Master's thesis
Master's Programme in Computer Science

Exoplanet Detection with Deep Learning

Christian Cardin

March 19, 2025

Supervisor(s): Haris Andras
Holmberg Daniel
Roos Teemu
Tuomi Mikko

Examiner(s): Roos Teemu
Tuomi Mikko

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Christian Cardin

Työn nimi — Arbetets titel — Title

Exoplanet Detection with Deep Learning

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Master's thesis	March 19, 2025	67+10

Tiivistelmä — Referat — Abstract

The detection and characterization of exoplanet transit signals from stellar light curves are fundamental for understanding planetary systems and their potential habitability. Traditional analytical methods, while effective, are often computationally intensive and slow. This thesis explores the application of modern deep learning techniques to accelerate and enhance the analysis of light curve data, focusing on the detection and characterization of exoplanets based on the transit method.

A multi-task deep learning paradigm is applied to integrate detection and characterization into a unified framework. Four deep learning models of increasing architectural complexity are developed, incorporating fully connected layers, convolutional layers, and transformers. The models are trained and evaluated using 2-minute cadence Presearch Data Conditioning Simple Aperture Photometry (PDCSAP) light curves from the Transiting Exoplanet Survey Satellite (TESS) satellite, augmented with synthetic planetary transit signals to create a curated supervised dataset.

In total, twelve specialized models were trained to target three categories of exoplanets: Jupiter-like, Neptune-like, and Mini-Neptune-like planets. The results demonstrate excellent detection capabilities for Jupiter-like planets and very good performance for Neptune-like planets. While the predicted transit midpoints and orbital periods lack the precision required for direct use, they might provide valuable initial estimates to constrain classical parameter estimation tools.

The high classification accuracy of the proposed models positions them as effective preliminary screening tools for new light curve data from ongoing and future space missions. Furthermore, the methodologies developed in this work have potential applications in other scientific and industrial domains, extending the impact of deep learning beyond astrophysics.

Avainsanat — Nyckelord — Keywords

deep learning, machine learning, exoplanets, astronomy, light curve, time-series, cnn, transformers

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Little Adventurer, where are you going?
Between the stars, into the unknowing.
What are you seeking, why do you care?
The Wonders await the mighty who dare.

Lifted the anchor, hoisted the sail,
For now begins the Adventurer's Tale,
Filled with Passion, with glimmering eyes,
To chart the course and traverse the skies.

Wandering sailor, I bid you farewell;
May you return with wonders to tell.
What lies beyond the infinite Voids,
Boundless as the Universe, of light devoid?

Honor to thee, the free and the brave,
Who cherish the mission, a path to pave,
With heart and mind that knows no limit,
This is the Journey of the Curious Spirit.

— Cabbage Adventures

Contents

1. Introduction

In its journey towards understanding, humankind has always lifted its eyes to the sky, searching for its place in the Universe. From ancient civilizations tracking the movement of celestial bodies to modern astronomers probing the depths of space, our fascination with the cosmos has driven centuries of exploration and discovery. Over time, as techniques and technology evolved, so too has our understanding of the Universe: blurred smudges in the sky have transformed into nebulae and star clusters; and what once appeared as mere dots on photographic film are now wonderfully resolved galaxies, revealing the incredible complexity and diversity of the cosmos. One of the most profound questions that has long intrigued scientists is whether our solar system is unique, or if there are other planetary systems beyond our own that might harbor life. This question remained unanswered until 1992, when the first exoplanet (a planet orbiting a star other than our Sun) was discovered and confirmed[†] by studying tiny irregularities in the pulsation of the millisecond pulsar PSR1257+12 [51]. This groundbreaking discovery marked the beginning of a new era in astronomy: one focused not only on understanding stars and galaxies, but also on exploring the planets that orbit them.

Since that first discovery, the field of exoplanet research has expanded rapidly, with a multitude of planets being detected using various techniques. Among these, the most popular methods include the transit method, radial velocity, and direct imaging. In 1999, the first multi-planet system was identified, demonstrating that planetary systems could be as complex as our own. Just two years later, in 2001, the first exoplanet located fully within the habitable zone of its star was discovered [39], raising the possibility that such worlds could support life and, in 2005, astronomers achieved the first direct observation of an exoplanet using the ground-based Very Large Telescope [6].

The launch of the Kepler Space Telescope in 2009 marked a significant leap forward in our ability to detect exoplanets [3]. For the first time, a space-based mission was dedicated entirely to searching for exoplanets using the transit method. Kepler's

[†]The first exoplanet was discovered in 1988 [4], but confirmed only in 2002 by later studies [16]

low-cadence light curve data revolutionized the field, contributing to the discovery of thousands of exoplanets over the years, more than any mission before it. The success of Kepler was followed by additional missions such as the Transiting Exoplanet Survey Satellite (TESS) [37], which continue to provide valuable insights today.

As of December 18th 2024, astronomers have confirmed the existence of 5,806 exoplanets [31], with an additional 7,351 candidate planets awaiting verification only from TESS data [45]. And as we look to the future, the new generation of ground and space-based telescopes such as PLATO, the Roman Space Telescope, and the Extremely Large Telescope, will further accelerate the field of exoplanet research by generating terabytes of data every day, posing significant challenges for traditional methods of analysis. The enormous volume and complexity of this data will require the development of faster and more sophisticated technology to enable the scientific community to keep pace with the research.

In recent years, the advent of machine learning has brought an important boost to the field. Various machine learning methods have proven successful in classifying telescope data to determine whether it contains evidence of a transiting exoplanet. However, the precise characterization of a planet's orbital period and the midpoint of its transit remains a difficult task, especially in the presence of complex noise generated by the host star's intrinsic variability and the occurrence of unpredictable phenomena like star spots and flares.

The goal of this thesis is to train state-of-the-art deep learning architectures on light curve data collected from space telescopes. The research focuses on three key objectives: first, to classify whether a light curve contains a transit event; second, to accurately extract the midpoint and period of any detected transits; and lastly, to measure the performance of the model's prediction. The thesis is structured as follows: Chapter 2 introduces the common methods of exoplanet discovery and introduces the light curve data. Chapter 3 presents the data collection pipeline and the insights gathered from the data exploration process. Chapter 4 delves into deep learning, from general concepts to the more advanced models, including a literature review of deep learning applications in astronomy. Chapter 5 covers the development and implementation of the models for exoplanet detection, synthetic dataset generation and training details. Chapter 6 compares the models' performance on the dataset. Finally, Chapter 7 summarizes the research contributions, addressing limitations and proposing directions for future work. An appendix provides additional technical details and supporting materials.

2. Exoplanets and How To Find Them

An exoplanet is a planetary body that is gravitationally bound in orbit around a star other than our Sun. Beyond conventional exoplanets, studies [30] have found isolated planetary mass objects that drift in interstellar space, unbound to any star system. These objects likely formed within a star system but were later ejected due to gravitational interactions with other bodies in the system. Some exoplanets orbit multiple stars, such as binary or ternary star systems [9], while others may orbit stellar remnants like neutron stars, and there are speculations about planets forming in the accretion disks of black holes [20].

According to simulations on star formation [7], it is likely that most stars in the Milky Way host at least one planet, as planets typically form in the same protoplanetary disk as their parent star. With hundreds of billions of stars in our galaxy, this suggests the existence of an enormous number of planets, some of which could be potentially habitable. If this estimation is accurate, it raises the exciting possibility of finding exoplanets that support life. Especially, it is conjectured that an exoplanet orbiting within its host star's "habitable zone" is more likely to be habitable, where the equilibrium temperature is just right to allow liquid water to exist on the planet's surface. The equilibrium temperature is determined by the incoming radiation from the star, and planets within this zone are prime candidates in the search for life as we know it, since the presence of liquid water is necessary for it to develop and evolve. However, it is worth noting that habitability is not strictly limited to surface conditions. Even planets far from their host stars may harbor life underground, sustained by internal heat or chemical reactions.

In addition to the search for habitable planets, the study of exoplanets contributes to our understanding of planetary formation processes, offering valuable insights into the formation of our own solar system. The discovery of planets in different types of systems allows us to test and refine theories of planetary system development. Over the

years, astronomers have developed several methods to detect exoplanets, with varying degrees of success. These methods include radial velocity measurements, microlensing and direct imaging, but by far the most successful detection method has been the transit method. This technique involves observing the periodic dimming of a star as a planet passes in front of it, momentarily blocking a small portion of the star's light. Since the launch of space telescopes like Kepler [3] and TESS [37], the transit method has led to the discovery of thousands of exoplanets. Figure 2.1 shows the count of exoplanets known for each year, together with the contribution of different methods to the discovery count.

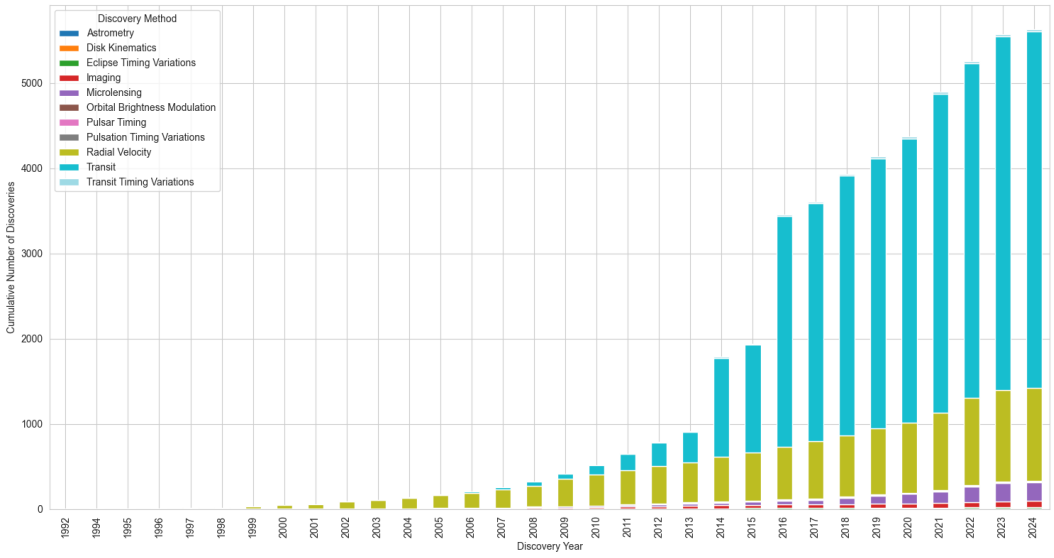


Figure 2.1: Confirmed exoplanets count by year and by method

2.1 The Transit Method

The transit method is one of the most effective techniques for detecting exoplanets, particularly those that are large and orbiting close to their host stars. It works by observing the periodic dimming of a star's light as a planet passes between the star and the observer, an event referred to as a "transit". This brief reduction in brightness occurs because the planet eclipses a small portion of the star's light, effectively creating a temporary dip in the star's apparent luminosity, as shown in Figure 2.2. Moreover, when the planet moves behind the star, there is an apparent increase in luminosity due to the light reflected by the planet's surface together with a second, smaller dip known as occultation.

By studying the shape of the transit dip, astronomers can learn important information about the planet and its host star such as the planet's size, mass, orbital distance,

and inclination relative to the observer [41]. Although the occultation depth is much smaller than the main transit depth, it provides insights into the planet's reflectivity and atmosphere. Unfortunately, due to the random and uniform orientation of planetary orbital planes, only a small fraction of exoplanets are detectable via this method, as the alignment must be just right for the transit to be visible from Earth.

