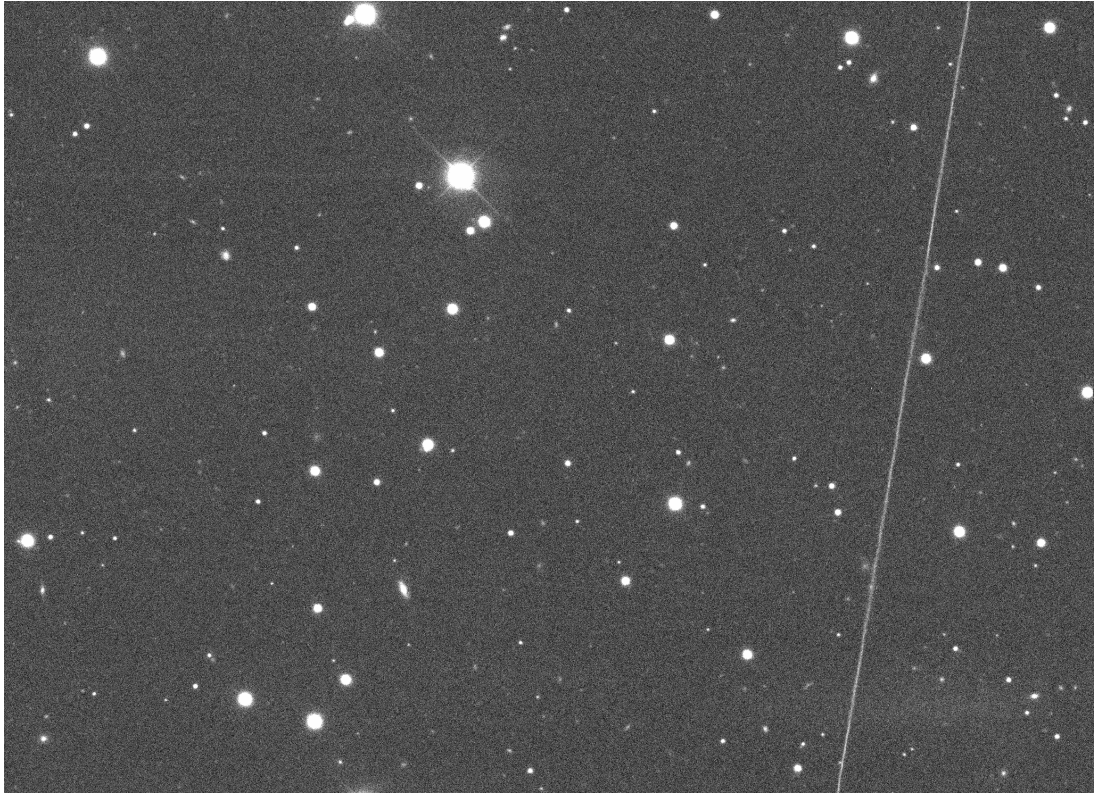


Figure 18: frame-g-000250-5-0189.fits altered using ZScale [41]

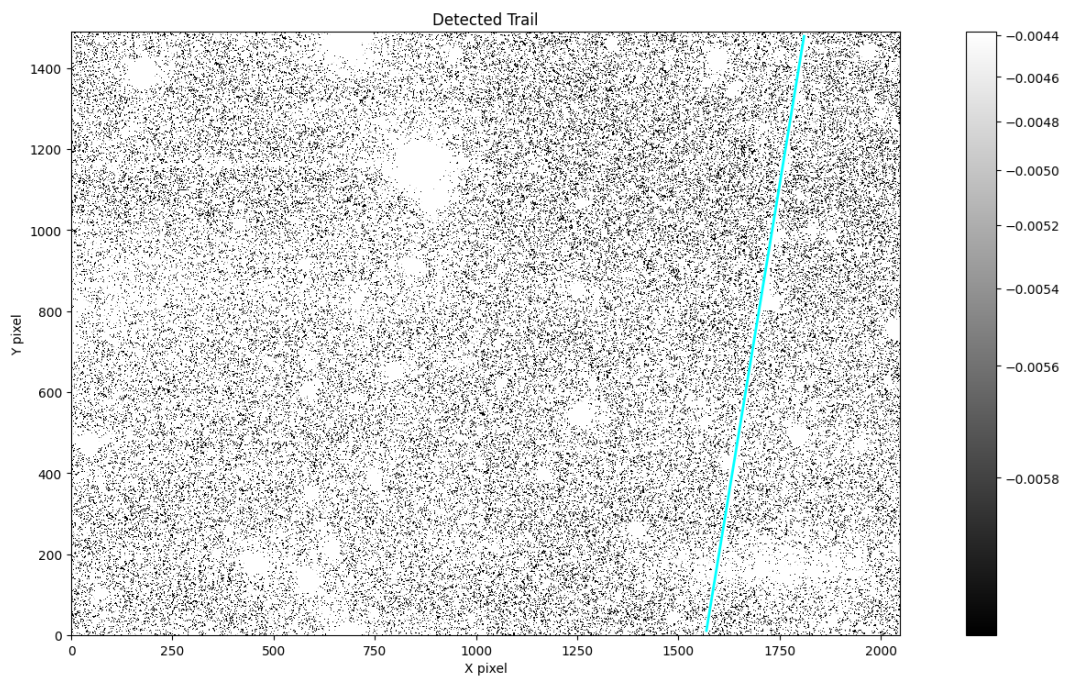


Figure 19: Result of the plot_trails method of the *TrailDetector* class for file frame-g-000250-5-0189.fits

3.3.2. Linear Artifact Profiling

Objects that reflect light in astronomical images can be described using what is commonly referred to as brightness profiles that are closely tied to their physical characteristics like size or light reflectivity. As an example, in the research of Mackey and Gilmore [44], *Surface brightness profiles and structural parameters for 53 rich stellar clusters in the Large Magellanic Cloud*, data from the Hubble Space Telescope was collected for 53 rich Large Magellanic Cloud clusters to derive their structural parameters (e.g., core radii, outer slopes, total luminosities). They observed interesting structural features across the clusters; younger clusters exhibited bumps and dips in their brightness profiles, while older ones showed behavior closely tied to the collapse of their core. The authors found that although most clusters follow a "standard" evolution (maintaining relatively small cores), some develop large, diffuse cores due to their unique stellar populations or external influences. This study is a good example of how brightness profiles can be used to gather information about the structure of celestial objects.

Brightness profiling of celestial objects is essential to modern astronomy, particularly as wide-field surveys like the SDSS and LSST continue to generate vast amounts of data. The brightness profiling of linear artifacts is crucial since this information could give insights into the objects causing these streaks. Profiling moving celestial objects has also been a longstanding practice in astronomy; for instance, in 1987, Jewitt and Meech [45] measured the radial surface brightness profiles of 10 comets at continuum wavelengths using CCD cameras. Despite the expectation that a spherically symmetric, steady-state coma (the envelope around the nucleus of a comet) would have a logarithmic derivative, the actual measurements revealed steeper profiles and a monotonic decrease with increasing distance from the nucleus. Interpreting these results with Monte Carlo models that accounted for solar radiation pressure led the authors to propose that the grains dominating the optical signal were not perfectly coupled to the sublimated gas.

Trail profiling is referred to here as the analysis of each linear artifact's geometric and photometric properties once detected. Examining attributes such as shape, length, width, brightness gradients, and intensity dips or flares forms a basis for differentiating between meteor trails, satellite tracks, or other phenomena. The *TrailProfiler* class was developed for this project, which provides a specialized environment for dissecting the linear trails flagged by automated detection methods (e.g., the *TrailDetector* class). *TrailDetector* locates potential streaks in an image, *TrailProfiler* refines these findings by collecting and organizing further information about each trail's spatial extent.

The *TrailProfiler* class relies on a handful of Python libraries to handle file management, numerical processing, and visualization tasks. The *os* module is used for file and directory operations, such as checking paths or saving outputs. *NumPy* provides array structures and mathematical functions for manipulating pixel data. *Matplotlib*'s *pyplot* is used, much like in the *TrailDetector* class, for the generation of plots, and the *astropy* is used because of its tools for manipulating astronomical images (e.g., *PercentileInterval*, *SqrtStretch*). The *map_coordinates* from *scipy.ndimage* supports interpolation and pixel sampling, which makes it possible to extract specific regions of interest within an image.

```

1 class TrailProfiler:
2     def __init__(self, fits_file, point1, point2, output_dir="trail_profiles"):
3         self.fits_file = fits_file
4         self.point1 = point1
5         self.point2 = point2
6         self.output_dir = output_dir
7         self.image_data = None
8         self.normalized_data = None
9         self.brightness_profiles = []
10        self.line_coordinates = []
11        self._load_image_data()
12        self._ensure_output_directory()
13        self._analyze_perpendicular_lines()

```

Listing 11: Constructor of the *TrailProfiler* class

The constructor of the *TrailProfiler* class shown in Listing 11 accepts a FITS file path along with two coordinates representing the endpoints of the trail under investigation. Upon initialization of an instance of the class, it reads the image data, ensures an output directory exists for saving results, and begins an initial analysis of lines perpendicular to the specified trail. These perpendicular lines extract brightness profiles along the streak and will later be used in the characterization of the trail.

The private method on Listing 12, `_analyze_perpendicular_lines`, samples brightness values along multiple lines perpendicular to the main trail. It divides the trail into segments, computes a perpendicular direction at each segment, and collects brightness measurements across these short "slices." To achieve this, it first derives a perpendicular unit vector based on the trail's endpoints, then uses `map_coordinates` to interpolate intensity values along evenly spaced points across each slice. The method finally normalizes these brightness profiles and stores both the resulting intensity arrays and the endpoint coordinates of each perpendicular line.

The `get_combined_median_profile` method shown in Listing 13 produces a representative brightness curve of the trail by taking the median across the previously calculated brightness profiles. Precisely, it stacks all input profiles into a single array and computes the median value at each index along their length. This resulting profile characterizes the overall intensity distribution of the trail across all sampled perpendicular lines, this mitigates the impact of outliers among individual slices that contain noise or other artifacts. The method can also save the median profile as a plot, including a vertical line indicating the midpoint of the profile (i.e., the intersection with the trail), thereby providing a visualization of the trail's brightness.

Alongside its core logic for analyzing perpendicular brightness slices, *TrailProfiler* includes methods for loading and normalizing FITS data, generating various plots, and calculating the metrics of each trail. The `_load_image_data` method handles reading the FITS file and applying an intensity stretch, `_ensure_output_directory` ensures that output folders exist. Plotting routines like `plot_brightness_profiles`, `plot_divided_image`, and `plot_all_perpendicular_lines` provide convenient visualizations of the trail and its perpendicular sampling lines, whereas `plot_profile_at_index` and `plot_line_on_image` focus on specific individual profiles. Metrics such as the median brightness curve can be retrieved or displayed via `get_combined_median_profile` and `plot_median_profile`.

```

1  def _analyze_perpendicular_lines(self, num_perpendicular_lines=10,
  ↪ short_window_size=100, step_size=0.1):
2      x0, y0 = self.point1
3      x1, y1 = self.point2
4      main_line_length = np.hypot(x1 - x0, y1 - y0)

5      perp_dx = -(y1 - y0) / main_line_length
6      perp_dy = (x1 - x0) / main_line_length
7      step_along_main_line = main_line_length / (num_perpendicular_lines - 1)

8      self.brightness_profiles = []
9      self.line_coordinates = []

10     for i in range(num_perpendicular_lines):
11         t = i * step_along_main_line / main_line_length
12         x_center = x0 + t * (x1 - x0)
13         y_center = y0 + t * (y1 - y0)

14         x_perp_start = x_center - short_window_size * perp_dx
15         y_perp_start = y_center - short_window_size * perp_dy
16         x_perp_end = x_center + short_window_size * perp_dx
17         y_perp_end = y_center + short_window_size * perp_dy

18         num_samples = int(2 * short_window_size / step_size)
19         x_perpendicular_full = np.linspace(x_perp_start, x_perp_end,
  ↪ num_samples)
20         y_perpendicular_full = np.linspace(y_perp_start, y_perp_end,
  ↪ num_samples)
21         perpendicular_coords = np.vstack((y_perpendicular_full,
  ↪ x_perpendicular_full))
22         brightness = map_coordinates(self.image_data, perpendicular_coords,
  ↪ order=3)

23         normalized_brightness = (brightness - np.min(self.image_data)) / (
24             np.max(self.image_data) - np.min(self.image_data)
25         )
26         self.brightness_profiles.append(normalized_brightness)
27         self.line_coordinates.append((x_perp_start, y_perp_start), (x_perp_end,
  ↪ y_perp_end))

```

Listing 12: The `_analyze_perpendicular_lines` method of the `TrailProfiler` class

```

1 def get_combined_median_profile(self, brightness_profiles=None,
  ↪ profile_name="profile", save_median_profile=False):
2     profiles_array = np.vstack(brightness_profiles if brightness_profiles else
  ↪ self.brightness_profiles)
3     median_profile = np.median(profiles_array, axis=0)

4     if save_median_profile:
5         output_file = os.path.join(self.output_dir, f"{profile_name}_median.png")
6         plt.figure(figsize=(12, 8))
7         plt.plot(median_profile, label='Combined Median Profile', color='blue')
8         plt.axvline(len(median_profile) // 2, color='red', linestyle='--',
  ↪ label='Intersection Point')
9         plt.xlabel('Position along perpendicular line')
10        plt.ylabel('Normalized Brightness (0-1)')
11        plt.title(f'Combined Median Profile: {profile_name}')
12        plt.legend()
13        plt.savefig(output_file, dpi=300, bbox_inches='tight')
14        plt.close()
15        print(f"Saved combined median profile plot to {output_file}")

16    return median_profile

```

Listing 13: The `get_combined_median_profile` method of the `TrailProfiler` class

```

1 from trail_profiler import TrailProfiler
2 t = TrailProfiler("frame-g-006371-5-0089.fits", (1186, 348), (8, 106))
3 t.plot_all_perp_lines()
4 t.plot_profile_by_index(1)
5 t.plot_perpendicular_profiles()
6 t.plot_median_profile()

```

Listing 14: Python session showing usage of the `TrailProfiler` class for various plots

Additional analyses are facilitated by `calculate_auc` and `calculate_auc_with_global_max`, each of which measures the area under a brightness curve using slightly different normalization schemes, and `calculate_fwhm` retrieves the full width at half maximum for a selected profile. Utility methods like `extend_line` ensure geometry remains valid within the image boundaries, while `remove_profile` handles the deletion of unwanted brightness samples. Finally, `plot_main_line` highlights the overarching linear feature on the FITS image.