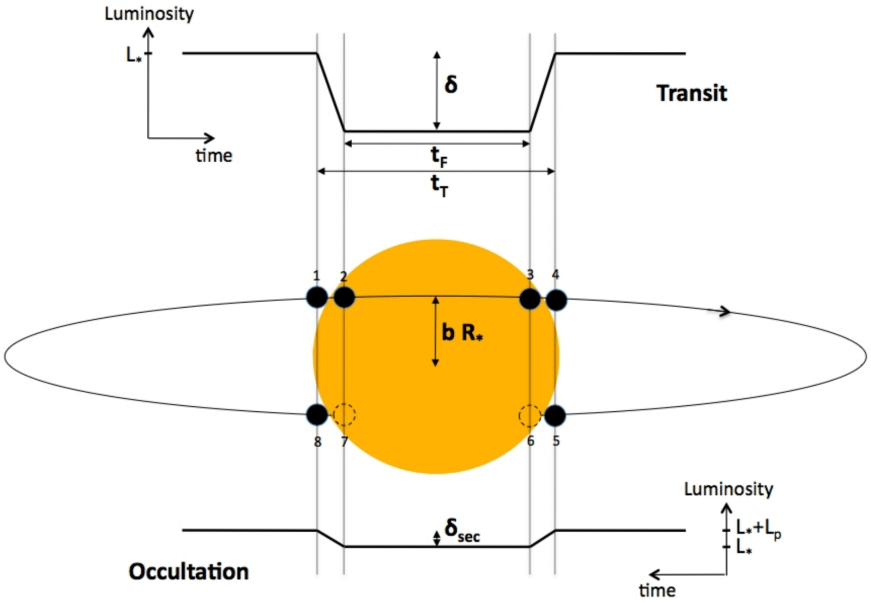


Figure 2.2: Schematic of a transiting planet and the resulting dip in apparent brightness [33]. The transit event consists of the total transit duration (t_T), full transit duration (t_F), and the transit depth (δ) normalized by the stellar luminosity (L_*). L_p represents the apparent brightness of the planet reflecting the star's light, and δ_{sec} is the relative depth of the occultation.

Using the transit depth (δ), we can estimate the planet-to-stellar ratio in a straightforward way

$$\frac{R_p}{R_*} = \sqrt{\delta}$$

Thus, deeper transits correspond to larger planets. Other parameters, such as the orbital semi-major axis (a) and distance from the star r, can be estimated using Kepler's third and first law, respectively, if the orbital period is known:

$$a^3 = \frac{GM_*P^2}{4\pi^2} \quad ; \quad r = \frac{a\,(1-e^2)}{1+e\cos v}$$

where P is the orbital period, G is the gravitational constant, M_* the mass of the star, $e = \sqrt{1-\left(\frac{b}{a}\right)^2}$ is the eccentricity of the orbit (how elongated it is), and the true anomaly v is the angle that describes the planet's position in its orbit at a given time. The orbital period can be also calculated directly from the photometric data if at least two transits are detected, but Seager et al. [41] shows that the period can also be calculated from a single transit event.

Knowing the distance r and estimating the star's temperature through blackbody radiation models, one can infer whether the planet is within its star's habitable zone, making it a candidate for hosting life. Another noteworthy quantity is the impact parameter b, or the distance between the center of the stellar disk and the projected path of the planet, defined in a range from 0 (transiting exactly at the center of the star) to 1 (transiting at the edge of the stellar disk), and it is directly correlated to the transit duration and the inclination angle of the planet's orbit.

An important aspect to take into account when characterizing transits is stellar limb darkening, which is the phenomenon where the center of a star appears brighter than its edges, as shown in Figure 2.3. This occurs because photons originating from the star's center emerge from deeper, hotter layers of the stellar atmosphere. Interestingly, a more pronounced limb darkening has been linked to a weaker stellar magnetic field [22]. The modeling of limb darkening is essential for accurate measurements of planetary radii and the impact parameter, as incorrect assumptions about the star's brightness distribution can distort the derived planet parameters.

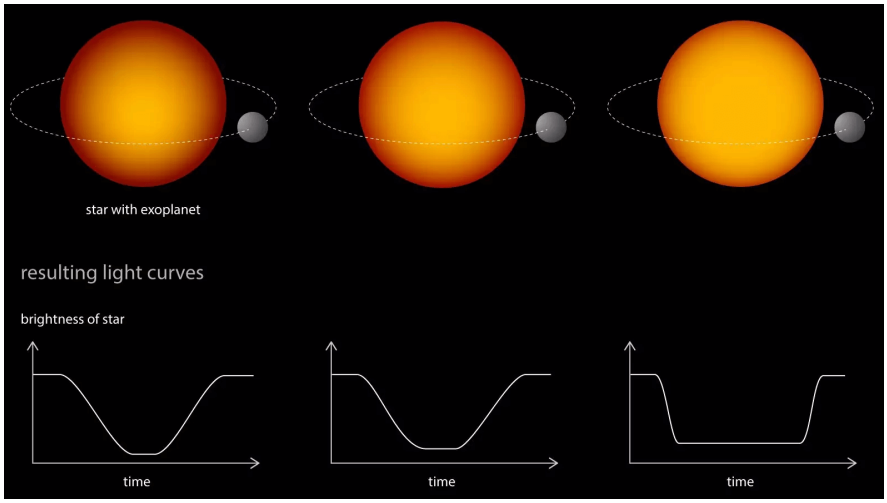


Figure 2.3: The effect of different levels of limb darkening to the shape of a planetary transit. Stronger limb darkening will result in a more gentle transition to the maximum depth, as opposed to a weaker limb darkening. [22]

2.2 Light Curve Data From Space Missions

Space telescopes such as Kepler and TESS collect data using the method of photometry, which measures the photometric flux emitted by a star over time. Photometric flux refers to the portion of the electromagnetic energy in the visible range emitted by a source (the star) and received by the observer per unit time. For simplicity, I will use the

term "flux" to refer to photometric flux. Both Kepler and TESS use Charge-Coupled Device (CCD) sensors to capture light from stars across a wide field of view. In the case of TESS, its four onboard cameras continuously take images with 2-second exposures. These images, known as Full Frame Images (FFIs), are transmitted to Earth, where they are processed with various data pipelines to extract science data from the raw frames. This processing includes segmenting the images to identify individual stars and measuring their brightness over time. The result is a time-series of brightness data, or light curve, which represents the star's flux as a function of time.

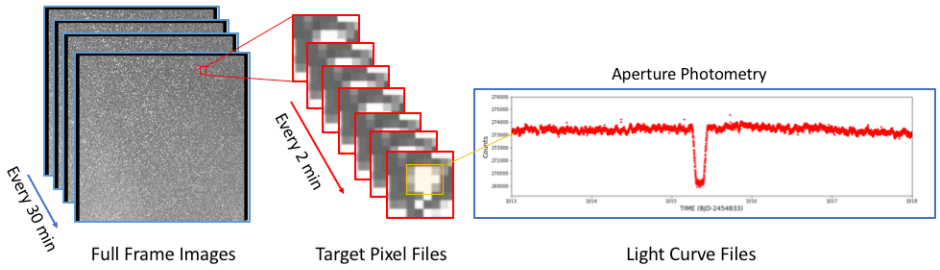


Figure 2.4: [21] Generation of light curve files from full frame images (FFI).

The Science Processing Operations Center (SPOC) is responsible for the processing of all the data received from the satellites, including the photometric and auxiliary science data (e.g. calibration data, instrument sensor data, stellar simulation) that are essential to the models used in the pipeline. The Compute Optimal Aperture algorithm identifies the pixels of interest for extracting photometric measurements from the CCD images for each target star in the SPOC pipeline. The identified pixels are then passed to the Photometric Analysis component for integration, to produce the final light curve of each target [21]. Additional steps involve data corrections for instrument biases, satellite drift, and other observational artifacts such as contamination by cosmic rays.

Figure 2.5: light curve data of star WASP-121, a F-type star of mass and radius slightly larger than our Sun ($1.35M_{\odot}$, $1.458R_{\odot}$). The periodic dips are transit events caused by the exoplanet WASP-121 b of mass $367.7M_{\oplus}$ (Earth masses) and radius $22R_{\oplus}$ (Earth radii). Such planet is considered a "Hot Jupiter" for its mass and high temperature due to the proximity to its host star.

Figure 2.5 shows the final processed and corrected light curve data for the star WASP-

121 and its transiting planet. The x-axis represents days, while the y-axis represents the measured flux normalized to one. The time axis is in units of TESS Barycentric Julian Day (btjd), a fractional representation of calendar day which is not affected by leap seconds, and corrected for relativistic effects due to the gravitational potential of Earth [32]. One notable feature in the light curve plot is the presence of a large gap around the time 2215. This gap is caused by the telescope's scheduled data processing and downlink operations, during which new data collection is paused, the telescope's antenna is oriented towards Earth and the stored data is transmitted to the ground laboratories.

2.3 Stellar Variability and False Positives

Stellar variability refers to changes in a star's brightness over time, which can complicate the interpretation of photometric data and lead to false-positive exoplanet detections. Variable stars are broadly classified into two categories: intrinsic and extrinsic variables. Intrinsic variability arises from internal physical processes, such as pulsations, thermonuclear explosions, or irregular changes in luminosity. Extrinsic variability, on the other hand, is caused by external factors, such as eclipses, starspots, or flares. Both types of variability can mimic the signals of exoplanet transits, making it challenging to distinguish between genuine planetary signals and stellar activity.

Intrinsic variables include pulsating stars like Cepheids and RR Lyrae stars, which exhibit periodic brightness changes, as well as irregular variables and cataclysmic variables, which undergo dramatic outbursts due to thermonuclear processes. Extrinsic variables include eclipsing binary systems, where periodic eclipses cause dips in brightness, and phenomena like starspots and flares, which result from magnetic activity on the stellar surface. These effects can produce signals in light curves that resemble planetary transits, especially when multiple sources of variability are present simultaneously.

Accurately detecting exoplanet transits requires a thorough understanding of how both intrinsic and extrinsic processes influence a star's apparent luminosity. Unfortunately, the complexity of these phenomena presents a significant challenge for our current methods of analysis, especially when multiple sources of variability are present simultaneously. For this reason, expert human scrutiny remains an essential step in validating and confirming detections produced by automated data analysis pipelines.

2.4 Statistical Analysis Tools

The traditional method for confirming a transit candidate as a planet involves a resource-intensive, multi-instrument follow-up campaign, including multicolor photometry, spectroscopy, and precise radial velocity measurements. However, the rapid increase in the number of transit candidates discovered by missions like Kepler has made this approach impractical, especially for faint targets that are challenging to observe with ground-based telescopes. As a result, probabilistic validation has emerged as a new paradigm for confirming transiting planets.

MCMC-Based Techniques: One of the most widely used methods for probabilistic validation is based on Markov Chain Monte Carlo (MCMC), a statistical technique that uses random sampling to approximate complex probability distributions and infer model parameters. MCMC constructs a chain of samples, where each subsequent sample is drawn from a probability distribution conditioned on the previous one. Over many iterations, the chain converges to an accurate estimation of the parameter distributions. In practice, an upper limit of iterations is set, and convergence is checked before accepting the results.

MCMC explores the posterior probability distribution $P(\theta|\text{Data})$ of the model parameters θ given the data. While analytical solutions to this problem are often infeasible due to the computational complexity of integrating over all possible parameter values (the "curse of dimensionality"), MCMC efficiently samples from the distribution without requiring a full exploration of the parameter space.

Several publicly available tools, such as `Pylightcurve`* [46], use MCMC routines to fit transit light curves. These tools enable simultaneous trend removal and provide a robust framework for probabilistic validation of exoplanet candidates.

Another tool is `exoplanet` [10], a toolkit built for modeling both transit and radial velocity observations. It is particularly designed with multiplanet systems in mind, making it a powerful resource for studying complex planetary systems. Like `Pylightcurve`, `exoplanet` uses MCMC for transit curve fitting and includes multiple limb-darkening models, although it does not offer detrending functionality.

Lomb-Scargle Periodograms: In addition to MCMC-based methods, periodic signal detection is another essential tool in analyzing light curves. However, due to the segmented nature of light curve data, traditional Fourier transform-based methods are ineffective. Instead, the Lomb-Scargle periodogram is used as an approximation to detect periodic signals in unevenly spaced data. The Lomb-Scargle method operates similarly

*https://github.com/ucl-exoplanets/pylightcurve

to the Fourier transform but is better suited for irregularly sampled datasets [48], under the assumption that the signals are of sinusoidal type and the noise component is Gaussian.

Gaussian Processes: Other powerful tools for light curve analysis are those based on Gaussian processes, which are particularly effective at modeling stellar variability. Gaussian processes are a family of models that can be used to fit and remove dominant trends in a light curve due to both periodic and non-periodic variability, such as star spots or flares, if the appropriate kernel is used. However, filtering out variability on time-scales similar to the planet's orbital period may also inadvertently remove the transit signal. By carefully detrending the light curve, Gaussian processes can help enhance the detection of smaller, fainter transits that might otherwise be obscured by stellar activity.

Hippke et al. developed the `Wōtan` [17] Python library, an open-source framework designed for detrending stellar light curves. It supports several detrending techniques, such as polynomial fitting, sliding window filters, splines, and Gaussian processes. These methods allow users to remove long-term trends or stellar variability effects while retaining potential transit signals. The most successful method used by `Wōtan` is a time-windowed slider combined with Tukey's biweight estimator [47]. This approach, highlighted for its effectiveness in detecting shallow exoplanet transits, mitigates noise by iteratively estimating the robust location of data points within a defined window, reducing the influence of outliers without overfitting the transit signal.

3. Data Acquisition and Exploration

3.1 Datasets

In this section, I describe the datasets utilized in the analysis of exoplanets and stellar properties. The same datasets are also used for the creation of the training dataset for the neural networks explored in this thesis, as explained in the Section 5.2. All datasets are publicly available and sourced from reputable scientific archives, ensuring their reliability and widespread use in the astrophysical community.

NASA Exoplanet Archive

The NASA Exoplanet Archive is a comprehensive and continuously updated catalog of all confirmed exoplanets detected through various observational methods and facilities [31]. It is maintained by the NASA Exoplanet Science Institute at IPAC, which is operated by the California Institute of Technology (CalTech). In the Planetary Systems table (PS), each row represents a self-contained set of parameters corresponding to a known exoplanet, its host star, and the system as a whole, for example: planet radius, mass, orbital period, orbital semi-major axis, time of first detected transit; and other stellar and system-level attributes. In addition, for each physical parameter, there is an optional upper and lower confidence range. It is important to note that there may be multiple entries for the same exoplanet, as different studies often derive different parameter values from each other. Each row references the publication detailing how the parameters in the row were derived, allowing users to trace the source of the data for reliability and cross-verification. For each planet, there is also a `default` flag indicating which of the multiple entries is considered the most complete or reliable.

TESS Objects of Interest

In addition to confirmed exoplanets, the NASA Exoplanet Archive also provides a list of candidate exoplanets detected by the TESS telescope, known as TESS Objects of Interest Table (TOI) [14]. A "candidate" is an object flagged as a potential planet, but not yet confirmed, so further studies or observations are required. This table includes a subset of estimated parameters and auxiliary fields to the Planetary System Table, but not as complete.

TESS Input Catalog

The TESS Input Catalog (TIC), available from the Mikulski Archive for Space Telescopes (MAST), is a comprehensive database of stellar parameters based on the data gathered from the Gaia mission Data Release 2 (DR2) [43]. It includes over 1.73 billion sources, and it is used for selecting observation targets for the TESS mission's two-minute cadence and for assessing the characteristics of transit candidates. It is maintained by the Smithsonian Astrophysical Observatory (SAO) as part of the TESS Science Office's responsibilities.

The dataset includes information about each star, including radius, mass, magnitude, celestial coordinates, and other physical parameters. Most importantly, it contains an identifier "`GAIA`" that can be used to cross-match each entry with its respective entry in the Gaia Data Release dataset, containing the most accurate estimations of stellar parameters to date.

The TIC was designed to include not only stars likely to be TESS targets, but also a vast number of fainter stars. This is useful for mitigating two common issues that arise during transit detection: blended eclipsing binaries and flux contamination. A faint eclipsing binary near a brighter target can create a photometric signal resembling a planetary transit [34]. Similarly, light from nearby stars can contaminate the target star's aperture, diluting the observed transit and affecting the apparent size of the exoplanet. By cataloging a broad range of stars, including these faint objects, the TIC helps account for these potential sources of error.

GAIA Data Release 3

The GAIA mission, launched by the European Space Agency (ESA) in 2013, aims to map the positions, distances, and motions of stars with unprecedented precision. With the latest data release in June 2022, GAIA Data Release 3 (DR3) provides astrometric

data for approximately 1.8 billion stars, down to a magnitude of $G \approx 21$ [11].

GAIA DR3 includes not only astrometric data but also astrophysical parameters for 470 million objects, including effective temperature, surface gravity, metallicity, distances, radius and mass.

The TESS Input Catalog is based on GAIA DR2 from 2018, so incorporating data from GAIA DR3 increases the precision and accuracy of the stars data.

TESS Observations

TESS produces a substantial volume of light curve data, which is conveniently organized in the MAST Archive. This archive allows users to search for specific Light Curve Files by their identifier and retrieve the corresponding data product's URL. Each entry in the database represents a unique data product, indexed by its Observation Identifier (`OBS_ID`) and linked to the star's `TIC_ID`, facilitating cross-referencing with other datasets. The database also provides the observation's start and end times, the TESS sector where the data were collected, the release date, and the URL for downloading the Flexible Image Transport System (FITS) files containing the light curves.

The light curve data consist of a FITS file, containing flux measurements and associated uncertainties for each timestamp, with missing data represented as `NaN`s. Systematic effects in the flux data are addressed by the SPOC algorithms, which correct instrumental and environmental effects. The final Light Curve File for a target contains the photometrically extracted light curve as well as corrected data after systematics removal. Each file corresponds to observations from a single TESS sector, meaning that targets observed over multiple sectors will have multiple Light Curve Files.

3.2 Data Collection and Preprocessing

General Data Collection Pipeline

I developed an automated Python pipeline for downloading, cleaning and aggregating data from the various astronomical sources listed in the previous section. At the core of this system are two widely adopted protocols in the astronomical community: the Table Access Protocol (TAP) and CasJobs.

The **Table Access Protocol** is a standardized service designed for the downloading

of tabular data, including astronomical catalogs [8]. TAP not only gives access to table data but also retrieves associated metadata that describes the structure and their content. The protocol is versatile, supporting multiple query languages such as SQL, and it accommodates both synchronous and asynchronous query execution.

CasJobs[*] is an online platform tailored for executing large-scale scientific queries on astronomical catalogs. CasJobs is particularly well-suited for handling complex SQL queries in batch mode, ensuring that long-running queries do not interfere with smaller, real-time operations. The platform offers both synchronous ('quick') and asynchronous ('long') job execution options.

The downloading process follows these steps for all the datasets:

1. **Authentication:** The pipeline initiates by authenticating with the dataset server, handled via the `astroquery` [13] Python library. Authentication is required especially for downloading the TIC and Gaia dataset, given their large size.

2. **Data Retrieval:** After successful authentication, a SQL query is submitted to the respective service to retrieve both the requested data and the accompanying field metadata. CasJobs is employed for querying the TIC dataset, while the TAP service is used for all the other datasets.

3. **Unit Verification:** The field metadata downloaded from the TAP service includes units for all numerical parameters. It is important to track these units to correctly perform calculations with physical quantities. A convenient way to store such data is in an `astropy.QTable`, a specialized data structure for tabular data where each column has an associated `astropy.unit` attribute representing the physical unit. Care must be taken when parsing the textual field metadata, which contains the units, and converting them to `astropy.unit` instances. In some cases, the units provided in the metadata may not directly correspond to the expected `astropy` units, requiring custom mappings to ensure all units are assigned correctly. Additionally, for consistency, similar fields are converted to the same units; for example, all time units initially expressed in various time formats are converted to days in Julian Date (jd) format.

4. **ID Parsing:** Identifiers that are stored as strings are converted into numerical formats. All the identifiers present in various datasets are renamed for clarity and easier cross-referencing. For example, in the Gaia dataset, the identifier is labeled as `source_id`, while in the Planetary System dataset, the equivalent field is named `gaia`. Both are standardized to `gaia_id` to maintain consistency.

[*] https://skyserver.sdss.org/CasJobs/

5. **Serialization:** Finally, the processed raw data is serialized in the `.feather` format, which optimizes read and write performance while maintaining a compact file size. Metadata is stored separately in a human-readable `.json` format. The Feather format is highly efficient for storing tables and dataframes, offering rapid input/output performance and reduced storage overhead.

Download Sequence and Cross-Matching

The datasets are downloaded in sequential order, repeating similar set of operations per each dataset as described before. All the datasets were collected on May 25^{th} 2024. The reported number of entries are all relative to this day.

1. Initially, the PS table from the NASA Exoplanet Archive is downloaded using the TAP service. The dataset contains 35299 rows, corresponding to 5632 unique planets orbiting 3967 stars.

2. In the same way, the TOI table is downloaded using TAP, counting 7147 candidate exoplanets orbiting 6877 stars.

3. TIC is accessed via the CasJobs API, fetching all the stars with mass within 0.7 and 1.3 solar masses ($M_{\odot}$). The returned dataset contains 324846 entries. Together with their `tic_id` and `gaia_id`, the query also returns the `priority` field, a float value assigned by the TESS team to loosely indicate the observation priority over other targets. Stars included in the PS and TOI databases are also fetched, ensuring completeness when cross-matching with the Gaia dataset.

4. The Gaia DR3 database is queried via TAP service to obtain key astrophysical parameters for all stars present in the PS, TOI, and TIC datasets. For this, the `gaia_id` field is used to query only the stars of interest, among the billions of entries in the full Gaia dataset. Targets labelled as binary stars, or potentially binary, are excluded to avoid the risk of obtaining data containing false positive transits. In total, the number of stars in the downloaded dataset is 295988.

5. Finally, the TESS Observation dataset is queried to fetch all the 2-minute cadence light curves metadata, corrected for systematic errors and collected for the stars having a `tic_id` in the PS, TOI and TIC datasets. Not all the stars from the TIC dataset have been observed, so only 132639 stars out of the potential 335690 were collected (3967 from PS, 6877 from TOI, 324846 from TIC), or about 40%. However, the total number of observation metadata collected are 425888, which is enough for creating a curated training set of a few tens of thousands entries.

Post-Processing and Data Imputation

Once all datasets have been collected, a final stage of post-processing is performed to clean and integrate the data into a unified `StarSystems` dataset. One key step involves the creation of a transiting-only exoplanet dataset, where missing stellar parameters, such as stellar radius, are imputed using data from Gaia DR3.

In addition, this reduced dataset is refined by consolidating multiple rows representing the same planet into a single entry. Since different studies may report varying parameter values for the same exoplanet, the most up-to-date and reliable values are selected. Any planets with incomplete or invalid data for key parameters such as period, radius, or orbital characteristics are flagged as "invalid" to ensure they are excluded from subsequent analyses. In total, it counts 4217 transiting exoplanets orbiting 3161 stars.

3.3 Exploratory Data Analysis

Planetary Parameters

Figure 3.1 presents histograms of the distribution of six key planetary parameters from the Planetary Systems dataset. These histograms highlight distinct trends in the population of detected exoplanets, with several parameters showing bimodal distributions.