The usage of the `TrailProfiler` class is demonstrated in Listing 14, where various profiling methods analyze a FITS image containing a meteor trail. The image in question, shown in Figure 20, is loaded into the `TrailProfiler` with specified trail start and end coordinates. The function `plot_all_perp_lines()` is then invoked, producing Figure 21, visualizing all perpendicular lines sampled along the meteor trail. The function `plot_profile_by_index(1)` is then used to extract and display the brightness profile along a specific perpendicular slice, corresponding to Figure 22. The method `plot_perpendicular_profiles()` is then executed, generating Figure 23, which overlays all sampled brightness profiles from the perpendicular slices on a single plot. Lastly, the method `plot_median_profile()` computes and visualizes the median brightness profile across these sampled profiles, as depicted in Figure 24.

The complete code of the `TrailProfiler` class can be found in the appendices.



Figure 20: frame-g-006371-5-0089.fits [41]

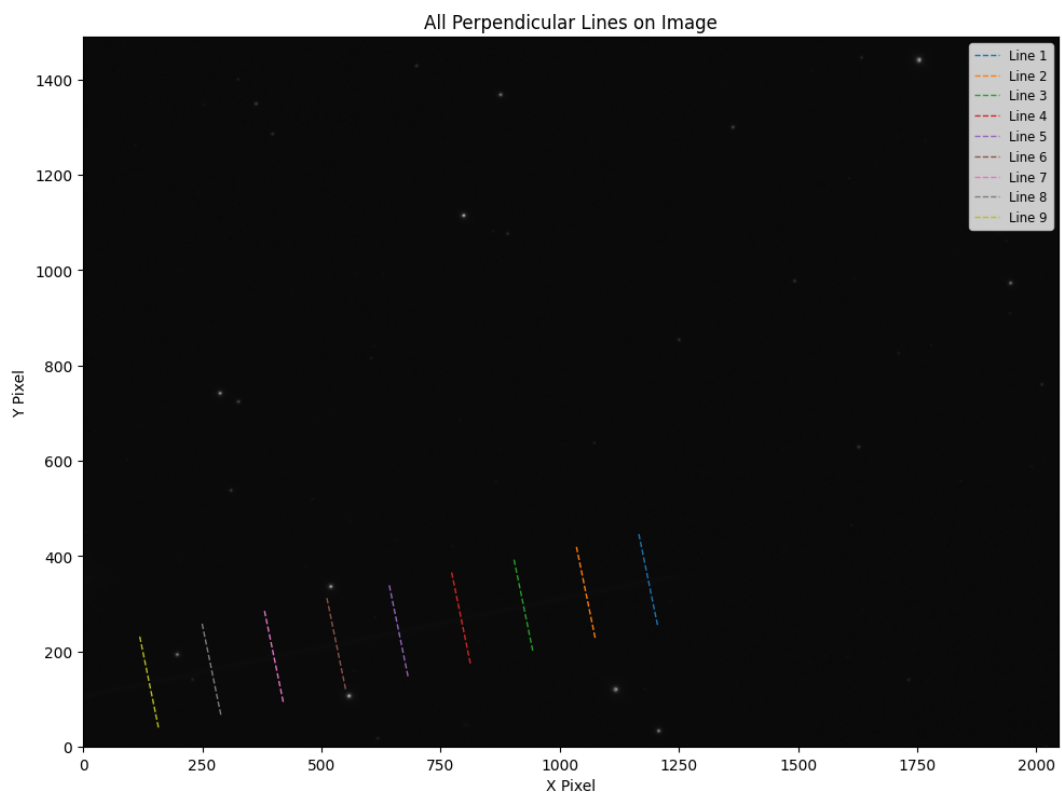


Figure 21: All perpendicular lines visualized for meteor on image frame-g-006371-5-0089.fits

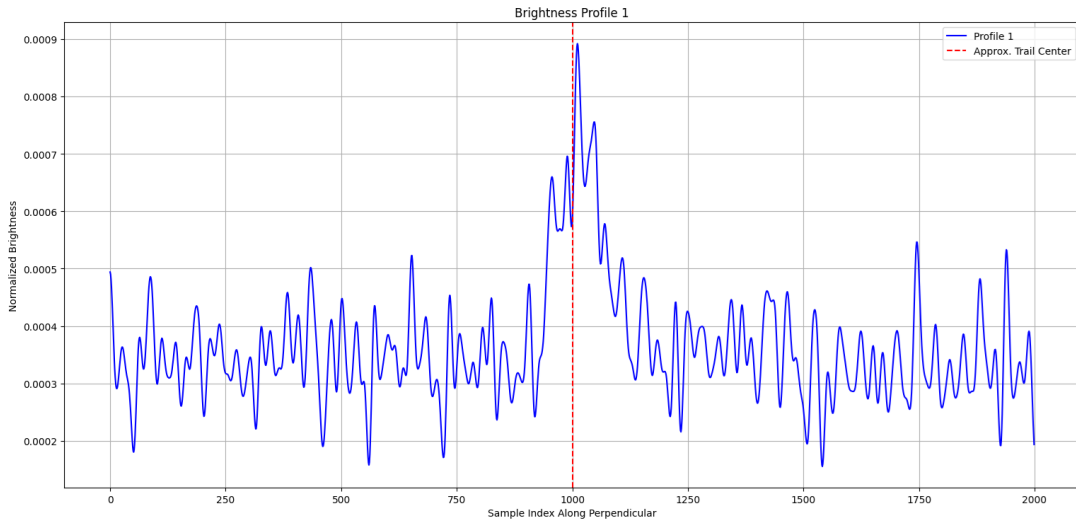


Figure 22: Brightness profile taken across the perpendicular line at index 1 of meteor in image frame-g-006371-5-0089.fits

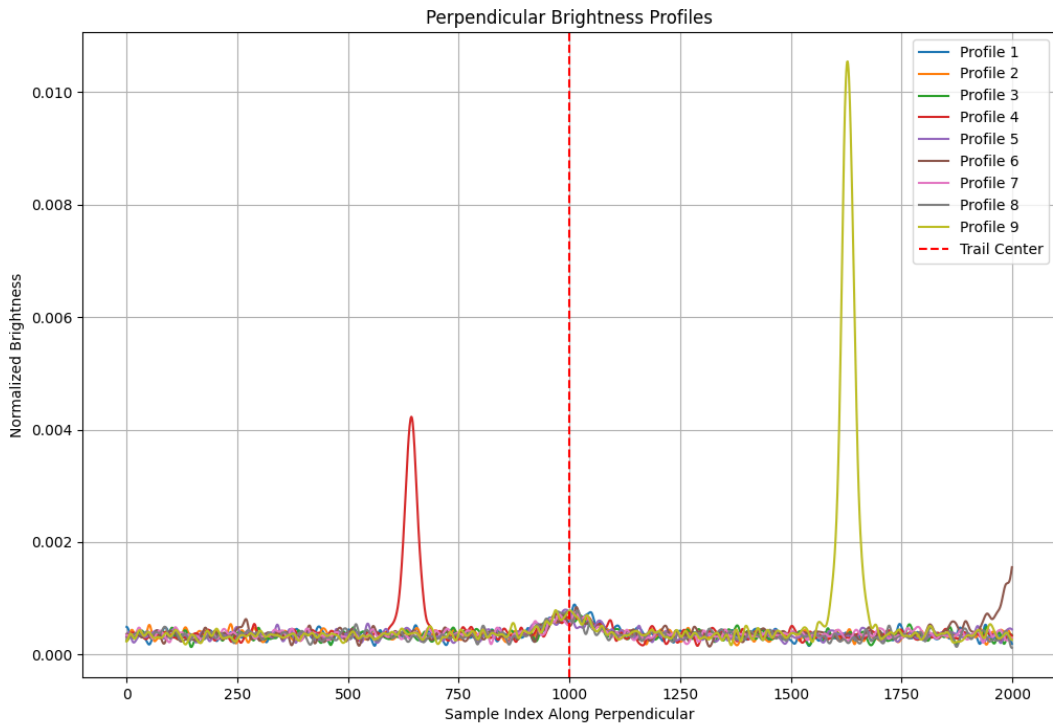


Figure 23: All brightness profiles taken across perpendicular lines of meteor in image frame-g-006371-5-0089.fits

3.3.3. Classification of Linear Artifacts

Artificial neural networks [46] were inspired by how the human brain excels at tasks like recognizing faces in low-quality images or understanding speech in noisy environments. These problems still challenge conventional digital computing. The brain's resilience and capacity for learning without "software updates" contrast to standard computers, which typically fail if the central processor is damaged. Neural networks usually output either a zero or a one each for

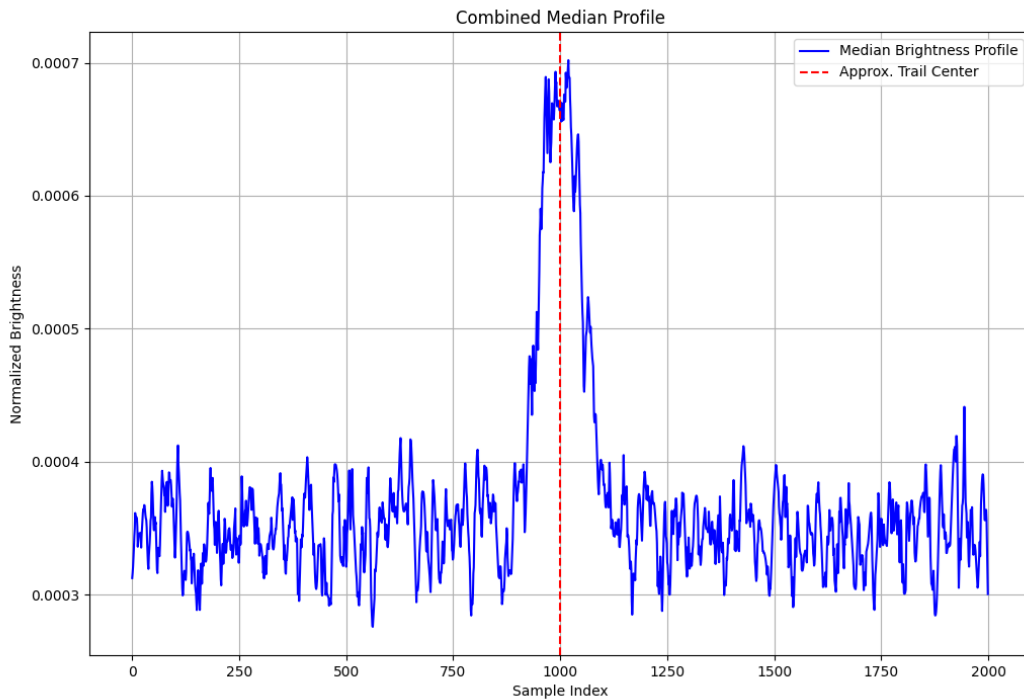


Figure 24: Example of median profile created using the *TrailProfiler* class

the output of one of two classes. Still, they also extend naturally to multi-class prediction by allocating one output unit per target category. In a clinical setting, for instance, where three treatments might exist for a given tumor, each output unit can correspond to whether that tumor responds positively to a specific treatment. According to Krogh [46], neural networks have succeeded in speech recognition, prediction of protein secondary structure, classification of cancers, and gene prediction. However, some researchers might still regard them less favorably since they risk misuse when more straightforward techniques suffice.

In the same way that neural networks can differentiate subtle signals in speech recognition or detect complex patterns in data, they can also be harnessed to analyze trails on astronomical images. For that reason, a neural network was developed for linear artifact classification. By converting linear artifacts in astronomical images into numerical brightness profiles, the network can classify trails according to fine distinctions that simpler methods would fail to recognize. In the particular neural networks developed for this work, an output of 1 would mean that the network predicts the examined trail as a meteor, and an output of 0 would mean that it predicts the trail to be of non-meteor origin.

Two sets of data were collected and were later used for the training and testing purposes of the neural network. They were assembled by manually inspecting FITS images containing linear features and documenting lines that appeared to be genuine meteor streaks versus those that could be attributed to other causes. Each record consists of a file name and coordinates marking the start and end of the observed line. Because these annotations required human judgment and domain expertise, the resulting data set encapsulates many irregularities, such as occluded trails, and is thus valuable data for training the neural network. 100 trails were

collected for the *meteor* data set and 100 trails were collected for the *not_meteor* data set. Both data sets contain 20 trails for each filter (u, g, r, i, z).

Listing 15 shows part of the script that creates a test version of the trail classifying neural network, which is run by providing a folder containing FITS files, and the program then reads two text files that represent the two sets of trail data, *meteors.txt* and *not_meteors.txt*. One line in these files is equal to data of one trail in the format "file.fits $x_1 y_1 x_2 y_2$ ", where the four values are coordinates of the two points defining the trail. After confirming each filter has the expected 40 lines of data total (20 meteors and 20 non-meteors per filter), the script randomly chooses 10 lines per filter for training and 10 for testing, creating sets of line coordinates which are then handed over to the *TrailProfiler* to convert into brightness profiles. The lines are skipped if invalid or fed into a multilayer perceptron (MLP) classifier, which is initialized on lines 30 through 36 in the Listing. This configuration sets up an MLP with two hidden layers, containing 64 neurons in the first layer and 32 in the second. The activation function for each neuron is the Rectified Linear Unit (ReLU), a choice generally perceived as efficient. Training employs the "adam" solver, a gradient-based optimization algorithm, often leading to faster convergence in practice (convergence here refers to the point at which further training achieves no significant improvements). The *max_iter* parameter is set to 200; this means the training routine stops if it hasn't converged by 200 iterations. The classifier's performance is evaluated on the test data, with the results displayed via a classification report that distinguishes between "Meteor" and "Not Meteor" predictions.

Because this version of the neural network was trained on half of the collected data and tested on the other half, it was only used to test the concept of using a neural network for trail classification. The results of this neural network are shown in Table 2. For the "Not Meteor" class, the model achieved a high precision of 0.91, meaning that most predictions labeled as not meteor were correct; however, the recall was 0.78, indicating that about 22% of actual non-meteor cases were missed. In contrast, the "Meteor" class exhibited a high recall of 0.92, demonstrating that the model correctly identified nearly all meteor trails but with a slightly lower precision of 0.81, suggesting some false positives in this category. Overall, the classification scores are promising. An accuracy of 85% indicates that the model correctly classifies most instances.