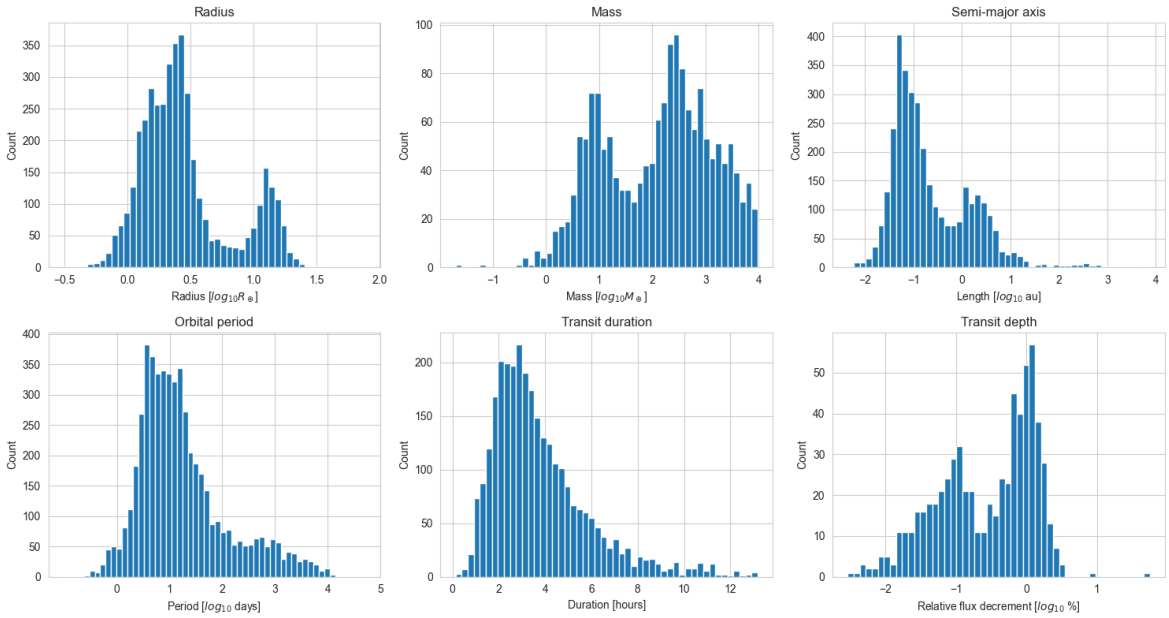


Figure 3.1: Distribution of six selected planetary parameters from Planetary Systems table.

The distribution of planetary radii reveals two prominent modes, one around $2.5R_\oplus$

and another near $13R_{\oplus}$. The first peak corresponds to Mini-Neptunes, while the second represents Jupiter-like gas giants. Similarly, the planetary mass distribution also exhibits a bimodal pattern, with peaks around $10M_{\oplus}$ and $300M_{\oplus}$, corresponding to the mass ranges spanning from super-Earths to gas giants, respectively, as seen on the logarithmic axis.

The semimajor axis, interpreted as the distance between the planet and its host star under the assumption of approximately circular orbits, reveals that most planets orbit very close to their stars, with a strong peak at 0.1 AU. This is expected due to the ease of detecting planets with short orbital distances, as they transit their stars more frequently. A secondary mode around 2.25 AU highlights the detection of planets further from their stars, though these are less common.

The orbital period histogram shows that the majority of planets exhibit periods between 1 and 100 days, and a peak concentrated between 3 and 15 days. This is likely a consequence of the observational bias towards shorter-period planets, whose frequent transits increase the signal-to-noise ratio on individual observations, making them easier to detect.

In terms of transit duration, most transits last between 2 and 4 hours, with a long tail of longer durations. Shorter transits are typical for planets that orbit close to their stars, as they have less distance to traverse during a transit. In contrast, planets further from their stars, with longer orbits, tend to exhibit longer transit durations as they cover larger distances across the stellar disk.

Lastly, the distribution of transit depths indicates that most planets cause a flux decrement around 1%, with a secondary peak at 0.1%. This shows that the majority of detected planets cause shallow transits, with less than 3.5% of the stellar flux being obscured during transit.

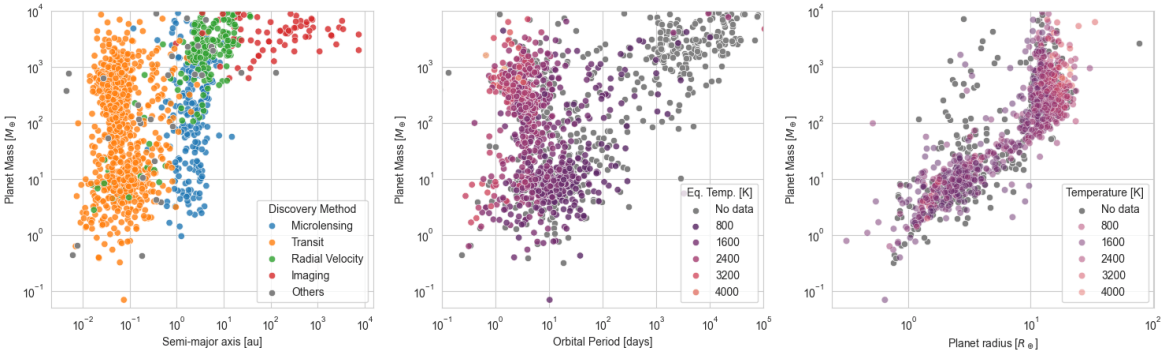


Figure 3.2: Relation of different planet properties

Figure 3.2 presents three scatter plots, each depicting the relationship between different

pairs of planetary parameters on logarithmic scales, with mass and radius given in Earth-relative units. These plots provide insights into how planetary characteristics relate to each other and how different detection methods exhibit biases toward certain planet types.

The leftmost plot shows the relation between the semi-major axis and planet mass, with each data point colored according to the method used to discover the planet. This plot highlights the observational biases of various detection techniques. For instance, the transit method (orange points) has predominantly detected planets within 1 AU, as it is most effective for planets orbiting close to their stars. Beyond 1 AU, radial velocity and microlensing methods take over, detecting planets at distances of 1 to 10 AU. Radial velocity is the preferred method for detecting more massive planets, while microlensing is more suitable for less massive ones, below $100M_{\oplus}$. For even farther and more massive planets, imaging methods become viable, as telescopes are able to mask the star to reveal the fainter light reflected by the planet orbiting far away.

The central plot illustrates the relationship between orbital period and planet mass, where the color gradient represents the planet's equilibrium temperature, with darker hues indicating cooler temperatures. Equilibrium temperature is an indication of the planet's temperature, calculated from the incident stellar energy of the host star. Planets with shorter orbital periods are hotter, as expected, due to their proximity to the star. This plot mirrors the trends seen in the semi-major axis vs. mass plot, as orbital period can be seen as a rough approximation for the distance between the planet and its star.

The rightmost plot presents the relationship between planet radius and mass, with color once again representing the planet equilibrium temperature. The nearly linear trend in the logarithmic scales suggests a power-law relationship between mass and radius, a feature consistent with planetary formation theories. At the upper-right end of the plot there are the gas giants, where mass increases disproportionately compared to radius due to gas compression at high planetary masses. The plot also reveals distinct populations of planets. Smaller, possibly rocky planets (with radii below $4R_{\oplus}$ and masses under $10M_{\oplus}$) form the first population; a second population comprises gas giants, with radii above $8R_{\oplus}$ and masses exceeding $100M_{\oplus}$. The intermediate range likely corresponds to Mini-Neptunes and ice giants. The temperature gradient provides further insight: there's a slight prevalence of Hot-Jupiters, since they are easier to detect, but hot planets seem to occur in every group. Cold planets are distributed in all groups as well.

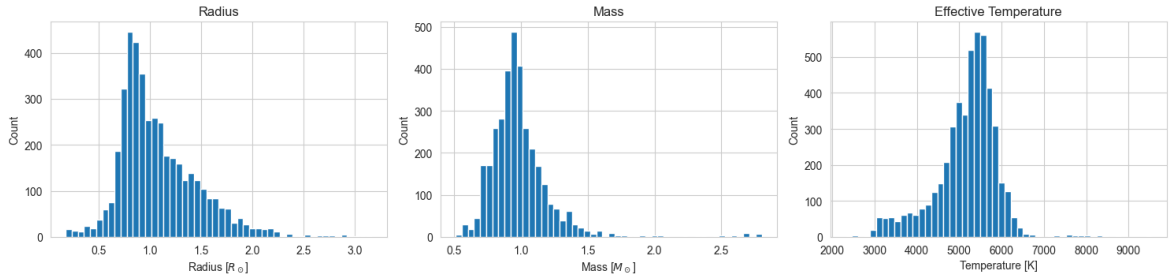


Figure 3.3: Distribution of three selected stellar parameters from Planetary System augmented with Gaia DR3 data.

Stellar Parameters

Figure 3.3 presents histograms showing the distribution of three selected stellar parameters: radius, mass, and effective temperature, for the stars hosting exoplanets in the Planetary System table, cross-matched with Gaia DR3 data. The effective temperature is a measure that approximates the surface temperature. The matched dataset contains a total of 4901 stars.

The radius and mass histograms reveal a sharp peak at $1R_\odot$ and $1M_\odot$, respectively, indicating that the majority of these stars are Sun-like in terms of size and mass. Similarly, the effective temperature distribution shows a peak around 5500 K, which is close to the Sun's effective temperature of 5772 K. This is not a coincidence, since Kepler's primary focus were Sun-like stars; it was only since the beginning of the second mission K2 and with TESS that a wider range of star types were added as targets.

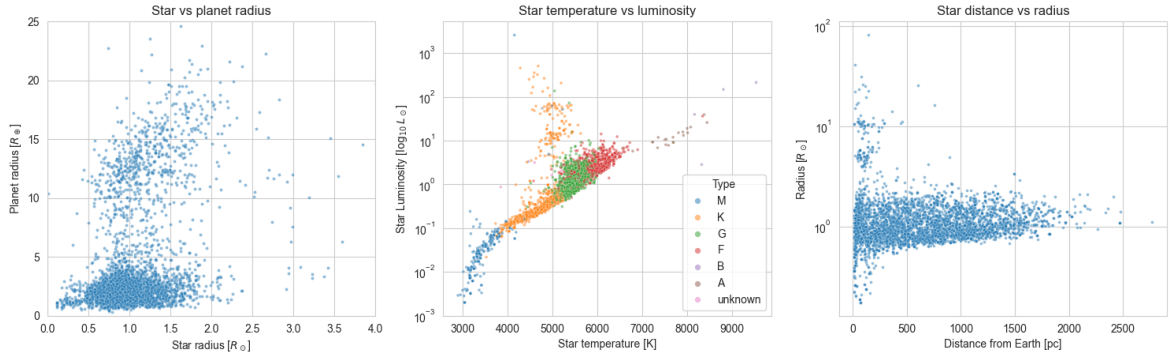


Figure 3.4: Relation of few selected stellar and planetary parameters

Figure 3.4 illustrates the relationship between various stellar parameters and planetary properties using three scatter plots.

The first subplot depicts the relation between planet radius and star radius. As seen in Figure 3.3, there is a noticeable overdensity around stars with a radius similar to the Sun ($1R_\odot$) and planets with a radius below $5R_\oplus$. This observation aligns with

the prevalence of Sun-like stars in the dataset. There is also a second region of lower density, which includes stars with a radius similar to the Sun, or slightly larger, but hosting planets with a larger radius, between 10 and 20 $R_{\oplus}$, which likely corresponds to gas giants and large planets in the dataset.

The central scatter plot is similar to the famous Hertzsprung-Russell (HR) diagram, showing the relationship between stellar temperature and luminosity. In this diagram, the stars are color-coded by their spectral type, ranging from M-type to B-type stars. As expected, the majority of the stars fall into the K, G, and F spectral types, which are similar to the Sun. These stars exhibit a wide range of luminosities and temperatures, with the majority ranging from 4500 to 6500 K.

The third subplot shows the relationship between stellar distance from Earth (in parsecs) and star radius. TESS's focus on nearby stars is evident from the overdensity of stars located within 100 parsecs. The majority of these stars have radii between 0.5 and 2 $R_{\odot}$, further emphasizing the focus on stars similar to the Sun. However, the distribution extends up to distances of around 2000 parsecs. Beyond this range, fewer stars are detected, likely due to the sensitivity limits of the instrument.

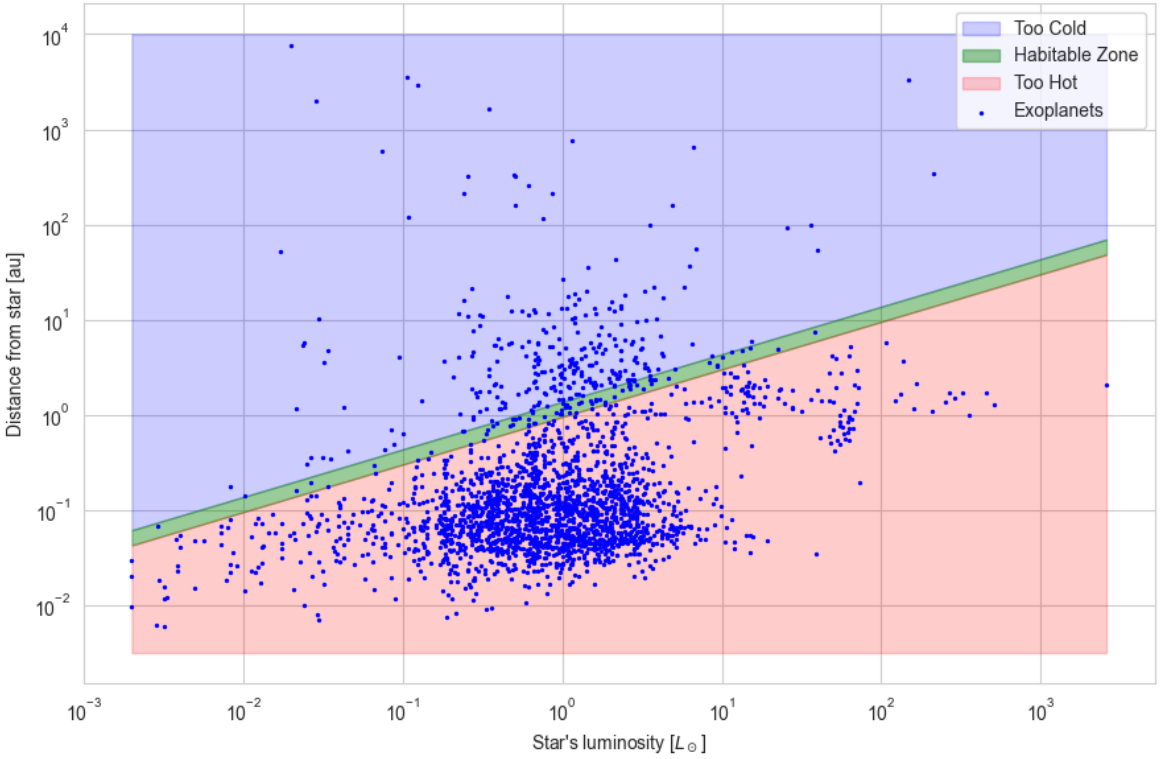


Figure 3.5: Exoplanets in relation with their habitable zone

Figure 3.5 displays a scatter plot illustrating the relationship between the absolute luminosity of stars and the orbital distances of their exoplanets. The green-shaded

region indicates the approximate location of the habitable zone, calculated based on the luminosity of each star, following the formulas provided by Whitmire et al. (1996) [50]. Both axes are log-scaled to aid the visualization.

The exoplanets are distributed across a broad range of distances from their host stars, with the majority located relatively close to their stars, predominantly within 1 AU. Planets situated below the green area reside in a region between the star and the inner boundary of the habitable zone, where conditions are likely too extreme (characterized by excessive heat or radiation) for liquid water to exist on the surface. Most exoplanets in the dataset occupy this inner region, which aligns with known observational biases that favor the detection of planets with shorter orbital periods.

Planets located above the green area are situated beyond the outer boundary of the habitable zone, where conditions are too cold for water to remain in liquid form. Only a small fraction of exoplanets are observed to lie within the green region, indicating that planets located within the habitable zone, and thus more likely of supporting life, are rare in this dataset. This scarcity suggests either an intrinsic rarity of such planets or the limitations of current detection methods.

4. Deep Learning

4.1 Deep Neural Networks

Deep Neural Networks (DNNs) are advanced computational models inspired by the structure and function of the human brain. They are built upon the foundational concept of the perceptron, a simple linear classifier proposed by Frank Rosenblatt in 1958 [38]. While the perceptron was limited to solving linearly separable problems, DNNs overcome this limitation by incorporating multiple layers of interconnected neurons and non-linear activation functions, enabling them to model complex, hierarchical patterns in data. This makes DNNs highly effective for solving sophisticated tasks in fields such as pattern recognition, regression, and classification.

At the core of DNNs is the Multilayer Perceptron (MLP), which consists of an input layer, one or more hidden layers, and an output layer. Neurons in each layer are fully connected to those in the subsequent layer. By introducing non-linear activation functions in hidden layers instead of a simple step function, the MLP is capable of solving non-linearly separable problems and modeling complex functions that a single-layer perceptron cannot handle.

One of the foundational results in the theory of neural networks is the Universal Approximation Theorem. This theorem states that a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a subset of $\mathbb{R}^n$, provided it uses a suitable activation function. This implies that neural networks, even with a single hidden layer, are powerful function approximators, capable of modeling arbitrarily complex functions. However, in practice, deeper networks with multiple hidden layers are often required for efficient learning and generalization.

Mathematically, an MLP can be described as follows. Given an input column vector $\mathbf{x} \in \mathbb{R}^{d_0 \times 1}$, the output of the k-th hidden layer is given by:

$$\mathbf{h}_k = \sigma(\mathbf{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k)$$

where:

- $\mathbf{h}_0 = \mathbf{x} \in \mathbb{R}^{d_0 \times 1}$ is the input column vector to the network,
- $\mathbf{h}_k \in \mathbb{R}^{d_k \times 1}$ is the activation column vector of the k-th layer,
- $\mathbf{W}_k \in \mathbb{R}^{d_k \times d_{k-1}}$ is the weight matrix for the k-th layer,
- $\mathbf{b}_k \in \mathbb{R}^{d_k \times 1}$ is the bias vector for the k-th layer,
- $\sigma(\cdot)$ is the activation function, applied element-wise.

The final layer outputs the predicted value, which can be a scalar in the case of regression or a vector of probabilities for classification.

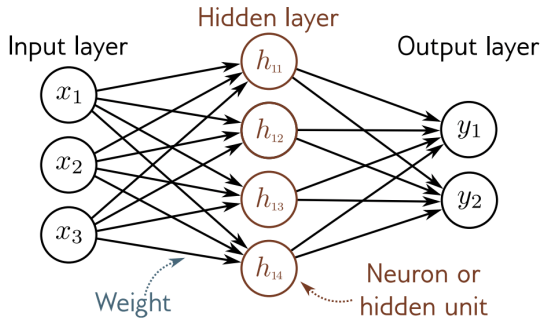


Figure 4.1: Visual representation of the multilayer Perceptron. The neurons in the hidden layers are indexed as h_{ij} where i refers to the layer number and j is the index of the neuron within the layer. [36]

The activation function $\sigma(\cdot)$ introduces non-linearity into the network. Without non-linear activations, the network would be equivalent to a single-layer linear model, regardless of the number of layers, and wouldn't be able to solve non-linearly separable problems. Commonly used activation functions include:

- **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$, which squashes the input to a value between 0 and 1. It is typically used in the output layer for binary classification problems.
- **Hyperbolic Tangent (tanh)**: $\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, which maps inputs to the range $(-1, 1)$.
- **Rectified Linear Unit (ReLU)**: $\sigma(z) = \max(0, z)$, which introduces sparsity and helps mitigate the vanishing gradient issue [18]. It has become the standard activation function in hidden layers of deep networks.

- **Leaky ReLU**: $\sigma(z) = \max(\alpha z, z)$, where α is a small constant. It allows a small, non-zero gradient when $z < 0$, solving the "dying ReLU" problem in which a network becomes hard to train due to a large number of its neurons permanently evaluating to 0.

Training a neural network involves optimizing the network's weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ to minimize a loss function, such as the mean squared error (MSE) for regression or cross-entropy for classification. The optimization is typically done using gradient-based methods, with gradient descent being the most commonly used technique.

In gradient descent, the network's weights are updated in the direction that reduces the loss, following the gradient of the loss function with respect to the weights. The update rule for the weights in layer k at iteration t is given by:

$$\mathbf{W}_k^{t+1} = \mathbf{W}_k^t - \eta \nabla_{\mathbf{W}_k} \mathcal{L}(\mathbf{W})$$

where:

- η is the learning rate, which controls the step size,
- $\nabla_{\mathbf{W}_k} \mathcal{L}(\mathbf{W})$ is the gradient of the chosen loss function $\mathcal{L}$ with respect to the weight matrix $\mathbf{W}_k$.

In practice, **Mini-Batch Stochastic Gradient Descent (SGD)** is often used, where the gradient is computed over a mini-batch of training examples rather than the entire dataset. This leads to noisy but computationally efficient updates:

$$\mathbf{W}_k^{t+1} = \mathbf{W}_k^t - \eta \nabla_{\mathbf{W}_k} \mathcal{L}(\mathbf{W}_k; \mathcal{B})$$

where $\mathcal{B}$ is a randomly sampled mini-batch of training examples. Mini-batch SGD allows the model to converge faster by performing frequent updates and provides implicit regularization by introducing noise into the updates.

To compute the gradients, the most commonly used algorithm is backpropagation [24]. It relies on the chain rule of calculus to propagate the errors backward from the output layer to the input layer, enabling the network to update the parameters of all layers. The key idea behind backpropagation is that the error at the output layer is used to adjust the weights of the preceding layers.

For a given loss function $\mathcal{L}$, the gradient of the loss with respect to the weight matrix $\mathbf{W}_k$ in the k-th layer is computed as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k} = \delta_k \mathbf{h}_{k-1}^\top$$

where: $\mathbf{h}_{k-1} \in \mathbb{R}^{d_{k-1} \times 1}$ is the output from the previous layer, and $\delta_k \in \mathbb{R}^{d_k \times 1}$ is the error term for layer k, computed recursively using the chain rule:

$$\delta_k = (\mathbf{W}_{k+1})^\top \delta_{k+1} \odot \sigma'(\mathbf{h}_k)$$

where $\sigma'(\mathbf{h}_k) \in \mathbb{R}^{d_k \times 1}$ is the element-wise derivative of the activation function for layer k, and $\odot$ denotes element-wise multiplication. By iterating this process for each layer, backpropagation computes the gradient for each neuron, allowing the network to update its weights and learn from the input data.

One significant challenge in backpropagation is the **vanishing/exploding gradient problem**, which occurs when gradients of the loss function become exceedingly small or large as they are propagated backward through many layers. This leads to extremely slow updates of the earlier layers' weights, causing the network to struggle in learning long-range dependencies. The problem is especially evident when using activation functions like the sigmoid or tanh, which tend to squash gradients in their concentrated regions.

4.2 Sequence Modeling

MLPs have been widely used for various classification and regression tasks, but they present limitations when applied to sequential data. For instance, they require a fixed input size, which complicates their application to variable-length sequences. Also, they treat the input as inherently non-sequential, disregarding any time or order dependency between points. To address these shortcomings, specialized deep learning architectures have been developed for modeling sequences of data.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models that excel at processing structured grid-like data, such as images, becoming the standard for tasks like image classification, object detection, image segmentation. Their success can be attributed to their capacity to capture local dependencies via convolutional layers, while simultaneously reducing the number of parameters through the implementation of two concepts: sparse connectivity and parameter sharing. Sparse connectivity refers to the

fact that each output of a convolution layer is connected to only a small region of the input, rather than the entire input. Parameter sharing implies that the same set of weights (the kernel) is applied across different regions of the input, reducing the number of learnable parameters significantly and, as a consequence, improving the efficiency of the network training and inference.