	Precision	Recall	F1-Score	Support
Not Meteor	0.91	0.78	0.84	50
Meteor	0.81	0.92	0.86	50
Accuracy			0.85	100
Macro avg	0.86	0.85	0.85	100
Weighted avg	0.86	0.85	0.85	100

Table 2: Classification report of the test neural network

Along with a neural network for classifying between meteor and non-meteor trails, a set of metrics was devised to measure their properties. The metrics used for the characterization of trails are defined as follows. The *Full Width at Half Maximum* (FWHM) of a brightness profile

```

1 main():
2     args = parse_args()

3     meteor_file = "meteors.txt"
4     not_meteor_file = "not_meteors.txt"

5     meteor_lines = load_lines_from_file(meteor_file, label=1,
6     ↪ fits_folder=args.fits_folder)
7     not_meteor_lines = load_lines_from_file(not_meteor_file, label=0,
8     ↪ fits_folder=args.fits_folder)

9     data_by_filter = defaultdict(list)
10    for entry in (meteor_lines + not_meteor_lines):
11        fits_file, x1, y1, x2, y2, label = entry
12        flt = extract_filter(fits_file)
13        data_by_filter[flt].append(entry)

14    train_data = []
15    test_data = []
16    for flt in ['u', 'g', 'r', 'i', 'z']:
17        lines_for_filter = data_by_filter[flt]
18        meteor_subset = [x for x in lines_for_filter if x[5] == 1]
19        not_meteor_subset = [x for x in lines_for_filter if x[5] == 0]

20        random.shuffle(meteor_subset)
21        random.shuffle(not_meteor_subset)

22        train_meteor = meteor_subset[:10]
23        test_meteor = meteor_subset[10:]
24        train_not_meteor = not_meteor_subset[:10]
25        test_not_meteor = not_meteor_subset[10:]

26        train_data.extend(train_meteor + train_not_meteor)
27        test_data.extend(test_meteor + test_not_meteor)

28    X_train, y_train = build_dataset(train_data)
29    X_test, y_test = build_dataset(test_data)

30    y_train = np.array(y_train)
31    y_test = np.array(y_test)

32    clf = MLPClassifier(
33        hidden_layer_sizes=(64, 32),
34        activation='relu',
35        solver='adam',
36        max_iter=200,
37        random_state=42
38    )

39    clf.fit(X_train, y_train)

40    y_pred = clf.predict(X_test)

```

Listing 15: The main() function of the *test_meteor_classifier.py* file

$f(x)$ is defined as the difference between the two positions, x_1 and x_2 , where

$$f(x_1) = f(x_2) = \frac{M}{2},$$

with

$$M = \max_x f(x).$$

A narrow FWHM implies a narrow trail, whereas a wide FWHM indicates a wide trail. In a similar vein, the width measured at 95% of the maximum, denoted as $\text{FWHM}_{0.95}$, is defined by the condition

$$f(x_1) = f(x_2) = 0.95 M,$$

so that

$$\text{FWHM}_{0.95} = x_2 - x_1,$$

which captures the extent of the brightest portion of the profile. The width at 70% of the maximum, $\text{FWHM}_{0.7}$, is defined where

$$f(x_1) = f(x_2) = 0.7 M.$$

The ratio

$$R = \frac{\text{FWHM}_{0.7}}{\text{FWHM}_{0.95}}$$

shows how the profile's width scales between these thresholds; a ratio close to 1 suggests a consistently narrow profile and a lower ratio indicates a profile that tapers off more steeply. The area under the curve (AUC) metrics are computed using different normalizations. Specifically,

$$\text{AUC}_{\text{med}} = \int \frac{f(x)}{m} dx,$$

where m is the median brightness of the entire FITS image, providing a measure of overall light coverage relative to the typical background. Similarly,

$$\text{AUC}_{\text{peak}} = \int \frac{f(x)}{M} dx,$$

normalizes by the peak value M of the median profile, and

$$\text{AUC}_{\text{full}} = \int \frac{f(x)}{M_{\text{FITS}}} dx,$$

normalizes by the maximum pixel value M_{FITS} of the FITS image. Finally, the kurtosis of the brightness profile is defined as

$$\text{kurtosis} = \frac{\sum_n (x_n - \mu)^4 f(x_n)}{\sigma^4 \sum_n f(x_n)} - 3,$$

where μ and σ denote the mean and standard deviation of the profile, respectively, with $f(x_n)$ serving as weights. A high kurtosis value indicates a sharply peaked brightness profile and a lower kurtosis suggests a more broadly distributed brightness profile. Along with the kurtosis

```

1 median_profile = t.calculate_median_profile()

2 t.calculate_auc_full(median_profile) # Output: 0.0032025473
3 t.calculate_auc_med(median_profile)  # Output: 273.3834228515625
4 t.calculate_auc_peak(median_profile)  # Output: 1045.89208984375

5 t.calculate_fwhm_default(median_profile) # Output: 1989.0
6 t.calculate_fwhm_07(median_profile)     # Output: 128.0
7 t.calculate_fwhm_095(median_profile)    # Output: 59.0

8 t.calculate_kurtosis(median_profile)     # Output: -1.0923731016909046
9 t.calculate_gaussian_kurtosis(median_profile) # Output: -1.1387414180921318

```

Listing 16: Computation of trail profile metrics for frame-g-006371-5-0089.fits

of the median trail, a kurtosis of the median trail fitted to a Gaussian curve is used alongside it, which is referred to in this work as Gaussian kurtosis. Gaussian kurtosis evaluates how much the brightness profile deviates from a normal distribution.

The median brightness profile extracted from the meteor trail in *frame-g-006371-5-0089.fits* (Figure 24) was characterized in Listing 16. The Full Width at Half Maximum (FWHM) was found to be $\text{FWHM} = 1989.0$. A narrower width at 70% of the maximum intensity, $\text{FWHM}_{0.7} = 128.0$, and an even narrower width at 95% of the maximum intensity, $\text{FWHM}_{0.95} = 59.0$, indicate the profile's shape. The compactness ratio defined as

$$R = \frac{\text{FWHM}_{0.7}}{\text{FWHM}_{0.95}} = \frac{128.0}{59.0} \approx 2.17,$$

suggests that the brightness distribution tapers off fairly quickly but is still in the expected range of what a meteor would show.

The AUC, when normalized by the global maximum of the FITS image, results in $\text{AUC}_{\text{full}} = 0.0032$, while normalizing by the median image brightness gives $\text{AUC}_{\text{med}} = 273.38$. The integral normalized by the peak brightness of the median profile yields $\text{AUC}_{\text{peak}} = 1045.89$, providing a measure of the total intensity concerning the brightest region. The kurtosis of the profile was computed as $\text{kurtosis} = -1.09$, and a Gaussian kurtosis correction results in $\text{GaussianKurtosis} = -1.14$. Since kurtosis measures the sharpness of a peak, these negative values indicate a relatively broad distribution.

3.4. Findings and Interpretations

This final section describes insights gained from the linear artifact analysis. The first subsection presents the results of the neural network classification and the differences between trails classified as meteors and non-meteors through the characterization metrics described in the previous section. It then highlights the graphical user interface (GUI) created for ease of use regarding trail profiling when examining meteor trails case-by-case. In the last subsection, potential pitfalls encountered during development are discussed, along with recommendations to enhance the pipeline in future development.

3.4.1. Evaluation Metrics for Classification of Linear Artifacts

The dataset initially contained 26,600 images, of which 22,390 had at least one detected linear feature; a portion of the original resulting dataset of the LFDA is not being taken into account because of the difference in the detection process, most likely several either too faint or too short trails that the *TrailDetector* class did not count. While this means that the final dataset is slightly smaller, it also means that only prominent trails are considered. Among these detected lines, 21,327 could be successfully fitted to a Gaussian function. The remaining cases either lacked a coherent shape or contained irregularities that prevented Gaussian fitting, making them less suitable for profiling.

Of these, 11,262 were classified as very likely meteors, having received a model prediction score greater than 0.8, while 8,077 were classified as very likely non-meteors, with scores below 0.2. A total of 12,373 trails were classified as meteors (with scores greater than 0.5), and 8,954 were classified as non-meteors (with scores less than or equal to 0.5). This distribution indicates a strong separation in classification, with a considerable portion of detected trails being confidently labeled as meteors.

With the *TrailDetector* class being used for detecting trails before classification, some of the manually defined test samples were not detected. Among the 200 test samples, 12 meteors and 18 non-meteors were missed, resulting in 170 detected test cases. Within these, the classification model correctly identified 78% (78/100) of meteors and 54% (54/100) of non-meteors. When considering only the detected trails, the model achieved an accuracy of 77.65%, indicating that the classifier performs well when provided with a detected trail. The primary source of error in overall classification performance stems from the *TrailDetector*'s ability to capture trails rather than the classifier itself.

The overall statistics of the metrics shown in Tables 3 and 4 provide insights into the properties of the detected trails in the dataset. The FWHM measurements show a broad distribution, with a large gap between the average and median values, suggesting the presence of both very wide and very narrow trails. The compactness metric, which represents the ratio between $FWHM_{0.7}$ and $FWHM_{0.95}$, also shows a substantial range. The AUC values, which measure the total brightness distribution of the trails under different normalizations, exhibit high variation, especially in the case of AUC_{med} . The kurtosis values suggest that most profiles are relatively flat, with an overall negative mean.

Table 3: Overall Metrics - FWHM and Compactness

Metric	$FWHM_{default}$	$FWHM_{0.7}$	$FWHM_{0.95}$	Compactness
Average	984.16	743.00	264.49	37.20
Median	192.00	44.00	10.00	3.92
Std Dev	962.69	919.67	553.90	107.96
Range	1991.00	1996.00	1999.00	1940.00

The subset of trails classified as very likely meteors (having a prediction score greater than 0.8) consists of 11,262 samples; their statistics are shown in Tables 5 and 6. The FWHM metrics for these trails show significantly high average values, with $FWHM_{default}$ reach-

Table 4: Overall Metrics - AUC and Kurtosis

Metric	AUC _{med}	AUC _{peak}	AUC _{full}	Kurtosis	Gaussian Kurtosis
Average	1467.33	864.43	0.05	-0.78	-0.98
Median	533.04	822.13	0.01	-1.12	-1.15
Std Dev	46667.57	581.26	1.17	1.27	0.38
Range	5391396.00	1963.34	140.73	58.20	1.24

ing 1677.23 pixels on average and FWHM_{0.7} reaching 1314.77 pixels, both suggesting that trails in this category tend to be long and prominent. However, the median values are much lower, particularly for FWHM_{0.95}, indicating that while some trails are extensively wide, a more significant portion are narrow. The compactness metric averages 61.80, suggesting that most of these trails exhibit gradual brightness falloff.

Regarding AUC metrics, the AUC_{med} and AUC_{peak} values show that these trails contain high relative brightness, averaging 1553.87 and 1273.39, respectively. AUC_{full} remains extremely low since these trails do not typically reach the maximum pixel intensity of the FITS images. Kurtosis values emphasize the shape differences, with the average kurtosis of -0.93 and Gaussian kurtosis of -1.11, indicating that these trails tend to be more spread out rather than peaked. A high standard deviation in all metrics, particularly in FWHM and AUC, suggests various meteor trails within this category.

Table 5: Metrics for Very Likely Meteors (score > 0.8) - FWHM and Compactness

Metric	FWHM _{default}	FWHM _{0.7}	FWHM _{0.95}	Compactness
Average	1677.23	1314.77	475.24	61.80
Median	1999.00	1988.00	34.00	4.59
Std Dev	697.08	889.40	675.93	140.41
Range	1987.00	1993.00	1999.00	1940.00

Table 6: Metrics for Very Likely Meteors (score > 0.8) - AUC and Kurtosis

Metric	AUC _{med}	AUC _{peak}	AUC _{full}	Kurtosis	Gaussian Kurtosis
Average	1553.87	1273.39	0.02	-0.93	-1.11
Median	470.57	1414.45	0.01	-1.18	-1.19
Std Dev	47568.27	429.36	0.21	1.29	0.26
Range	3753967.50	1898.30	14.51	30.05	1.20

The subset of trails classified as very likely non-meteors (having a prediction score lower than 0.2) consists of 8,077 samples, and their data is shown on Tables 7 and 8. The FWHM metrics for this group are significantly lower than those in the meteor group, with FWHM_{default} averaging only 112.66 pixels and FWHM_{0.7} at 76.18 pixels, indicating that these trails are much shorter. The FWHM_{0.95} is only 23.22 pixels on average. The compactness metric, at 7.50 on average, is also significantly lower than the values found for meteors.

The AUC metrics also suggest notable differences in brightness distribution from meteors. The AUC_{med} averages 1123.99, but the AUC_{peak} is much lower at 322.04, implying that these trails tend to have less overall brightness concentration. The AUC_{full}, at 0.09, is slightly higher than that of very likely meteors, suggesting that some of these trails may be noise. Fi-

nally, the kurtosis values are higher than those of meteors, with an average of -0.55 for kurtosis and -0.77 for Gaussian kurtosis, meaning that these profiles tend to be more sharply peaked than their meteor counterparts.

Table 7: Metrics for Very Likely Non-Meteors (score < 0.2) - FWHM and Compactness

Metric	FWHM _{default}	FWHM _{0.7}	FWHM _{0.95}	Compactness
Average	112.66	76.18	23.22	7.50
Median	38.00	26.00	7.00	3.67
Std Dev	363.12	299.90	162.79	21.64
Range	1991.00	1996.00	1999.00	666.33

Table 8: Metrics for Very Likely Non-Meteors (score < 0.2) - AUC and Kurtosis

Metric	AUC _{med}	AUC _{peak}	AUC _{full}	Kurtosis	Gaussian Kurtosis
Average	1123.99	322.04	0.09	-0.55	-0.77
Median	633.88	254.54	0.01	-0.89	-1.00
Std Dev	45725.25	279.20	1.89	1.23	0.45
Range	4135586.00	1963.34	140.73	58.20	1.24

The more significant subset of trails classified as meteors (having a prediction score greater than 0.5) consists of 12,373 samples. As seen in Table 9, the FWHM metrics indicate that these trails are significantly wider than non-meteors, with an average FWHM_{default} of 1604.06 pixels and FWHM_{0.7} at 1223.33 pixels. These values suggest that meteor trails are generally long and continuous. The FWHM_{0.95} for this subset stands at 438.24 pixels on average.