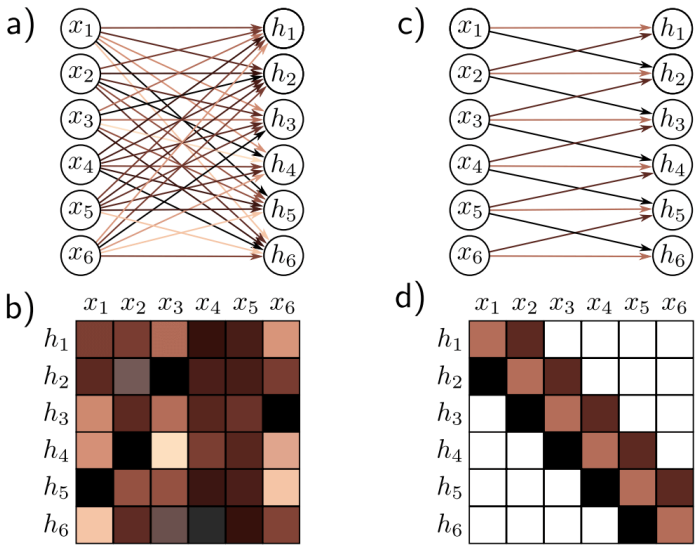


Figure 4.2: Fully connected (left) vs. convolutional layers (right). The top row shows the connections between neurons in each network. In this example, the fully connected network uses 36 different parameters, while the CNN shares the same 3, while keeping most of the weight matrix at 0. [36]

At the core of CNNs is the cross-correlation operation, often referred to as "convolution" in the deep learning literature. Given a 1D input sequence x and a kernel w of size k, the cross-correlation y_i is defined as:

$$y_i = \sum_{j=0}^{k} w_j \cdot x_{i+j-m}$$

where m is a non-negative number satisfying the relation $k = 2m + 1$, under the assumption that k is odd. This operation can be extended to multiple dimensions to process higher dimensional data such as gray-scale images (2D), colored images (3D) or videos (4D). The convolution operation can involve multiple kernels, each learning different features from the input. As the convolution is applied across the input image, each filter detects specific patterns, such as edges, textures, or more complex structures.

CNNs are characterized by key parameters: the number of filters (channels), kernel size (defining local receptive fields), padding (to control output dimensions), stride (step size of kernel movement), dilation (expanding receptive field without increasing kernel size), and pooling (downsampling via max or average operations).

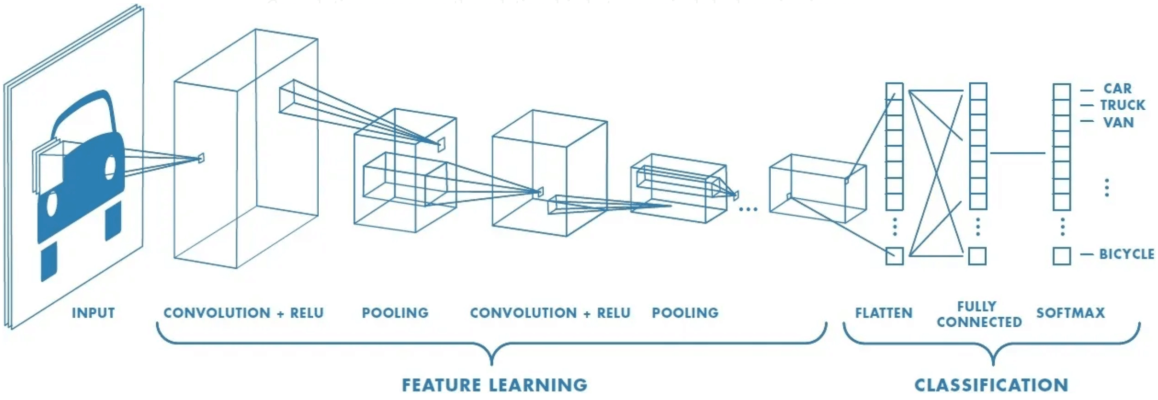


Figure 4.3: Typical architecture of a CNN for classification tasks. The input is passed through multiple convolutional layers, each using different kernels to extract features, including downsampling through striding and/or pooling. Finally, the output is flattened and classified using a fully connected network.[2]

Downsampling reduces computational complexity and enhances feature extraction by progressively condensing input data. This is achieved via strided convolutions, which move the kernel with a step size greater than one, or pooling layers, which summarize local neighborhoods.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) extend the standard feedforward neural network by introducing recurrent connections, which allow information from previous time steps to influence the current output, capturing non-local dependencies. They have been used in contexts such as timeseries forecasting, speech recognition, audio and natural language processing.

At each time step t, the hidden state h_t is updated based on the input x_t and the previous hidden state h_{t-1}:

$$h_t = \sigma(W_x x_t + W_h h_{t-1})$$

where W_h and W_x are the weight matrices, and σ is a non-linear activation function, such as tanh.

However, training RNNs can be challenging, particularly for long sequences, due to the issues of vanishing and exploding gradients, greatly limiting the size of inputs. To address these limitations, architectures like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) were developed [19]. LSTMs introduce a gating mechanism to regulate the flow of information and maintain long-term dependencies.

LSTMs consist of three primary gates that control the information added to, removed from, and read from the cell state. The input gate controls how much of the new information from x_t should be stored. The forget gate determines how much of the previous cell state should be retained, while the output gate dictates the portion of the cell state that will be output.

GRUs are a more streamlined version of LSTMs, combining the forget and input gates into a single update gate. The GRU architecture simplifies the network without losing significant performance, often providing similar results to LSTMs on various sequence tasks. For example, GRUs have been shown to perform comparably to LSTMs in time-series forecasting applications such as stock price prediction [12] and energy consumption modeling [27].

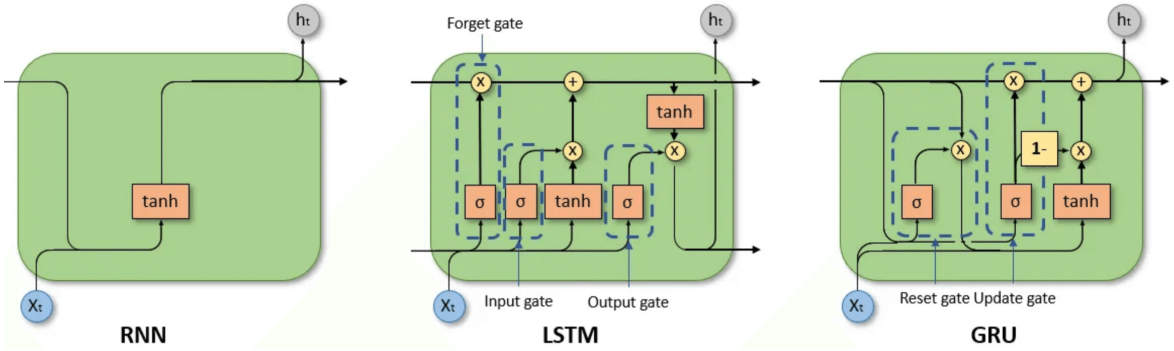


Figure 4.4: Architectures of RNN, LSTM and GRU networks, illustrating their respective gating mechanisms.

While standard RNNs, LSTMs, and GRUs process the input sequence in a single direction, Bidirectional RNNs (BiRNNs) extend this by processing the sequence both forwards and backwards. This allows the model to capture context from both past and future time steps, which can be particularly useful in tasks where the entire sequence is available beforehand [40].

Transformer Models

Transformer models have revolutionized sequence modeling tasks by addressing many of the limitations found in traditional RNN architectures. Unlike RNNs, which rely on sequential processing, Transformers use a self-attention mechanism that allows for parallelization, making them significantly more efficient at handling long sequences. Their ability to capture long-range dependencies without the need for recurrent connections has made them a popular choice in tasks like natural language processing, machine translation, and time-series prediction.

The core innovation of Transformer models is the self-attention mechanism, which computes the relationships between all pairs of inputs in a sequence, allowing the model to focus on important elements regardless of their position [49]. Given an input sequence X, the self-attention mechanism computes three matrices: queries $Q = XW^q$, keys $K = XW^k$, and values $V = XW^v$, where W^q, W^k, W^v are learnable weight matrices. The attention score is computed as the dot product of the query with all keys, normalized by a softmax function. This score is then used to compute a weighted sum of the values:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimension of the key vectors. The self-attention mechanism allows each element in the sequence to attend to every other element, making it effective for capturing global dependencies.

In tasks where the model must generate a sequence, such as language generation, causal attention is used to ensure that predictions at a given position depend only on previous positions in the sequence. This is achieved by masking the future positions during training, preventing the model from looking ahead. This causal structure allows Transformers to be used for autoregressive tasks, like text generation or forecasting.

An extension of the self-attention mechanism is multi-head attention, in which the input sequence is divided into multiple subsets, and attention is computed independently for each, in parallel. This allows the model to attend to different parts of the sequence simultaneously, capturing various types of relationships between elements. The output from each attention head is then concatenated and linearly transformed to form the final output.

Since the Transformer does not inherently process data in a sequential manner, it lacks the ability to track the order of the elements in a sequence. To address this, positional encoding is added to the input embeddings, providing the model with information about the relative positions of the tokens in the sequence. A common approach is to use sinusoidal functions to encode the positions, which are added to the input embeddings:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad \text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

These encodings allow the Transformer to maintain a notion of sequence order while processing all positions in parallel.

Another advantage of Transformer models over RNNs is their ability to perform parallel training. Since self-attention does not rely on the output from previous time steps, the entire sequence can be processed simultaneously, allowing for more efficient distributed

training on modern hardware like GPUs and TPUs. With multi-head attention, the computation can be accelerated even more by distributing the workload to a finer granularity, making Transformers highly scalable and faster to train, particularly on large datasets.

4.3 The Conformer Model in Astrophysics

Conformer models are an innovative architecture that combines the strengths of Convolutional Neural Networks and Transformers, addressing the limitations of each model. While Transformers excel in capturing long-range dependencies through self-attention mechanisms, they are less effective at extracting fine-grained local patterns. On the other hand, CNNs are suitable for learning local features, but capturing global context typically requires deeper architectures with more layers or parameters. The Conformer model combines these approaches, extracting both global and local information simultaneously via Multi-Head Self-Attention (MHSA) and convolutional modules within each Conformer block.

Originally introduced by Gulati et al. in 2020 for speech recognition tasks [15], the Conformer architecture has since been adapted for various other domains, including time series regression. In the field of astronomy, the AstroConformer model [35] is a successful application of this architecture for predicting the surface gravity of stars, represented as $\log g$, using light curves recorded by the Kepler Space Telescope. Each data point in the light curve is proportional to the number of photons detected by the instrument's sensor during a specific exposure period, referred to as cadence. Typically, 2-minute cadence is used. However, in the AstroConformer model, the data is resampled to a 30-minute cadence to reduce noise and improve the robustness of the predictions.

In the AstroConformer model, shown in Figure 4.5, the raw light curve is first divided into patches of 20 data points, corresponding to 10 hours of observation each. Each patch is embedded into a 128-dimensional representation using a fully connected layer. A sequence of 200 embedded patches is then fed into an 8-head MHSA block, which captures long-range dependencies. This is followed by normalization layers and convolutional layers, with a kernel size of 3, a stride of 1, padding of 1, and 128 channels, to extract local features. The final output, of size 200×128, undergoes average pooling before passing through a fully connected layer to predict a single value for $\log g$.

To incorporate positional information into the input sequence, the authors of AstroConformer employed Rotary Positional Encoding (RoPE) [44]. Positional encoding is necessary because the MHSA mechanism is agnostic to the sequence order of its

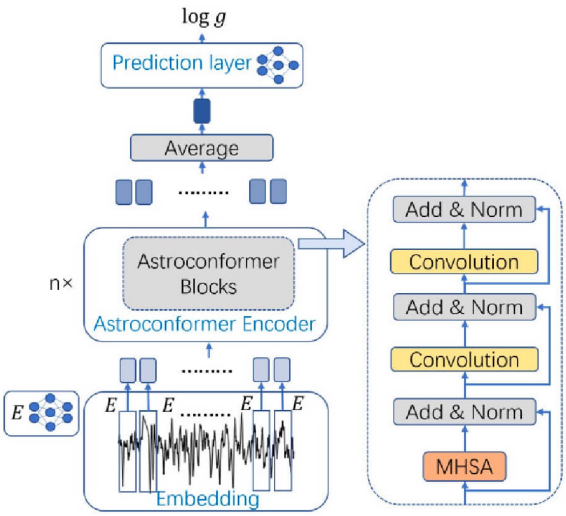


Figure 4.5: AstroConformer architecture. [15]

input, and thus lacks a built-in ability to model positional dependencies. Compared to basic sinusoidal positional encoding [49], which adds positional information to the input embeddings through additive sinusoidal functions of fixed frequencies, RoPE introduces positional information directly into the attention mechanism by applying rotational phase terms to the query and key vectors. Specifically, in RoPE, for a query at position m, the updated query vector q'_m is given by $q'_m = q_m e^{i\omega m}$, and for a key at position n, the updated key vector k'_n is given by $k'_n = k_n e^{i\omega n}$, where ω is a fixed frequency parameter.

The model was trained using a dataset of 14,000 normalized and scaled light curves of stars with known properties. The AdamW optimizer was employed with a cyclic learning rate scheduler [42]. The training consisted of 20,000 steps with a batch size of 256, optimizing for Root Mean Squared Error (RMSE) as the objective function. The model was trained for The model was 100,000 training steps for about 50 hours.

The performance of AstroConformer was evaluated against several baseline models, including the SWAN k-NN method [29] and two ResNet-based models. AstroConformer demonstrated superior generalization capabilities, producing fewer outlier predictions and achieving lower RMSE compared to these models. Its ability to effectively capture both short-term and long-term patterns in time series data suggests that this architecture could serve as a basis for other tasks beyond astrophysics.

5. Multitask Models for Exoplanet Detection

In this chapter, I present a discussion on the multi-task learning approach employed for the detection of transiting exoplanets in light curve timeseries data. The chapter begins with a definition of multi-task learning and proceeds to outline the objectives of the tasks of the proposed model. I then discuss the creation of the training dataset from the data retrieved as described in Section 3.2, followed by the description of each model considered in this work. Finally, I describe the training strategies and software architectural choices for the actual implementation. In this chapter, there are references to specific parts of the source code, which can be found at the GitLab repository https://version.helsinki.fi/ccardin/exoplanet-finder.

5.1 Multi-Task Learning Setup

5.1.1 Definition

The models developed in this thesis follow a multi-task learning (MTL) approach. In MTL, a model is designed to solve multiple tasks simultaneously, each task using the same underlying features extracted from the model's inner layers. This design can be implemented using a single encoder and a combination of multiple prediction blocks. An encoder processes the input data into a latent representation, while multiple decoders/blocks are tasked with specific objectives, such as classification, segmentation, regression, or other tasks. Each output block utilizes the shared latent representation as the starting point for its task-specific predictions, but the blocks are separated from each other.

A multi-task model with shared encoder architecture captures richer and more generalizable latent representations, as the encoder is encouraged to extract features that are beneficial for all tasks. Empirical studies [5] have shown that multi-task models

can outperform their single-task counterparts in various applications, such as image segmentation combined with depth estimation [26].

Each task within an MTL framework is associated with a specific loss function, which can be weighted according to the importance of the corresponding task.

5.1.2 Task Objectives

To characterize an exoplanet from a detected transit event, several parameters are typically required: the mid-time of any occurrence of the transit (usually the first, called transit midpoint), the period, the duration, the transit depth, and the limb darkening profile of the host star. However, it is possible to simplify the problem by focusing on only two key parameters: the transit midpoint and the period. Once these are known, a folding operation can be applied to the light curve timeseries to align the transits, effectively separating the signal from the noise. This process facilitates the estimation of other parameters using traditional methods, as seen in Figure 5.1.

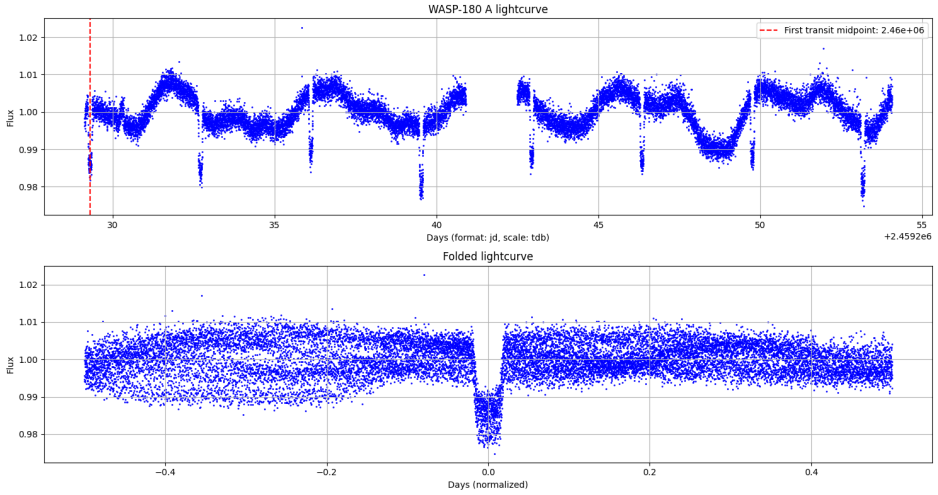


Figure 5.1: Star WASP-180 A and its companion planet "b". The planet's period is 3.41 days, and the midpoint of first transit is at 2458206.519399 Julian Days. With this information, we can isolate the transit and study its properties in the folded light curve, shown in the bottom plot centered on the transit midpoint. Some scatter is expected due to spot-induced rotational modulation of the star, which introduces variability unrelated to the planet's transit.

Mathematically, the folding operation is described by the following equation*:

$$\phi = \frac{t - t_{\text{mid}}}{P} - \left\lfloor \frac{t - t_{\text{mid}}}{P} \right\rfloor$$

*https://pyastronomy.readthedocs.io/en/latest/pyaslDoc/aslDoc/folding.html

where the phase ϕ is a number between 0 and 1, t is an array of time values, P is the period used for folding (in the same units as t), and t_{mid} is the transit midpoint, but can be any reference point in time. The Gaussian brackets $\lfloor \cdot \rfloor$ represent the floor operation.

The primary goal of this thesis is to design and implement a deep learning model capable of inferring the transit midpoint and period directly from a light curve. The high-level model architecture remains the same for all experiments and is depicted in Figure 5.2.

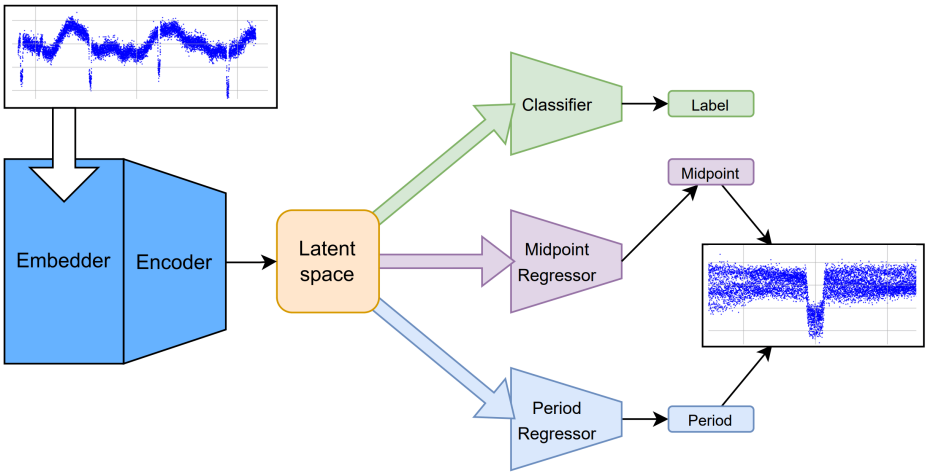


Figure 5.2: High level architecture of the proposed multitask models.

The input to the model is a 2D tensor of size $2 \times N$, where N is the number of observations in a light curve. The first dimension corresponds to the time of each observation t, and the second dimension corresponds to the measured flux y at time t. The model consists of the following key modules:

- **Embedder:** The first module processes the input light curve by embedding the flux data alongside the time component. This embedding module extracts preliminary features from the raw input.
- **Encoder:** The second module, the encoder, further processes the embedded light curve to produce a compressed latent representation. This latent representation serves as the shared basis for the subsequent tasks.
- **Output Blocks:** Three separate blocks utilize the shared latent representation to perform distinct tasks:
    1. **Multiclass Classifier:** Assigns one of three labels to the light curve, indicating the presence of no transits, a single transit, or multiple transits.
    2. **Midpoint Regressor:** Estimates the time t_{mid} of the first detected transit event in the light curve.

3. **Period Regressor:** Predicts the period of the transit event.

5.2 Training Dataset

5.2.1 Synthetic Transits Generation

The training data is created by injecting synthetic transit signals into real stellar light curves. The resulting light curve, along with the midpoint and period parameters that generated the synthetic signal, constitutes an entry in the training dataset.

This approach enables the model to be trained on a diverse range of transit events with known characteristics, perturbed with noise and features that well represent the processes undergoing in real stars. Figure 5.3 illustrates the process of injecting an artificial transit signal into the light curve of TIC 219854185, with varying degrees of intensity controlled by a parameter, k. This parameter modulates the level of perturbation, blending the clean transit signal with the original light curve flux. The perturbed flux is defined as:

$$\text{flux}_{\text{perturbed}} = \text{flux}_{\text{light curve}} \cdot k + \text{transit} + (1 - k) \tag{5.1}$$

where the light curve flux is normalized to 1, and the transits median is 0.

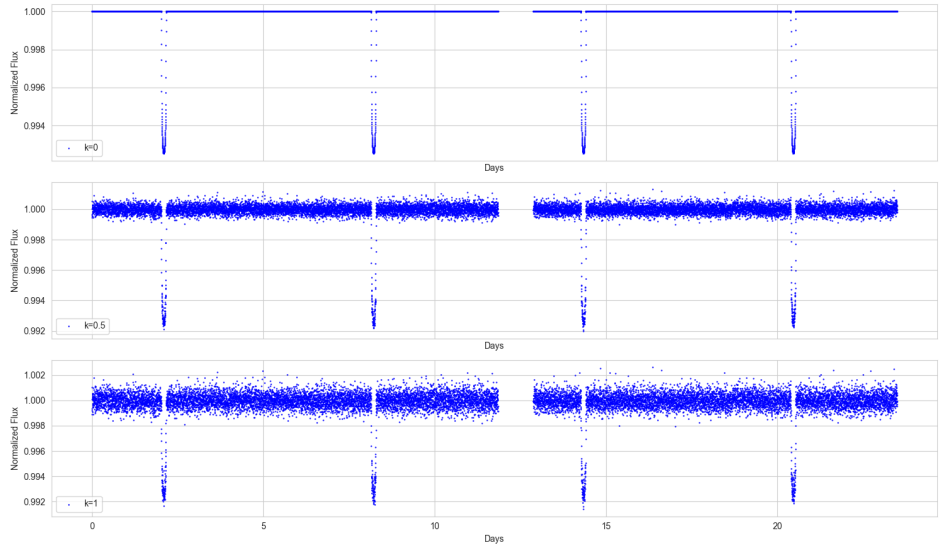


Figure 5.3: Randomly generated artificial transit signal injected into light curve "TIC 219854185" with varying intensity levels, k. For simplicity, the effect of the secondary transit (when the planet passes behind the star, blocking the light it reflects) is omitted.