The AUC metrics of this subset can be seen in Table 10. The AUC_{med} averages 1624.14, while the AUC_{peak} averages 1234.32, suggesting that meteor trails typically maintain a high brightness level along their profile in this more significant subset. The AUC_{full}, though small at 0.02, remains consistent with the fact that meteors tend to occupy substantial portions of their images rather than appearing as isolated bright spots. The average kurtosis of -0.93 and Gaussian kurtosis of -1.11 indicate that meteor profiles tend to be flatter and more spread out.

Table 9: Metrics for Meteors (score > 0.5) - FWHM and Compactness

Metric	FWHM _{default}	FWHM _{0.7}	FWHM _{0.95}	Compactness
Average	1604.06	1223.33	438.24	58.54
Median	1999.00	1971.00	26.00	4.55
Std Dev	755.70	916.49	660.86	136.43
Range	1987.00	1993.00	1999.00	1940.00

The larger subset of trails classified as non-meteors (having a prediction score less than or equal to 0.5) consists of 8,954 samples. As shown in Table 11, the FWHM metrics indicate that these trails are narrower than those classified as meteors. The FWHM_{default} has an average of 127.55 pixels, with FWHM_{0.7} averaging 79.27 pixels, and FWHM_{0.95} at 24.40 pixels. This suggests that non-meteor trails tend to be shorter, thinner, or more fragmented.

The AUC metrics for this subset are shown in Table 12. The AUC_{med} averages 1250.64, while the AUC_{peak} is 353.29. The AUC_{full}, at 0.09 on average, is also higher than that of meteors,

Table 10: Metrics for Meteors (score > 0.5) - AUC and Kurtosis

Metric	AUC _{med}	AUC _{peak}	AUC _{full}	Kurtosis	Gaussian Kurtosis
Average	1624.14	1234.32	0.02	-0.93	-1.11
Median	470.13	1358.63	0.01	-1.17	-1.18
Std Dev	47120.54	439.84	0.20	1.27	0.26
Range	3753967.50	1898.30	14.51	30.05	1.20

indicating that these trails are much brighter when compared to the maximum pixel value of the FITS image. The average kurtosis of -0.57 and Gaussian kurtosis of -0.80 suggest that these trails tend to have more peaked, distinguishing them from the smoother profiles of meteor trails. The relatively small standard deviation and range across the FWHM values reinforce the idea that non-meteors are generally more consistent in their profile shapes.

Table 11: Metrics for Non-Meteors (score ≤ 0.5) - FWHM and Compactness

Metric	FWHM _{default}	FWHM _{0.7}	FWHM _{0.95}	Compactness
Average	127.55	79.27	24.40	7.71
Median	39.00	27.00	7.00	3.67
Std Dev	391.77	307.42	167.06	23.27
Range	1991.00	1996.00	1999.00	666.33

Table 12: Metrics for Non-Meteors (score ≤ 0.5) - AUC and Kurtosis

Metric	AUC _{med}	AUC _{peak}	AUC _{full}	Kurtosis	Gaussian Kurtosis
Average	1250.64	353.29	0.09	-0.57	-0.80
Median	616.46	277.23	0.01	-0.92	-1.02
Std Dev	46033.43	295.09	1.79	1.24	0.44
Range	4135586.00	1963.34	140.73	58.20	1.24

To show a direct comparison of average FWHM values across different prediction categories, an overview of FWHM_{default}, FWHM_{0.7}, and FWHM_{0.95} for each subset can be seen in Table 13. This table shows the difference between the two groups: Meteors consistently have much higher FWHM values, indicating more expansive trails, whereas non-meteors tend to be significantly narrower. Table 14 compares the median values of FWHM_{default}, FWHM_{0.7}, FWHM_{0.95}, and Compactness for each category. Outliers less influence the median values, but much as the averages, they point to the same conclusion: meteors tend to be wider, and non-meteors tend to be slimmer.

Table 15 presents the average values of AUC_{med}, AUC_{peak}, AUC_{full}, Kurtosis, and Gaussian Kurtosis for each category. The results show that meteors tend to have higher AUC values and lower kurtosis, indicating that they are more extended and less sharply peaked than non-meteor trails. Table 16 presents the median values of AUC_{med}, AUC_{peak}, AUC_{full}, Kurtosis, and Gaussian Kurtosis for each classification category. The results demonstrate that meteors consistently exhibit higher AUC values and lower kurtosis, again pointing to the trend seen in comparing the averages of these values.

Trails classified as meteors generally exhibit significantly larger FWHM values across all definitions (FWHM_{default}, FWHM_{0.7}, FWHM_{0.95}) compared to non-meteors, which suggests that

Table 13: Comparison of Averages of FWHM Metrics and Compactness Across Prediction Categories

Category	FWHM _{default}	FWHM _{0.7}	FWHM _{0.95}	Compactness
Very Likely Meteors (score > 0.8)	1677.23	1314.77	475.24	61.80
Very Likely Non-Meteors (score < 0.2)	112.66	76.18	23.22	7.50
Meteors (score > 0.5)	1604.06	1223.33	438.24	58.54
Non-Meteors (score <= 0.5)	127.55	79.27	24.40	7.71

Table 14: Comparison of Medians of FWHM Metrics and Compactness Across Prediction Categories

Category	FWHM _{default}	FWHM _{0.7}	FWHM _{0.95}	Compactness
Very Likely Meteors (score > 0.8)	1999.00	1988.00	34.00	4.59
Very Likely Non-Meteors (score < 0.2)	38.00	26.00	7.00	3.67
Meteors (score > 0.5)	1999.00	1971.00	26.00	4.55
Non-Meteors (score <= 0.5)	39.00	27.00	7.00	3.67

meteors tend to produce more expansive trails. In contrast, non-meteors are typically narrower and more confined. The compactness metric, which represents the ratio $FWHM_{0.7} / FWHM_{0.95}$, is higher for meteors, meaning their brightness profiles extend gradually. Meteors consistently have higher AUC values (AUC_{med} , AUC_{peak}) than non-meteors. The higher AUC_{peak} values for meteors suggest that they maintain a consistent close-to-peak value across their trail. In contrast, non-meteors like satellites might more frequently have an intense spike in the center of the profile, but after that, the profile dips and stays at the background level. The AUC_{full} metric, which considers the global image maximum, is considerably lower in both categories. Still, the average of this metric is almost five times larger for non-meteors than it is for meteors, so while meteors dominate in the first two AUC metrics, non-meteors like satellites beat them in this regard. This is not a surprise since satellites tend to appear as more intense streaks even though they are condensed and less wide.

Table 15: Comparison of Averages of AUC and Kurtosis Across Prediction Categories

Category	AUC _{med}	AUC _{peak}	AUC _{full}	Kurtosis	Gaussian Kurtosis
Very Likely Meteors (score > 0.8)	1553.87	1273.39	0.02	-0.93	-1.11
Very Likely Non-Meteors (score < 0.2)	1123.99	322.04	0.09	-0.55	-0.77
Meteors (score > 0.5)	1624.14	1234.32	0.02	-0.93	-1.11
Non-Meteors (score <= 0.5)	1250.64	353.29	0.09	-0.57	-0.80

Table 16: Comparison of Medians of AUC and Kurtosis Across Prediction Categories

Category	AUC _{med}	AUC _{peak}	AUC _{full}	Kurtosis	Gaussian Kurtosis
Very Likely Meteors (score > 0.8)	470.57	1414.45	0.01	-1.18	-1.19
Very Likely Non-Meteors (score < 0.2)	633.88	254.54	0.01	-0.89	-1.00
Meteors (score > 0.5)	470.13	1358.63	0.01	-1.17	-1.18
Non-Meteors (score <= 0.5)	616.46	277.23	0.01	-0.92	-1.02

3.4.2. Graphical User Interface for Trail Examination

The graphical user interface (GUI) developed in this work is built using Python's Tkinter framework, and it uses several libraries to provide a tool for trail profiling and classification when done on a case-by-case basis. It uses *astropy* for FITS file handling and image normalization, *OpenCV* and *NumPy* for image processing, and *Matplotlib* for visualization. The GUI enables users to load and display FITS images, automatically detect linear artifacts with the *TrailDetector* class, and perform trail profiling with the *TrailProfiler* class. The interface supports manual line selection, reanalysis of perpendicular brightness profiles, and visualization of individual and combined brightness profiles. It also allows users to load a neural network model, such as the one evaluated in the previous chapter, and obtain predictions on the median brightness profile of the given FITS image.

The application's main menu shown in Figure 25 is organized into sections according to phases of the workflow one would follow when doing trail analysis. The sections are shown on the left side as follows: the File Controls section allows users to load FITS images; the Detection Controls section provides tools for automatic and manual selection of trail lines; the Profiling and Additional Controls sections offer options for managing the brightness profiles derived from these trails; and finally, the Neural Network Model Controls enable the user to load a model and obtain predictions based on the median brightness profile.

If the user decides to detect linear artifacts in the loaded FITS image automatically, they can select the "Detect Lines Automatically" button in the main menu. The application displays the detected lines in a dedicated panel, each with a label identifying them according to line coordinates. This detection relies on the *TrailDetector* class. The user is then prompted to review each detected line, with options to view a plot with the highlighted artifact and to toggle whether the line should be extended to the image boundaries. Once the user is satisfied with a particular line, they can select "Use this line for profiling," passing the chosen endpoints to the *TrailProfiler* class.

After the user chooses an automatically detected trail or provides their own coordinates,

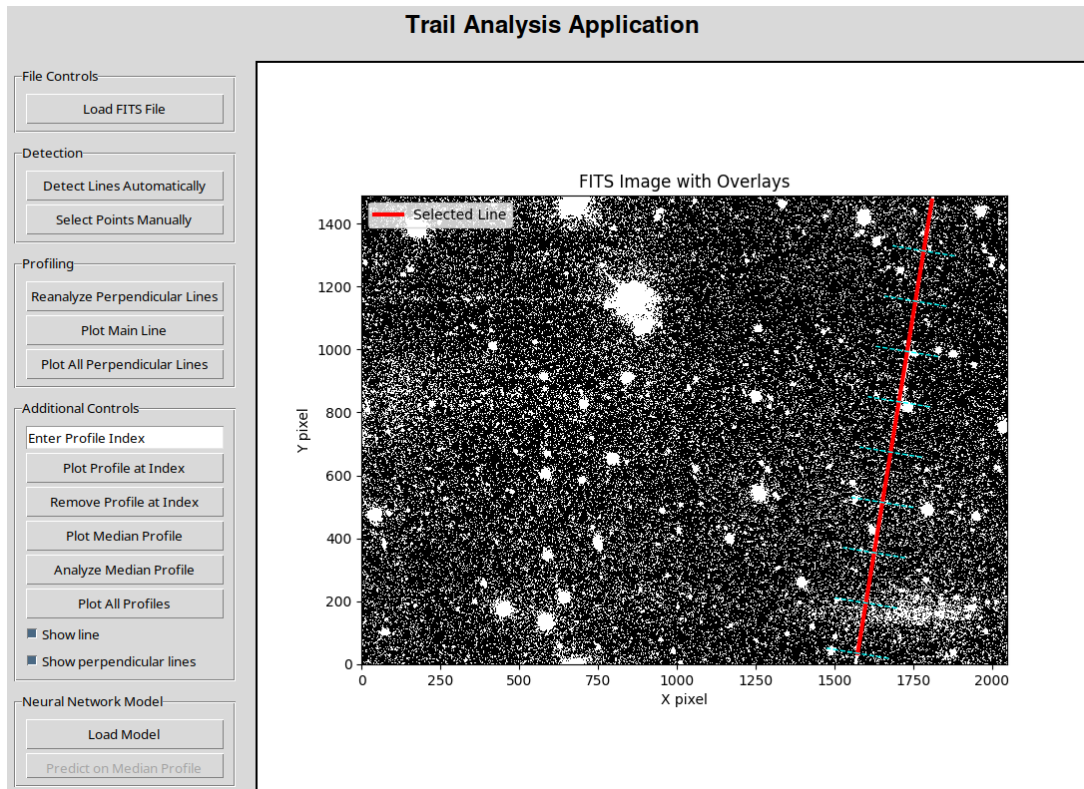


Figure 25: Main menu of the Trail Analysis Application

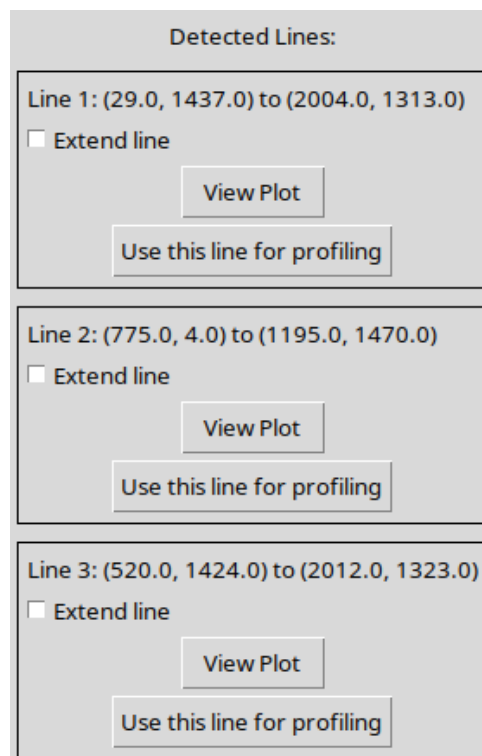


Figure 26: Detected lines menu of the Trail Analysis Application

they can move on to the profiling section of the application with tools to analyze the selected trail's brightness distribution in depth. Once a line from the detection panel is chosen, the

TrailProfiler is invoked to sample brightness values along multiple perpendicular slices across the trail. Selecting the "Plot Main Line" button opens a plot of the main line overlaid on the image, which can be viewed in Figure 27. Figure 28 shows the plot after selecting the "Plot All Perpendicular Lines" button.

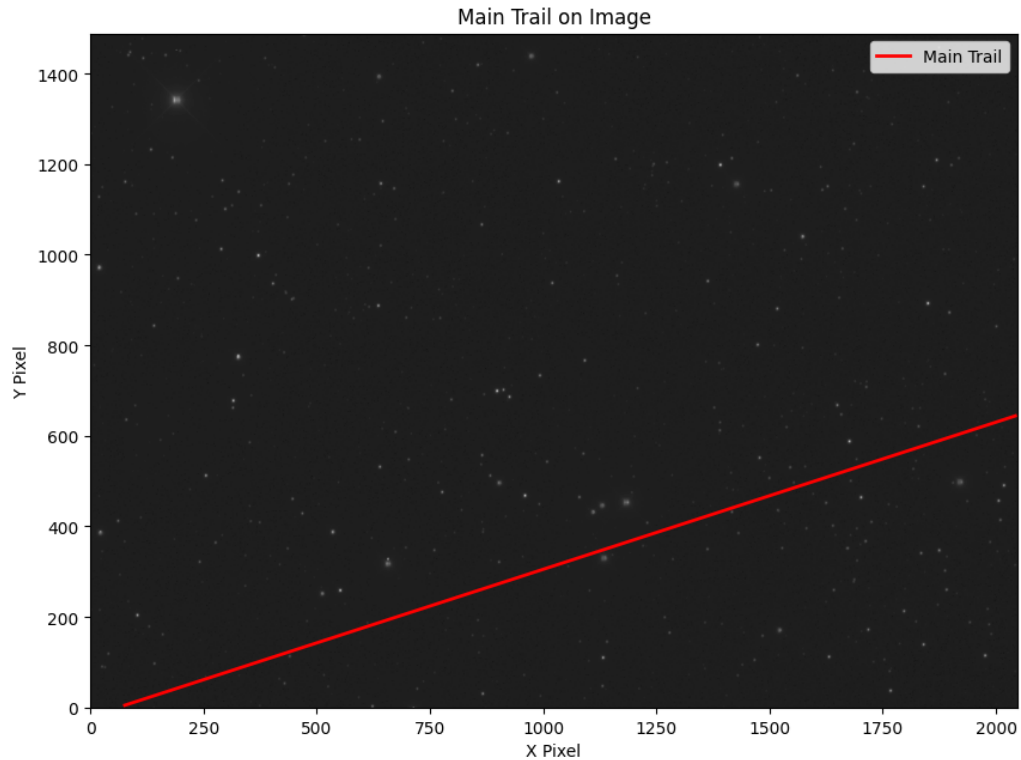


Figure 27: Plot of the main chosen line after selecting the "Plot Main Line" button

The sampled profiles along the perpendicular lines can be visualized individually or together; figure 29 shows the plot of a single profile at a chosen index, and figure 28 shows all of them overlaid on the same plot. The interface also offers functions to compute the characterization metrics shown in Figure 31, which can be viewed by selecting the "Analyze Median Profile" button. Additionally, users can resample the trail with different parameters by clicking the "Reanalyze Perpendicular Lines" button. When the user is done with trail profiling, they may import a neural network model and get a prediction based on the median trail, an example shown in Figure 32.

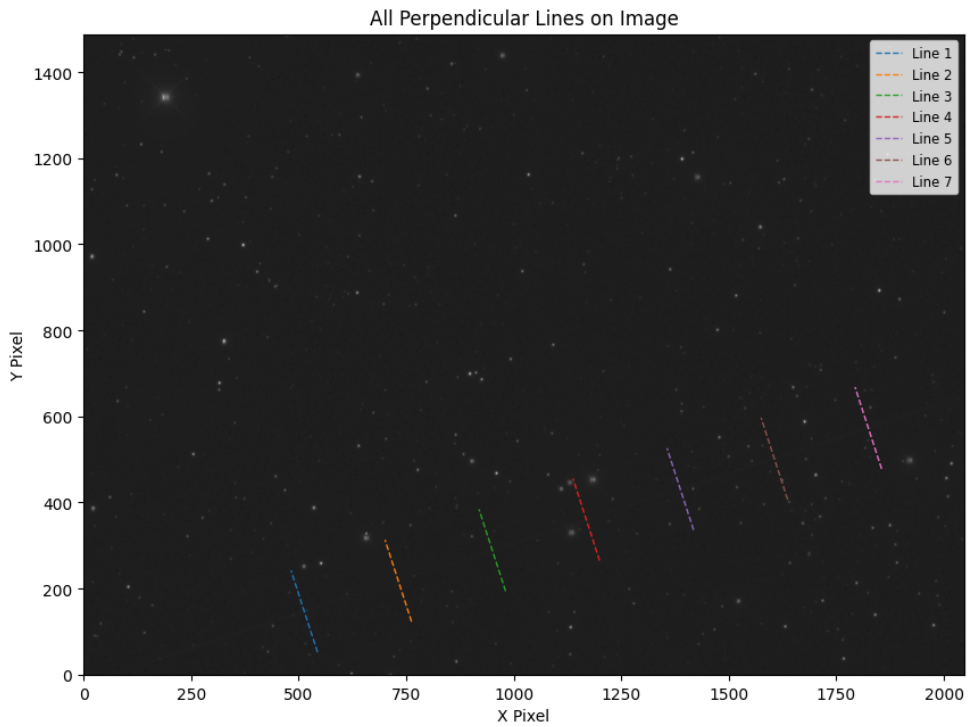


Figure 28: Plot of the perpendicular lines after selecting the "Plot All Perpendicular Lines" button

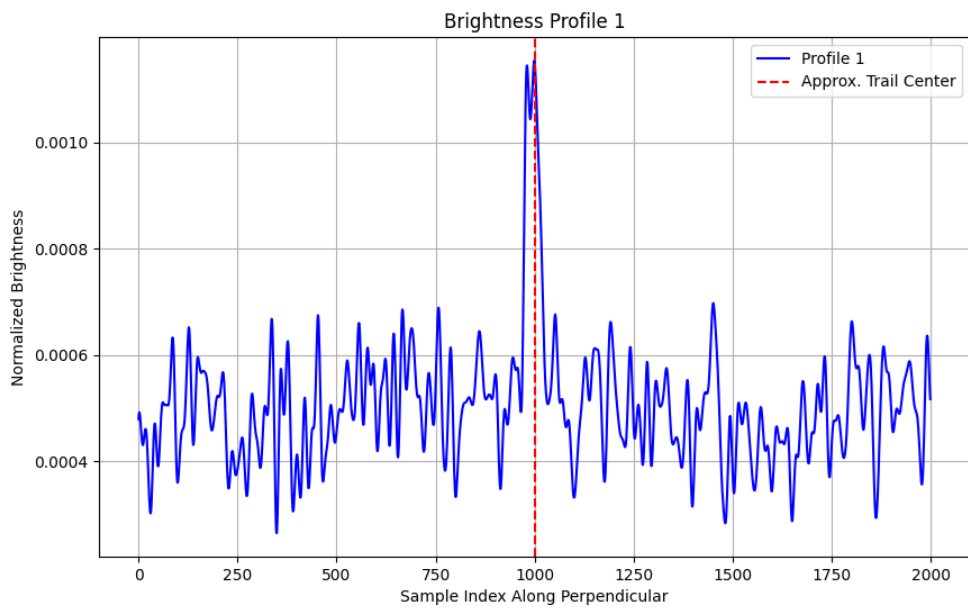


Figure 29: Plot of a single profile after entering an index in the provided field and selecting the "Plot Profile at Index" button

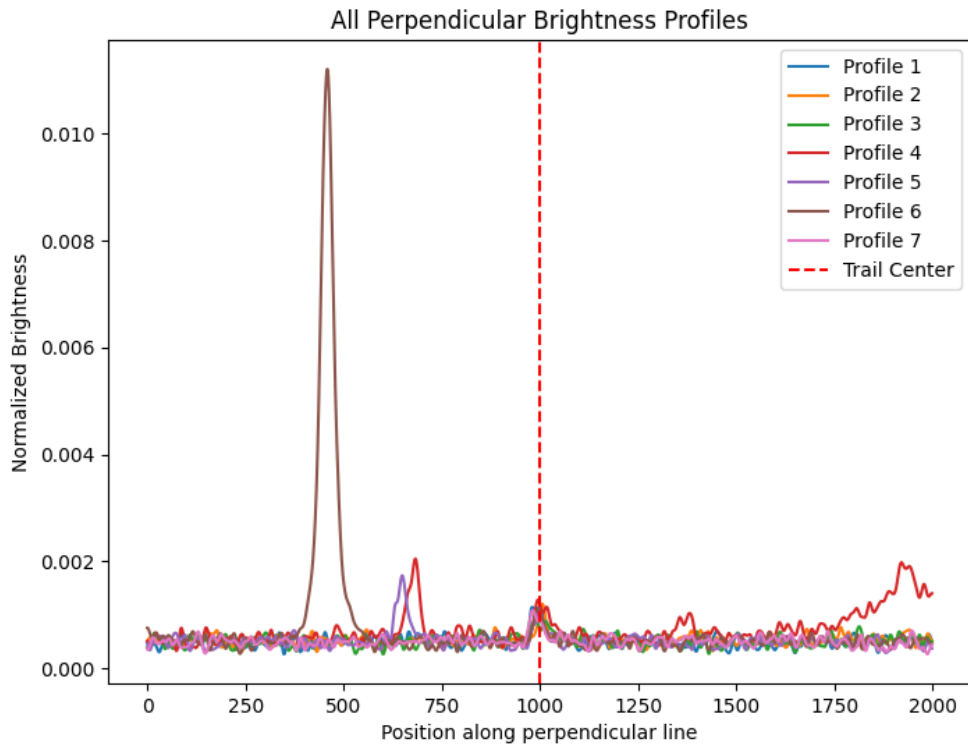




Figure 30: Plot of all of the perpendicular line profiles after selecting the "Plot All Profiles" button


FWHM (0.5 factor): 1980.00
FWHM (0.7 factor): 39.00
FWHM (0.95 factor): 7.00
Compactness
(FWHM0.7/FWHM0.95): 5.57
AUC (median normalized): 429.57
AUC (peak normalized): 984.69
AUC (global max normalized): 0.01
Kurtosis: -1.16
Gaussian Kurtosis: -1.18

OK

Figure 31: View of calculated metrics after selecting the "Analyze Median Profile" button


Predicted probability of meteor:
0.822

OK

Figure 32: Neural network prediction view after importing a model and selecting the "Predict on Median Profile" button

3.4.3. Technical Obstacles and Improvement Possibilities

This work's technical obstacles reveal avenues for future refinement of the linear artifact classifying process. For instance, while effective, the current detection algorithm still fails to identify a number of more ambiguous trails, resulting in valuable data being left out. In the profiling stage, brightness profiles extending beyond the image boundaries are discarded outright rather than being adaptively resampled to capture the entire trail; this limitation may result in an incomplete characterization of trails that move close along the edge of the image. The neural network classifier is presently configured for binary classification—differentiating between meteors and non-meteors—but it can be utilized for nuanced discrimination, distinguishing between many classes, such as wide meteor trails, sharp meteor trails, rotating satellites, and other noise. Finally, the limited dataset used for training suggests that a larger volume of annotated data could substantially improve the model's performance.

Although these technical challenges have been identified, the scope of the current project is focused on establishing a pipeline for meteor trail analysis. Addressing every issue would require more standalone effort involving additional refinement in all areas of the project. Nevertheless, the project succeeds in detecting, profiling, and classifying trails, demonstrating the feasibility of automated meteor classification. At the same time, these challenges serve as promising directions for future work rather than shortcomings in the present study.

4. Conclusion

This work proposes a solution to the problem of meteor classification through a pipeline that detects, profiles, and classifies trails in astronomical images. The pipeline workflow minimizes the effort required when manually inspecting the images to determine between meteor and non-meteor trails.

Incorporating a neural network model for the classification between meteors and non-meteors proved effective; the neural network consistently separated actual meteor events from other lines with high accuracy. The clear separation of trails as meteors and non-meteors indicates that the collected dataset of 200 trails used for the training of the neural network model was sufficient enough to use for the classification process in the given dataset. However, a larger dataset would yield better results since the developed model still has many examples hovering around the 0.5 prediction value.

The findings of the characterization metrics demonstrate that FWHM, AUC, and kurtosis are practical factors that distinguish between the most common meteor and non-meteor trails. The classification model aligns well with initial expectations; the trails classified as meteors produce broader trails, while those classified as non-meteors (e.g., noise or satellite trails) tend to be sharper, more confined, and less intense across the whole profile but more intense in the center.

5. Acknowledgments

I would like to express my sincere gratitude to Dr. Dejan Vinković for his invaluable guidance throughout this work. His direction shaped the approach taken for the trail profiling process, and I am genuinely grateful for his time, patience, and support. He was involved in developing the LFDA, an essential foundation for this research.

I want to extend my sincere gratitude to Dr. Dino Bektešević for his assistance with the LFDA and for giving me insights into its functionality. His willingness to share knowledge helped me a great deal in developing this research, and I am grateful for the time he spared to meet with me online despite the time zone difference.

I would like to express my gratitude to Dr. Željko Ivezić for his generosity in responding to my inquiry and for directing me to the work and guidance of Dr. Dejan Vinković and Dr. Dino Bektešević.

I sincerely thank Prof. Bogdan Okreša Đurić for his patience, support, and invaluable guidance throughout the development of this thesis. I am incredibly grateful for his constructive feedback and encouragement, which kept me on the right path.

I also want to thank Nick Kanas, who kindly responded to my email early in the project. Although the research ultimately took a different direction, his insights were greatly appreciated. Additionally, I extend my thanks for his contributions in *Star Maps: History, Artistry, and Cartography* [5], which inspired the early stages of this work.

Bibliography

- [1] D. Bektešević and D. Vinković, “Linear feature detection algorithm for astronomical surveys— i. algorithm description,” *Monthly Notices of the Royal Astronomical Society*, vol. 471, no. 3, pp. 2626–2641, 2017.
- [2] Wikipedia contributors, *Celestial cartography* — *Wikipedia, the free encyclopedia*, [Online; accessed 30-June-2024], 2024.
- [3] *Uranography*, Merriam-Webster.com Dictionary, 2024.
- [4] F. H. Shu, *Cosmology*, Encyclopedia Britannica, 2024.
- [5] N. Kanas, *Star Maps: History, Artistry, and Cartography*. Springer, 2019.
- [6] P. Natarajan, *Mapping the Heavens: The Radical Scientific Ideas That Reveal the Cosmos*. HarperCollins India, 2016, ISBN: 9789350297728.
- [7] Wikipedia contributors, *Nebra sky disc* — *Wikipedia, the free encyclopedia*, [Online; accessed 22-June-2024], 2024.
- [8] Wikipedia contributors, *Ptolemy’s world map* — *Wikipedia, the free encyclopedia*, [Online; accessed 23-June-2024], 2024.
- [9] L. of Congress, *The daguerreotype medium*.
- [10] S. D. S. S. (SDSS), *About sdss*, <https://skyserver.sdss.org/dr1/en/sdss/>, Accessed: January 5, 2025.
- [11] Encyclopedia Britannica, *Spectroscopy*, <https://www.britannica.com/science/spectroscopy>, Accessed: January 5, 2025.
- [12] Wikipedia contributors, *Sloan digital sky survey* — *Wikipedia, the free encyclopedia*, [Online; accessed 5-January-2025], 2024.
- [13] Wikipedia contributors, *Charge-coupled device* — *Wikipedia, the free encyclopedia*, [Online; accessed 6-January-2025], 2025.
- [14] D. L. Tucker, S. Kent, M. Richmond, *et al.*, “The sloan digital sky survey monitor telescope pipeline,” *Astronomische Nachrichten: Astronomical Notes*, vol. 327, no. 9, pp. 821–843, 2006.
- [15] A. S. Szalay, P. Kunszt, A. Thakar, J. Gray, and D. Slutz, “The sloan digital sky survey and its archive,” *arXiv preprint astro-ph/9912382*, 1999.
- [16] Wikipedia contributors, *Vera c. rubin observatory* — *Wikipedia, the free encyclopedia*, [Online; accessed 6-January-2025], 2025.

- [17] Ž. Ivezić, S. M. Kahn, J. A. Tyson, *et al.*, “Lsst: From science drivers to reference design and anticipated data products,” *The Astrophysical Journal*, vol. 873, p. 111, 2019. DOI: 10.3847/1538-4357/ab042c.
- [18] A. E. Rubin and J. N. Grossman, “Meteorite and meteoroid: New comprehensive definitions,” *Meteoritics & Planetary Science*, vol. 45, no. 1, pp. 114–122, 2010.
- [19] J. Rao, “The leonids: The lion king of meteor showers,” *WGN, Journal of the International Meteor Organization*, vol. 23, no. 4, p. 120-135, vol. 23, pp. 120–135, 1995.
- [20] P. Jenniskens, “Meteor showers and their parent comets,” in *Proceedings of the International Meteor Conference, 25th IMC, Roden, Netherlands, 2006*, 2007, pp. 56–62.
- [21] Wikipedia contributors, *Leonids — Wikipedia, the free encyclopedia*, <https://en.wikipedia.org/w/index.php?title=Leonids&oldid=1254105817>, [Online; accessed 18-January-2025], 2024.
- [22] Wikipedia contributors, *Perseids — Wikipedia, the free encyclopedia*, [Online; accessed 18-January-2025], 2025.
- [23] Wikipedia contributors, *Olympus-1 — Wikipedia, the free encyclopedia*, [Online; accessed 18-January-2025], 2024.
- [24] C. Price and M. Blum, “Elf/vlf radiation produced by the 1999 leonid meteors,” *Earth, Moon, and Planets*, vol. 82, pp. 545–554, 1998.
- [25] K. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, *et al.*, “The first data release of the sloan digital sky survey,” *The Astronomical Journal*, vol. 126, no. 4, p. 2081, 2003.
- [26] K. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, *et al.*, “The second data release of the sloan digital sky survey,” *The Astronomical Journal*, vol. 128, no. 1, p. 502, 2004.
- [27] K. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, *et al.*, “The third data release of the sloan digital sky survey,” *The Astronomical Journal*, vol. 129, no. 3, p. 1755, 2005.
- [28] J. K. Adelman-McCarthy, M. A. Agüeros, S. S. Allam, *et al.*, “The fourth data release of the sloan digital sky survey,” *The Astrophysical Journal Supplement Series*, vol. 162, no. 1, p. 38, 2006.
- [29] K. N. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, *et al.*, “The seventh data release of the sloan digital sky survey,” *The Astrophysical Journal Supplement Series*, vol. 182, no. 2, p. 543, 2009.
- [30] H. Aihara, C. A. Prieto, D. An, *et al.*, “The eighth data release of the sloan digital sky survey: First data from sdss-iii,” *The Astrophysical Journal Supplement Series*, vol. 193, no. 2, p. 29, 2011.
- [31] I. Pâris, P. Petitjean, É. Aubourg, *et al.*, “The sloan digital sky survey quasar catalog: Ninth data release,” *Astronomy & Astrophysics*, vol. 548, A66, 2012.
- [32] C. P. Ahn, R. Alexandroff, C. A. Prieto, *et al.*, “The tenth data release of the sloan digital sky survey: First spectroscopic data from the sdss-iii apache point observatory galactic evolution experiment,” *The Astrophysical Journal Supplement Series*, vol. 211, no. 2, p. 17, 2014.

- [33] S. Alam, F. D. Albareti, C. A. Prieto, *et al.*, “The eleventh and twelfth data releases of the sloan digital sky survey: Final data from sdss-iii,” *The Astrophysical Journal Supplement Series*, vol. 219, no. 1, p. 12, 2015.
- [34] F. D. Albareti, C. A. Prieto, A. Almeida, *et al.*, “The 13th data release of the sloan digital sky survey: First spectroscopic data from the sdss-iv survey mapping nearby galaxies at apache point observatory,” *The Astrophysical Journal Supplement Series*, vol. 233, no. 2, p. 25, 2017.
- [35] I. Pâris, P. Petitjean, É. Aubourg, *et al.*, “The sloan digital sky survey quasar catalog: Fourteenth data release,” *Astronomy & Astrophysics*, vol. 613, A51, 2018.
- [36] D. S. Aguado, R. Ahumada, A. Almeida, *et al.*, “The fifteenth data release of the sloan digital sky surveys: First release of manga-derived quantities, data visualization tools, and stellar library,” *The Astrophysical Journal Supplement Series*, vol. 240, no. 2, p. 23, 2019.
- [37] R. Ahumada, C. A. Prieto, A. Almeida, *et al.*, “The 16th data release of the sloan digital sky surveys: First release from the apogee-2 southern survey and full release of eboss spectra,” *The Astrophysical Journal Supplement Series*, vol. 249, no. 1, p. 3, 2020.
- [38] N. Abdurro'uf, K. Accetta, C. Aerts, *et al.*, “The seventeenth data release of the sloan digital sky surveys: Complete release of manga, mastar, and apogee-2 data,” *The Astrophysical Journal. Supplement Series*, vol. 259, no. 2, 2022.
- [39] A. Almeida, S. F. Anderson, M. Argudo-Fernández, *et al.*, “The eighteenth data release of the sloan digital sky surveys: Targeting and first spectra from sdss-v,” *The Astrophysical Journal Supplement Series*, vol. 267, no. 2, p. 44, 2023.
- [40] D. C. Wells and E. W. Greisen, “Fits—a flexible image transport system,” in *Image processing in astronomy*, 1979, p. 445.
- [41] *Sloan digital sky survey (sdss) science archive, data release 12*, Accessed: 19 January 2025.
- [42] R. B. Fisher, *Hough transform (hipr2 project)*, <https://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>, Accessed: 2025-02-15, 2003.
- [43] R. B. Fisher, *Canny edge detection (hipr2 project)*, <https://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm>, Accessed: 2025-02-15, 2003.
- [44] A. Mackey and G. Gilmore, “Surface brightness profiles and structural parameters for 53 rich stellar clusters in the large magellanic cloud,” *Monthly Notices of the Royal Astronomical Society*, vol. 338, no. 1, pp. 85–119, 2003.
- [45] D. Jewitt and K. J. Meech, “Surface brightness profiles of 10 comets,” *Astrophysical Journal, Part 1 (ISSN 0004-637X)*, vol. 317, June 15, 1987, p. 992-1001. *NASA-supported research.*, vol. 317, pp. 992–1001, 1987.
- [46] A. Krogh, “What are artificial neural networks?” *Nature biotechnology*, vol. 26, no. 2, pp. 195–197, 2008.

List of Figures

1.	Photograph of the Nebra sky disc by Frank Vincentz [7]	4
2.	Ptolemy's 150 CE World Map (redrawn in the 15 th century) [8]	5
3.	A table of star positions from Volume I of Argelander's " <i>Beobachtungen</i> ". This volume covers the area of the sky between +45 and +80 degrees declination, published in 1846. The information includes star numbers, magnitudes, right ascension, and declination. [5]	7
4.	The imaging camera of the SDSS telescope showcases the 5 rows (one for each filter) of 6 charge-coupled devices per row. [12]	10
5.	Image of a Leonid Meteor by Navicore [21]	14
6.	Image of Perseids by Ahmed abd Elkader Mohamed [22]	14
7.	frame-g-004822-5-0011.fits adjusted using Z-scale, SDSS Data Release 12 [41]	19
8.	frame-g-000109-5-0090.fits, showing a meteor trail [41]	22
9.	frame-g-000109-5-0090.fits with reduced intensity [41]	22
10.	frame-g-000109-5-0125.fits, showing a satellite trail [41]	23
11.	frame-g-000109-5-0090.fits with reduced intensity [41]	24
12.	frame-z-008149-5-0096.fits, containing a thin, faint meteor trail [41]	25
13.	frame-i-006177-3-0331.fits, containing a wide, bright meteor trail [41]	25
14.	frame-r-002566-1-0334.fits, containing a faint although wide meteor trail [41]	26
15.	frame-z-003355-6-0110.fits, containing what appears to be a trail of a rotating satellite [41]	27
16.	frame-u-002243-5-0072.fits, containing a very wide meteor trail [41]	27
17.	frame-r-000094-5-0168.fits, containing a meteor trail with branching tails [41]	28
18.	frame-g-000250-5-0189.fits altered using ZScale [41]	36
19.	Result of the plot_trails method of the <i>TrailDetector</i> class for file frame-g-000250-5-0189.fits	36
20.	frame-g-006371-5-0089.fits [41]	41

21.	All perpendicular lines visualized for meteor on image frame-g-006371-5-0089.fits	41
22.	Brightness profile taken across the perpendicular line at index 1 of meteor in image frame-g-006371-5-0089.fits	42
23.	All brightness profiles taken across perpendicular lines of meteor in image frame-g-006371-5-0089.fits	42
24.	Example of median profile created using the <i>TrailProfiler</i> class	43
25.	Main menu of the Trail Analysis Application	54
26.	Detected lines menu of the Trail Analysis Application	54
27.	Plot of the main chosen line after selecting the "Plot Main Line" button	55
28.	Plot of the perpendicular lines after selecting the "Plot All Perpendicular Lines" button	56
29.	Plot of a single profile after entering an index in the provided field and selecting the "Plot Profile at Index" button	56
30.	Plot of all of the perpendicular line profiles after selecting the "Plot All Profiles" button	57
31.	View of calculated metrics after selecting the "Analyze Median Profile" button . .	57
32.	Neural network prediction view after importing a model and selecting the "Predict on Median Profile" button	57

List of Tables

1.	SDSS Photometric Filters and Their Assigned Letter and Wavelength	9
2.	Classification report of the test neural network	44
3.	Overall Metrics - FWHM and Compactness	48
4.	Overall Metrics - AUC and Kurtosis	49
5.	Metrics for Very Likely Meteors (score > 0.8) - FWHM and Compactness	49
6.	Metrics for Very Likely Meteors (score > 0.8) - AUC and Kurtosis	49
7.	Metrics for Very Likely Non-Meteors (score < 0.2) - FWHM and Compactness	50
8.	Metrics for Very Likely Non-Meteors (score < 0.2) - AUC and Kurtosis	50
9.	Metrics for Meteors (score > 0.5) - FWHM and Compactness	50
10.	Metrics for Meteors (score > 0.5) - AUC and Kurtosis	51
11.	Metrics for Non-Meteors (score \leq 0.5) - FWHM and Compactness	51
12.	Metrics for Non-Meteors (score \leq 0.5) - AUC and Kurtosis	51
13.	Comparison of Averages of FWHM Metrics and Compactness Across Prediction Categories	52
14.	Comparison of Medians of FWHM Metrics and Compactness Across Prediction Categories	52
15.	Comparison of Averages of AUC and Kurtosis Across Prediction Categories	53
16.	Comparison of Medians of AUC and Kurtosis Across Prediction Categories	53

List of Listings

1.	Opening a FITS file and displaying the header data information	19
2.	Displaying the header data information of the primary HDU	20
3.	Imports for the <i>TrailDetector</i> class	31
4.	Constructor of the <i>TrailDetector</i> class	31
5.	The <i>detect_trails</i> method of the <i>TrailDetector</i> class	32
6.	The <i>_attempt_detection</i> method of the <i>TrailDetector</i> class	33
7.	The <i>_preprocess_bright</i> method of the <i>TrailDetector</i> class	34
8.	The <i>_preprocess_dim</i> method of the <i>TrailDetector</i> class	34
9.	The <i>_merge_lines</i> method of the <i>TrailDetector</i> class	35
10.	Example usage of the <i>TrailDetector</i> class	35
11.	Constructor of the <i>TrailProfiler</i> class	38
12.	The <i>_analyze_perpendicular_lines</i> method of the <i>TrailProfiler</i> class	39
13.	The <i>get_combined_median_profile</i> method of the <i>TrailProfiler</i> class	40
14.	Python session showing usage of the <i>TrailProfiler</i> class for various plots	40
15.	The <i>main()</i> function of the <i>test_meteor_classifier.py</i> file	45
16.	Computation of trail profile metrics for <i>frame-g-006371-5-0089.fits</i>	47

Appendices

```

1  import os
2  import numpy as np
3  import cv2
4  import matplotlib.pyplot as plt
5  from astropy.io import fits
6  from astropy.visualization import (
7      ImageNormalize,
8      PercentileInterval,
9      SqrtStretch,
10     ZScaleInterval,
11 )
12 from scipy.spatial.distance import pdist, squareform
13 from sklearn.cluster import DBSCAN
14
15 """
16 TrailDetector is a class for detecting linear trails, such as meteor streaks,
17 in astronomical FITS images. It employs edge detection, Hough line detection,
18 and DBSCAN clustering to identify candidate trails.
19 """
20
21 class TrailDetector:
22     """
23     A class for detecting and analyzing linear trails in FITS images.
24
25     This class processes astronomical FITS images using edge detection,
26     Hough transforms, and DBSCAN clustering to extract linear features
27     corresponding to potential meteor trails. It attempts two levels of
28     detection: one for bright trails and another for dim ones.
29
30     Attributes:
31         canny_params (dict): Parameters for Canny edge detection.
32         hough_params (dict): Parameters for Hough line detection.
33         dbscan_eps (float): DBSCAN clustering epsilon value.
34         dbscan_min_samples (int): Minimum samples required for DBSCAN clustering.
35         merged_lines (list): List of detected and merged line segments.
36         best_line (list): The most prominent detected trail.
37     """
38
39     def __init__(
40         self,
41         canny_params=None,
42         hough_params=None,
43         dbscan_eps=150,
44         dbscan_min_samples=2,
45     ):
46         """
47         Initializes the TrailDetector class with default or provided parameters.
48
49         Args:
50             canny_params (dict, optional): Parameters for Canny edge detection.
51             hough_params (dict, optional): Parameters for Hough line detection.
52             dbscan_eps (float, optional): DBSCAN epsilon value for clustering.

```

```

47         dbscan_min_samples (int, optional): Minimum samples required for
         ↪ clustering.
48     """
49     self.canny_params = canny_params or {"threshold1": 6, "threshold2": 18}
50     self.hough_params = hough_params or {
51         "rho": 1,
52         "theta": np.pi / 180,
53         "threshold": 250,
54         "minLineLength": 300,
55         "maxLineGap": 150,
56     }
57     self.dbscan_eps = dbscan_eps
58     self.dbscan_min_samples = dbscan_min_samples
59     self.merged_lines = []
60     self.best_line = None

61     def detect_trails(self, fits_file, save_processed=False,
62     ↪ processed_dir="processed_images"):
63         """
64         Detects linear trails in a given FITS file using image processing.
65
66         Args:
67             fits_file (str): Path to the FITS file.
68             save_processed (bool, optional): If True, saves preprocessed images.
69             processed_dir (str, optional): Directory to store processed images.
70
71         Returns:
72             list: A list containing the best detected line, or an empty list if none
73             ↪ are found.
74         """
75         if save_processed:
76             os.makedirs(processed_dir, exist_ok=True)
77
78         lines_bright = self._attempt_detection(
79             fits_file=fits_file,
80             mode='bright',
81             save_processed=save_processed,
82             processed_dir=processed_dir
83         )
84         if lines_bright:
85             self.merged_lines = lines_bright
86             self.best_line = self._choose_best_line(lines_bright)
87             return [self.best_line]
88
89         lines_dim = self._attempt_detection(
90             fits_file=fits_file,
91             mode='dim',
92             save_processed=save_processed,
93             processed_dir=processed_dir
94         )
95         if lines_dim:
96             self.merged_lines = lines_dim
97             self.best_line = self._choose_best_line(lines_dim)

```

```

92         return [self.best_line]

93     print(f"No trails detected in {fits_file} after both attempts.")
94     return []

95     def _attempt_detection(self, fits_file, mode, save_processed=False,
↪ processed_dir="processed_images"):
96         """
97         Attempts to detect trails in an image by preprocessing and applying edge
↪ detection.

98         Args:
99             fits_file (str): Path to the FITS file.
100            mode (str): Either "bright" or "dim", determining preprocessing
↪ strategy.
101            save_processed (bool, optional): If True, saves processed images.
102            processed_dir (str, optional): Directory to store processed images.

103         Returns:
104            list: Merged detected line segments.
105         """
106         with fits.open(fits_file) as hdul:
107             image_data = hdul[0].data

108         if mode == 'bright':
109             processed = self._preprocess_bright(image_data)
110         else:
111             processed = self._preprocess_dim(image_data)

112         if save_processed:
113             base_no_ext = os.path.splitext(os.path.basename(fits_file))[0]
114             out_name = f"{base_no_ext}_{mode}_processed.png"
115             out_path = os.path.join(processed_dir, out_name)
116             cv2.imwrite(out_path, processed)
117             print(f"Saved pre-processed '{mode}' image to: {out_path}")

118         edges = cv2.Canny(processed, **self.canny_params)
119         lines = cv2.HoughLinesP(edges, **self.hough_params)
120         if lines is None:
121             return []

122         lines_data = self._process_lines(lines)
123         if len(lines_data) == 0:
124             return []

125         merged = self._merge_lines(lines_data)
126         return merged

127     def _preprocess_bright(self, image_data):
128         """
129         Preprocesses for detection of bright trails by applying normalization and
↪ histogram equalization.

```

```

130     Args:
131         image_data (numpy.ndarray): FITS image data.

132     Returns:
133         numpy.ndarray: Processed image.
134     """
135     norm = ImageNormalize(image_data, interval=ZScaleInterval())
136     float_img = norm(image_data)
137     gray_8u = cv2.convertScaleAbs(float_img)
138     equ = cv2.equalizeHist(gray_8u)
139     kernel = np.ones((4, 4), np.uint8)
140     equ = cv2.erode(equ, kernel, iterations=1)
141     equ = cv2.dilate(equ, kernel, iterations=1)
142     return equ

143 def _preprocess_dim(self, image_data):
144     """
145     Preprocesses for detection dim trails by applying normalization and
146     ↪ histogram equalization.

147     Args:
148         image_data (numpy.ndarray): FITS image data.

149     Returns:
150         numpy.ndarray: Processed image.
151     """
152     norm = ImageNormalize(
153         image_data,
154         interval=PercentileInterval(1, 99),
155         stretch=SqrtStretch()
156     )
157     float_img = norm(image_data)
158     gray_8u = cv2.convertScaleAbs(float_img)
159     equ = cv2.equalizeHist(gray_8u)
160     kernel = np.ones((3, 3), np.uint8)
161     equ = cv2.erode(equ, kernel, iterations=1)
162     kernel = np.ones((9, 9), np.uint8)
163     equ = cv2.dilate(equ, kernel, iterations=1)
164     return equ

165 def _process_lines(self, lines):
166     """
167     Converts raw Hough transform line detections into structured data.

168     Args:
169         lines (numpy.ndarray): An array of detected lines from HoughLinesP.

170     Returns:
171         numpy.ndarray: An array containing structured line data, where each row
172         ↪ consists of:
173         - x1, y1, x2, y2: Start and end points of the line.
174         - length: Euclidean distance between start and end points.
175         - angle: Angle of the line with respect to the x-axis.

```

```

174         """
175         lines_data = []
176         for line in lines:
177             x1, y1, x2, y2 = line[0]
178             length = np.hypot(x2 - x1, y2 - y1)
179             angle = np.arctan2((y2 - y1), (x2 - x1))
180             lines_data.append([x1, y1, x2, y2, length, angle])
181         return np.array(lines_data)

182     def _choose_best_line(self, lines):
183         """
184         Selects the longest detected line from the given list of merged lines.

185         Args:
186             lines (list): A list of detected and merged lines, where each line is
187                 represented as [x1, y1, x2, y2].

188         Returns:
189             list or None: The longest detected line in the form [x1, y1, x2, y2],
190                 or None if no lines are found.
191         """
192         if not lines:
193             return None

194         best_line = None
195         best_length = 0

196         for line in lines:
197             x1, y1, x2, y2 = line
198             length = np.hypot(x2 - x1, y2 - y1)
199             if length > best_length:
200                 best_length = length
201                 best_line = line

202         return best_line

203     def _merge_lines(self, lines_data):
204         """
205         Merges similar lines using DBSCAN clustering.

206         Args:
207             lines_data (numpy.ndarray): Array of detected lines.

208         Returns:
209             list: List of merged lines.
210         """
211         def custom_distance(l1, l2):
212             pos_dist = np.hypot(l1[0] - l2[0], l1[1] - l2[1])
213             angle_dist = abs(l1[5] - l2[5])
214             return pos_dist + 50 * angle_dist

215         dist_matrix = squareform(pdist(lines_data, metric=custom_distance))
216         clustering = DBSCAN(

```

```

217         eps=self.dbscan_eps,
218         min_samples=self.dbscan_min_samples,
219         metric="precomputed"
220     ).fit(dist_matrix)

221     merged_lines = []
222     for label in set(clustering.labels_):
223         if label == -1:
224             continue
225         cluster = lines_data[clustering.labels_ == label]
226         x1_mean = int(cluster[:, 0].mean())
227         y1_mean = int(cluster[:, 1].mean())
228         x2_mean = int(cluster[:, 2].mean())
229         y2_mean = int(cluster[:, 3].mean())
230         merged_lines.append([x1_mean, y1_mean, x2_mean, y2_mean])
231     return merged_lines

232     def plot_trails(self, fits_file, output_file=None):
233         """
234         Plots the detected trail on the original FITS image.

235         Args:
236             fits_file (str): Path to the FITS file.
237             output_file (str, optional): If specified, saves the plot to the given
238             ↪ file.
239         """
240         if not self.best_line:
241             print("No best line to plot.")
242             return
243         with fits.open(fits_file) as hdul:
244             image_data = hdul[0].data
245             plt.figure(figsize=(10, 10))
246             norm = ImageNormalize(
247                 image_data,
248                 interval=PercentileInterval(1, 99),
249                 stretch=SqrtStretch()
250             )
251             plt.imshow(image_data, cmap='gray', origin='lower', norm=norm)
252             x1, y1, x2, y2 = self.best_line
253             plt.plot([x1, x2], [y1, y2], color='cyan', linewidth=2)
254             plt.title("Detected Trail")
255             plt.xlabel("X pixel")
256             plt.ylabel("Y pixel")
257             plt.colorbar()
258             if output_file:
259                 plt.savefig(output_file, dpi=300, bbox_inches='tight')
260             plt.show()

```

```

1  import os
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from astropy.visualization import ImageNormalize, PercentileInterval, SqrtStretch

```



```

5  from scipy.ndimage import map_coordinates

6  class TrailProfiler:
7      """
8      The TrailProfiler class measures brightness distributions along lines
9      perpendicular to a user-defined trail in an astronomical FITS image.
10     It also provides utility methods to visualize and analyze these profiles,
11     including metrics for further classification.
12     """

13     def __init__(self, fits_file, point1, point2, output_dir="trail_profiles"):
14         """
15         Initializes the profiler with a path to a FITS file and two points
16         (x0, y0) and (x1, y1) defining the main trail.

17         Args:
18             fits_file (str): Path to the FITS file containing the image data.
19             point1 (tuple): Coordinates of the first trail endpoint (x0, y0).
20             point2 (tuple): Coordinates of the second trail endpoint (x1, y1).
21             output_dir (str): Directory where plots and data will be saved.
22         """
23         self.fits_file = fits_file
24         self.point1 = point1
25         self.point2 = point2
26         self.output_dir = output_dir

27         self.image_data = None
28         self.normalized_data = None
29         self.brightness_profiles = []
30         self.line_coordinates = []

31         self._load_fits_data()
32         self._create_output_dir()
33         self._sample_perpendicular_profiles()

34     def _load_fits_data(self):
35         """
36         Loads the FITS image data from disk and applies a brightness normalization.
37         """
38         from astropy.io import fits

39         with fits.open(self.fits_file) as hdul:
40             self.image_data = hdul[0].data

41         norm = ImageNormalize(
42             self.image_data,
43             interval=PercentileInterval(99.5),
44             stretch=SqrtStretch()
45         )
46         self.normalized_data = norm(self.image_data)

47     def _create_output_dir(self):
48         """

```

```

49     Ensures that the directory for output files exists.
50     """
51     if not os.path.exists(self.output_dir):
52         os.makedirs(self.output_dir)

53     def _sample_perpendicular_profiles(self, num_perp_lines=10,
54     ↪ half_line_length=100, sampling_step=0.1):
55         """
56         Draws multiple perpendicular lines along the main trail, then samples
57         brightness values at regularly spaced points on each line. If any line
58         would go outside the image boundaries, it is skipped entirely.

59         Args:
60         num_perp_lines (int): Number of perpendicular slices to take along the
61         ↪ trail.
62         half_line_length (float): Half the length of each perpendicular line in
63         ↪ pixels.
64         sampling_step (float): Step size (in pixels) used for sampling
65         ↪ brightness.
66         """
67         x0, y0 = self.point1
68         x1, y1 = self.point2
69         main_length = np.hypot(x1 - x0, y1 - y0)

70         perp_dx = -(y1 - y0) / main_length
71         perp_dy = (x1 - x0) / main_length

72         slice_spacing = main_length / (num_perp_lines - 1) if num_perp_lines > 1
73         ↪ else main_length

74         height, width = self.image_data.shape
75         self.brightness_profiles.clear()
76         self.line_coordinates.clear()

77         for i in range(num_perp_lines):
78             t = i * slice_spacing / main_length
79             x_center = x0 + t * (x1 - x0)
80             y_center = y0 + t * (y1 - y0)

81             x_start = x_center - half_line_length * perp_dx
82             y_start = y_center - half_line_length * perp_dy
83             x_end = x_center + half_line_length * perp_dx
84             y_end = y_center + half_line_length * perp_dy

85             if not (0 <= x_start <= (width - 1) and 0 <= x_end <= (width - 1) and
86             ↪ 0 <= y_start <= (height - 1) and 0 <= y_end <= (height - 1)):
87                 continue

88             num_samples = int(2 * half_line_length / sampling_step)
89             x_coords = np.linspace(x_start, x_end, num_samples)
90             y_coords = np.linspace(y_start, y_end, num_samples)
91             coords_for_sampling = np.vstack((y_coords, x_coords))

```

```

87         brightness_vals = map_coordinates(self.image_data, coords_for_sampling,
      ↪ order=3)

88         data_min = np.min(self.image_data)
89         data_max = np.max(self.image_data)
90         if data_max == data_min:
91             norm_vals = np.zeros_like(brightness_vals)
92         else:
93             norm_vals = (brightness_vals - data_min) / (data_max - data_min)

94         self.brightness_profiles.append(norm_vals)
95         self.line_coordinates.append((x_start, y_start), (x_end, y_end))

96     def plot_perpendicular_profiles(self, save_plot=False,
      ↪ filename="perp_profiles"):
97         """
98         Plots all sampled brightness profiles in a single figure.

99         Args:
100             save_plot (bool): If True, saves the plot to disk.
101             filename (str): Filename (without extension) for the saved plot.
102         """
103         plt.figure(figsize=(12, 8))
104         for i, profile in enumerate(self.brightness_profiles):
105             plt.plot(profile, label=f"Profile {i+1}")

106         if self.brightness_profiles:
107             center_index = len(self.brightness_profiles[0]) // 2
108             plt.axvline(center_index, color='red', linestyle='--', label='Trail
      ↪ Center')

109         plt.xlabel("Sample Index Along Perpendicular")
110         plt.ylabel("Normalized Brightness")
111         plt.title("Perpendicular Brightness Profiles")
112         plt.legend()
113         plt.grid()

114         if save_plot:
115             outpath = os.path.join(self.output_dir, f"{filename}.png")
116             plt.savefig(outpath, dpi=300, bbox_inches="tight")
117             print(f"Perpendicular profiles plot saved to {outpath}")

118         plt.show()

119     def plot_main_line(self, save_plot=False, filename="main_line"):
120         """
121         Displays the original image with the main trail (point1-point2) overlaid.

122         Args:
123             save_plot (bool): If True, saves the plot to disk.
124             filename (str): Filename (without extension) for the saved plot.
125         """
126         x0, y0 = self.point1

```

```

127     x1, y1 = self.point2

128     plt.figure(figsize=(10, 10))
129     plt.imshow(self.normalized_data, cmap="gray", origin="lower")
130     plt.plot([x0, x1], [y0, y1], color="red", linestyle="-", linewidth=2,
131             ↪ label="Main Trail")
132     plt.title("Main Trail on Image")
133     plt.xlabel("X Pixel")
134     plt.ylabel("Y Pixel")
135     plt.legend()

136     if save_plot:
137         outpath = os.path.join(self.output_dir, f"{filename}.png")
138         plt.savefig(outpath, dpi=300, bbox_inches="tight")
139         print(f"Main line plot saved to {outpath}")

140     plt.show()

141 def plot_all_perp_lines(self, save_plot=False, filename="all_perp_lines"):
142     """
143     Displays the original image with all perpendicular lines overlaid.
144
145     Args:
146         save_plot (bool): If True, saves the plot to disk.
147         filename (str): Filename (without extension) for the saved plot.
148     """
149     if not self.line_coordinates:
150         raise ValueError("No perpendicular lines to plot. Run
151         ↪ _sample_perpendicular_profiles first.")

152     plt.figure(figsize=(10, 10))
153     plt.imshow(self.normalized_data, cmap="gray", origin="lower")

154     for i, (start, end) in enumerate(self.line_coordinates):
155         xs = [start[0], end[0]]
156         ys = [start[1], end[1]]
157         plt.plot(xs, ys, linestyle="--", linewidth=1, label=f"Line {i+1}")

158     plt.title("All Perpendicular Lines on Image")
159     plt.xlabel("X Pixel")
160     plt.ylabel("Y Pixel")
161     plt.legend(loc="upper right", fontsize="small")

162     if save_plot:
163         outpath = os.path.join(self.output_dir, f"{filename}.png")
164         plt.savefig(outpath, dpi=300, bbox_inches="tight")
165         print(f"All perpendicular lines plot saved to {outpath}")

166     plt.show()

167 def calculate_median_profile(self, profiles=None):
168     """
169     Calculates the median brightness profile across multiple samples.

```

```

167     Args:
168         profiles (list or None): Optional list of arrays containing brightness
           ↪ profiles.
169
           Defaults to self.brightness_profiles if None.
170
171     Returns:
172         numpy.ndarray: Array of median brightness values.
173     """
174     chosen_profiles = profiles if profiles is not None else
           ↪ self.brightness_profiles
175     if not chosen_profiles:
176         raise ValueError("No brightness profiles available for median
           ↪ calculation.")
177     stacked = np.vstack(chosen_profiles)
178     median_vals = np.median(stacked, axis=0)
179     return median_vals
180
181 def plot_median_profile(self, profiles=None, save_plot=False,
           ↪ filename="median_profile"):
182     """
183     Displays and optionally saves the median brightness profile.
184
185     Args:
186         profiles (list or None): Optional list of arrays containing brightness
           ↪ profiles.
187
           Defaults to self.brightness_profiles if None.
188         save_plot (bool): If True, saves the plot to disk.
189         filename (str): Filename (without extension) for the saved plot.
190     """
191     median_vals = self.calculate_median_profile(profiles=profiles)
192
193     plt.figure(figsize=(12, 8))
194     plt.plot(median_vals, label="Median Brightness Profile", color="blue")
195     center_idx = len(median_vals) // 2
196     plt.axvline(center_idx, color="red", linestyle="--", label="Approx. Trail
           ↪ Center")
197     plt.xlabel("Sample Index")
198     plt.ylabel("Normalized Brightness")
199     plt.title("Combined Median Profile")
200     plt.legend()
201     plt.grid()
202
203     if save_plot:
204         outpath = os.path.join(self.output_dir, f"{filename}.png")
205         plt.savefig(outpath, dpi=300, bbox_inches="tight")
206         print(f"Median profile plot saved to {outpath}")
207
208     plt.show()
209
210 def calculate_fwhm(self, profile, half_max_factor=0.5):
211     """
212     Determines the Full Width at Half Maximum (FWHM) of a profile.

```

```

206     Args:
207         profile (numpy.ndarray): Brightness profile of the trail.
208         half_max_factor (float): Fraction of the max intensity defining "half
    ↪ max."

209     Returns:
210         float: The FWHM in index units.
211     """
212     peak = np.max(profile)
213     if peak == 0:
214         return 0
215     half_max = peak * half_max_factor
216     indices = np.where(profile >= half_max)[0]
217     if len(indices) < 2:
218         return 0
219     return indices[-1] - indices[0]

220 def calculate_fwhm_default(self, profile):
221     """
222     Computes the default FWHM using half_max_factor = 0.5.

223     Returns:
224         float: FWHM (default).
225     """
226     return self.calculate_fwhm(profile, half_max_factor=0.5)

227 def calculate_fwhm_07(self, profile):
228     """
229     Computes the FWHM at 70% of the peak intensity.

230     Returns:
231         float: FWHM with factor 0.7.
232     """
233     return self.calculate_fwhm(profile, half_max_factor=0.7)

234 def calculate_fwhm_095(self, profile):
235     """
236     Computes the FWHM at 95% of the peak intensity.

237     Returns:
238         float: FWHM with factor 0.95.
239     """
240     return self.calculate_fwhm(profile, half_max_factor=0.95)

241 def calculate_auc_med(self, profile):
242     """
243     Computes the area under the curve (AUC) for a brightness profile normalized
244     by the median value of the entire FITS image.

245     Returns:
246         float: AUC with normalization by the median (CoverageMed).
247     """

```

```

248     m = np.median(self.image_data)
249     if m == 0:
250         return 0
251     normalized_profile = profile / m
252     return np.sum(normalized_profile)

253 def calculate_auc_peak(self, profile):
254     """
255     Computes the area under the curve (AUC) for a brightness profile normalized
256     by the peak value of the median profile.

257     Returns:
258         float: AUC with normalization by the profile's peak (CoveragePeak).
259     """
260     M = np.max(profile)
261     if M == 0:
262         return 0
263     normalized_profile = profile / M
264     return np.sum(normalized_profile)

265 def calculate_auc_full(self, profile):
266     """
267     Computes the area under the curve (AUC) for a brightness profile normalized
268     by the maximum pixel value of the entire FITS image.

269     Returns:
270         float: AUC with normalization by the global maximum (CoverageFull).
271     """
272     M_fits = np.max(self.image_data)
273     if M_fits == 0:
274         return 0
275     normalized_profile = profile / M_fits
276     return np.sum(normalized_profile)

277 def calculate_kurtosis(self, profile):
278     """
279     Calculates the weighted kurtosis of a brightness profile.
280     The brightness values serve as weights for the positional values.

281     Returns:
282         float: Kurtosis of the profile.
283     """
284     x = np.arange(len(profile))
285     weights = profile
286     total_weight = np.sum(weights)
287     if total_weight == 0:
288         return 0
289     mu = np.sum(x * weights) / total_weight
290     sigma2 = np.sum(((x - mu) ** 2) * weights) / total_weight
291     if sigma2 == 0:
292         return 0
293     sigma = np.sqrt(sigma2)
294     fourth_moment = np.sum(((x - mu) ** 4) * weights) / total_weight

```

```

295     return fourth_moment / (sigma ** 4) - 3

296 def calculate_gaussian_kurtosis(self, profile):
297     """
298     Fits the given brightness profile to a Gaussian function and computes the
299     ↪ weighted kurtosis
300     of the fitted profile.
301
302     Args:
303     profile (numpy.ndarray): The brightness profile to be fitted.
304
305     Returns:
306     float: The kurtosis of the fitted Gaussian profile, or None if the
307     ↪ fitting fails.
308     """
309     from scipy.optimize import curve_fit
310
311     def gaussian(x, A, mu, sigma):
312         return A * np.exp(-((x - mu) ** 2) / (2 * sigma ** 2))
313
314     x = np.arange(len(profile))
315     initial_guess = [np.max(profile), np.argmax(profile), len(profile) / 4]
316
317     try:
318         params, _ = curve_fit(gaussian, x, profile, p0=initial_guess)
319     except Exception as e:
320         print(f"[WARNING] Gaussian fitting failed: {e}.")
321         return None
322
323     fitted_profile = gaussian(x, *params)
324
325     total_weight = np.sum(fitted_profile)
326     if total_weight == 0:
327         return 0
328     mu_weighted = np.sum(x * fitted_profile) / total_weight
329     sigma2 = np.sum(((x - mu_weighted) ** 2) * fitted_profile) / total_weight
330     if sigma2 == 0:
331         return 0
332     sigma_val = np.sqrt(sigma2)
333     fourth_moment = np.sum(((x - mu_weighted) ** 4) * fitted_profile) /
334     ↪ total_weight
335     gaussian_kurtosis = fourth_moment / (sigma_val ** 4) - 3
336     return gaussian_kurtosis
337
338 def remove_profile_by_index(self, index):
339     """
340     Removes one brightness profile and its associated line coordinates.
341
342     Args:
343     index (int): 1-based index of the profile to remove.
344     """
345     idx = index - 1

```



```

334     if idx < 0 or idx >= len(self.brightness_profiles):
335         raise IndexError(f"Index {index} out of range. Valid: 1 to
           ↪ {len(self.brightness_profiles)}")
336     del self.brightness_profiles[idx]
337     del self.line_coordinates[idx]
338     print(f"Removed brightness profile and line at index {index}.")

339 def plot_profile_by_index(self, index, save_plot=False,
           ↪ filename="profile_by_index"):
340     """
341     Plots a single brightness profile from the stored list by index.

342     Args:
343         index (int): 1-based index of the profile to plot.
344         save_plot (bool): If True, saves the plot to disk.
345         filename (str): Filename (without extension) for the saved plot.
346     """
347     idx = index - 1
348     if idx < 0 or idx >= len(self.brightness_profiles):
349         raise IndexError(f"Index {index} out of range. Valid: 1 to
           ↪ {len(self.brightness_profiles)}")
350     profile = self.brightness_profiles[idx]
351     plt.figure(figsize=(10, 6))
352     plt.plot(profile, label=f"Profile {index}", color="blue")
353     plt.axvline(len(profile) // 2, color="red", linestyle="--", label="Approx.
           ↪ Trail Center")
354     plt.xlabel("Sample Index Along Perpendicular")
355     plt.ylabel("Normalized Brightness")
356     plt.title(f"Brightness Profile {index}")
357     plt.legend()
358     plt.grid()
359     if save_plot:
360         outpath = os.path.join(self.output_dir, f"{filename}_index_{index}.png")
361         plt.savefig(outpath, dpi=300, bbox_inches="tight")
362         print(f"Profile {index} plot saved to {outpath}")
363     plt.show()

364 def plot_line_by_index(self, index, save_plot=False, filename="line_by_index"):
365     """
366     Overlays a single perpendicular line on the normalized image.

367     Args:
368         index (int): 1-based index of the line to plot.
369         save_plot (bool): If True, saves the plot to disk.
370         filename (str): Filename (without extension) for the saved plot.
371     """
372     idx = index - 1
373     if idx < 0 or idx >= len(self.line_coordinates):
374         raise IndexError(f"Index {index} out of range. Valid: 1 to
           ↪ {len(self.line_coordinates)}")
375     start, end = self.line_coordinates[idx]
376     plt.figure(figsize=(10, 10))
377     plt.imshow(self.normalized_data, cmap="gray", origin="lower")

```

```

378     plt.plot([start[0], end[0]], [start[1], end[1]], color="cyan",
379             ↪ linestyle="--", linewidth=2, label=f"Line {index}")
380 plt.title(f"Perpendicular Line for Profile {index}")
381 plt.xlabel("X Pixel")
382 plt.ylabel("Y Pixel")
383 plt.legend()
384 if save_plot:
385     outpath = os.path.join(self.output_dir, f"{filename}_index_{index}.png")
386     plt.savefig(outpath, dpi=300, bbox_inches="tight")
387     print(f"Line {index} plot saved to {outpath}")
388 plt.show()
389
390 def extend_line_to_image_edges(self, x0, y0, x1, y1):
391     """
392     Extends a line segment so that it intersects the image boundaries,
393     ensuring the endpoints remain within valid pixel coordinates.
394
395     Args:
396         x0, y0 (float): Start of the line.
397         x1, y1 (float): End of the line.
398
399     Returns:
400         tuple: (x0_extended, y0_extended, x1_extended, y1_extended)
401     """
402     dx, dy = x1 - x0, y1 - y0
403     height, width = self.image_data.shape
404
405     if dx != 0:
406         t_left = -x0 / dx
407         t_right = (width - 1 - x0) / dx
408     else:
409         t_left = -np.inf
410         t_right = np.inf
411
412     if dy != 0:
413         t_top = -y0 / dy
414         t_bottom = (height - 1 - y0) / dy
415     else:
416         t_top = -np.inf
417         t_bottom = np.inf
418
419     t_min = max(min(t_left, t_right), min(t_top, t_bottom))
420     t_max = min(max(t_left, t_right), max(t_top, t_bottom))
421     x0_ext = x0 + t_min * dx
422     y0_ext = y0 + t_min * dy
423     x1_ext = x0 + t_max * dx
424     y1_ext = y0 + t_max * dy
425     x0_ext = np.clip(x0_ext, 0, width - 1)
426     x1_ext = np.clip(x1_ext, 0, width - 1)
427     y0_ext = np.clip(y0_ext, 0, height - 1)
428     y1_ext = np.clip(y1_ext, 0, height - 1)
429     return x0_ext, y0_ext, x1_ext, y1_ext

```