The parameter k was useful in the development phase, as it gradually exposes the model

to increasing levels of perturbation in the transit signal. Initial model prototypes were developed using $k = 0$, representing a noiseless scenario, to ensure stable convergence. Once convergence was reliably achieved, I ran several experiments to test how varying k during training helped the model to produce better results; ultimately, I trained the final models with the full perturbation $k = 1$, as the tests resulted in erratic loss trends when training with varying values for k.

Synthetic transits were generated using the `batman` Python package [23], which simulates transit signals based on planetary and orbital parameters. This package implements a widely used set of analytical equations developed by Mandel and Agol [28]. Each planet was characterized by two primary parameters: its type and its distance from the host star.

- **Planet Type:** The planet type determines the ranges for radius (measured in Earth radii) and density (measured in g/cm^3) as reported in Table A.1. These quantities are then used to calculate the planet's mass and stellar-to-planet radius ratio from the randomly sampled radii and densities.
- **Distance from Star:** Specifies ranges for the planet's distance from its host star, measured in relative stellar radii (of the host star). This distance is used for calculating the semi-major axis of the orbit, which, combined with the masses of the star and the planet, determines the orbital period using Kepler's third law.

Using Kepler's third law, either the period or semi-major axis can be derived depending on which parameter is provided:

$$P = \sqrt{\frac{4\pi^2 a^3}{G(M_{\text{star}} + M_{\text{planet}})}} \quad \text{and} \quad a = \sqrt[3]{\frac{G(M_{\text{star}} + M_{\text{planet}})P^2}{2\pi^2}} \tag{5.2}$$

where G represents the gravitational constant, M_{star} is the mass of the host star, and M_{planet} is the mass of the planet. The first equation calculates the orbital period P given a known semi-major axis a, while the second equation determines the semi-major axis a when the orbital period P is provided.

Additional parameters, such as eccentricity and limb darkening coefficients, were sampled from distributions reflecting typical values observed in known planetary systems. See Figure 5.4 for an example of randomly generated transits per planet type and distance; for additional details about the generation process, consult the Appendix A.1.

For cases where only a single transit is required within the duration of a light curve, the midpoint of the transit is set randomly within the range of the light curve, and the

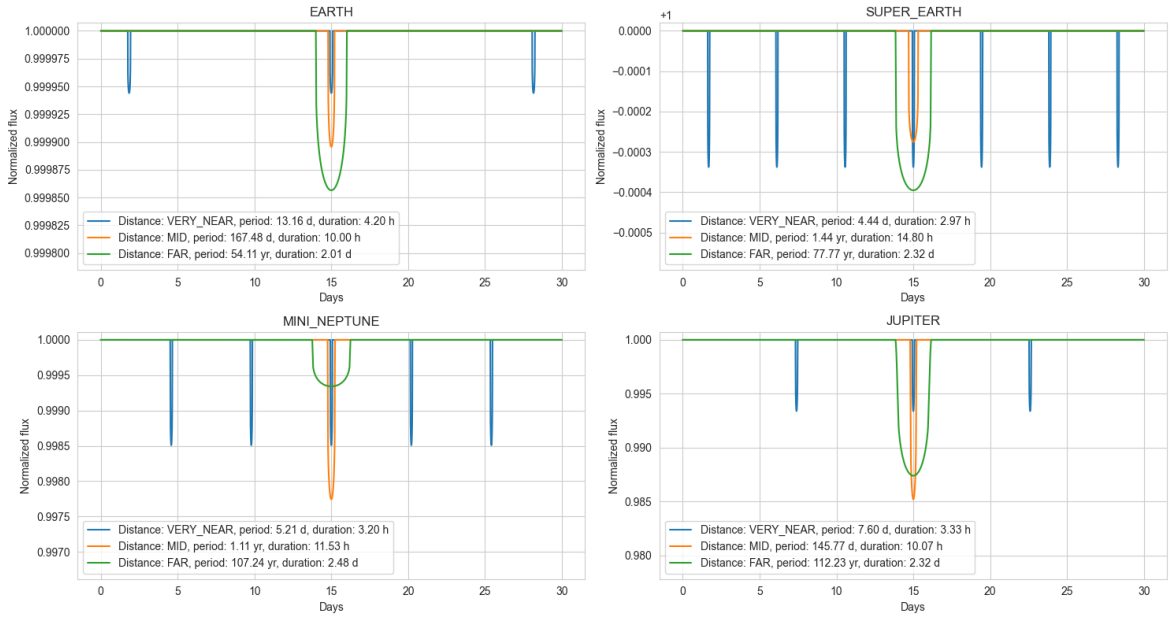


Figure 5.4: Random transit signals generated by planets of varying types (Earth, Super Earth, Mini-Neptune, Jupiter) and orbital distances (Very Near, Mid, Far) from a star with a fixed radius of $1R_{\odot}$. The parameters defining these distances are shown in Table A.3. Large planets produce deeper transits, as evident from the y scale, and transits from planets further away are broader (longer duration). Closer orbital proximity results in more frequent transits with shorter periods and reduced transit durations.

period is adjusted to ensure only one transit event appears. Alternatively, if the light curve is meant to display multiple transits, the midpoint is selected as the time of the first transit occurrence.

The transit depth is directly affected by the planet type, as shown in Figure 5.5. We notice that, for smaller planet types, the density concentrates on a smaller range of possible depths. For Earths and Super Earths types, it is particularly small, peaking around 0.01% and 0.03%. However, larger planet types are allowed to vary between a wider range of possible radii, showcasing a long tail towards deeper transit depths in the generated data.

5.2.2 Light curve Augmentation Set

In order to create the training set, I had to first select a subset of high-quality light curves to blend with the generated planet transits, from all the several hundreds of thousands available. Although the original dataset comprises 425,888 observations from 132,639 stars without confirmed planets (see Section 3.2 for details), the amount of data to process is too large for the computing resources I have available. To streamline the dataset to a manageable size, I applied a series of preliminary selection and filtering

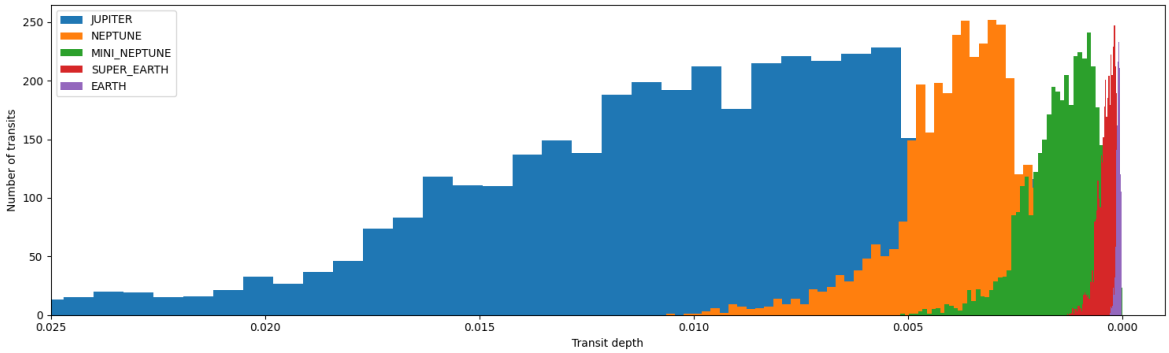


Figure 5.5: Distribution of transit depths for different types of planets.

criteria, reducing the total to around 14,000 high-quality light curves. The following sections outline the selection and filtering methods used to construct this augmentation set.

Preliminary selection

To ensure cleaner data, only light curves processed with the Pre-search Data Conditioning Simple Aperture Photometry pipeline (PDCSAP) will be considered. PDCSAP data is corrected for long-term trends in flux using Co-trending Basis Vectors, which effectively reduce systematic errors and enhance data quality*.

The preliminary selection process begins by excluding stars with known or candidate exoplanets, as the objective is to generate a synthetic dataset with only known synthetic transit events. Only the stars whose TIC IDs do not appear in any known or candidate exoplanet datasets are retained. Additionally, only the most recent 20% of observations are selected to minimize the impact of early instrument errors and pipeline flaws.

From this subset, I further narrow down the selection to 6,000 randomly sampled stars, yielding approximately 30,000 light curves.

Quality Filtering

After downloading 30,000 light curves, quality metrics were evaluated to ensure minimal missing data and preprocessing defects. The key metrics included flux statistics (median, min, and max values), total measurements, time gap ratio, median absolute deviation (MAD), and normalized standard deviation (STD).

Aggregated results were analyzed, and a subset was visually inspected. High MAD and STD values were linked to irregular or corrupted data patterns, while high time gap

*https://heasarc.gsfc.nasa.gov/docs/tess/lightcurveFile-Object-Tutorial.html

ratios posed risks of missing transit events. Figure 5.6 illustrates differences between light curves with extreme values for these metrics.

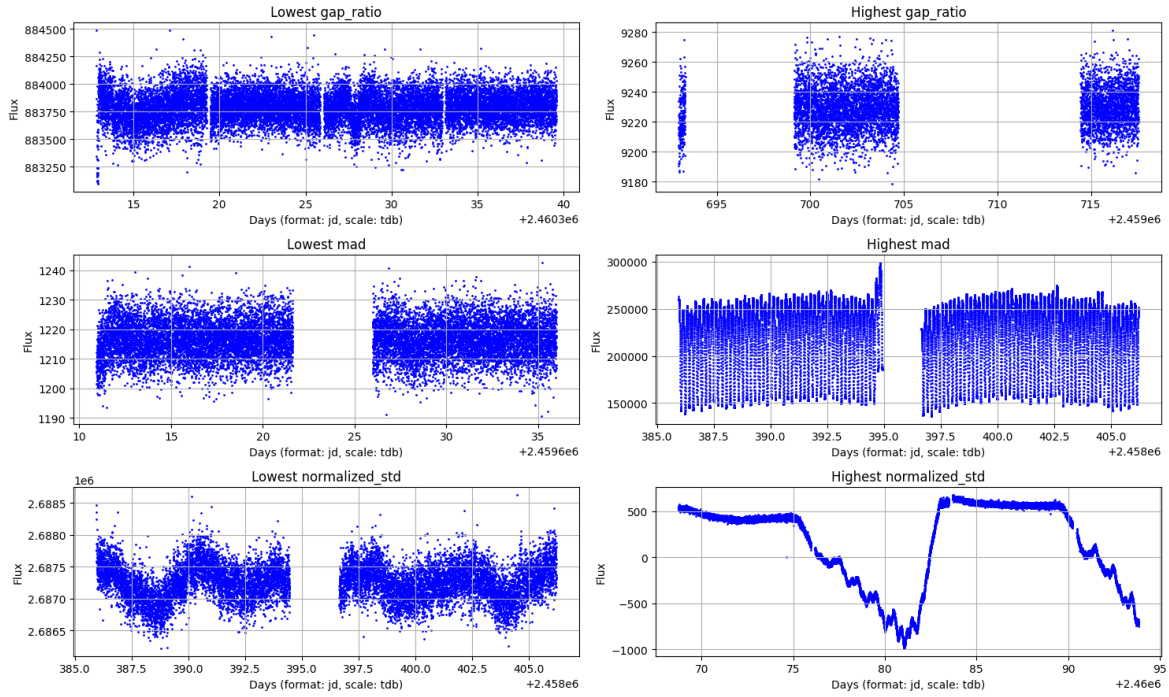


Figure 5.6: Comparison of light curves with lowest (left column) and highest (right column) values for gap ratio, MAD, and normalized STD. High MAD and STD values indicate noisier or potentially corrupted light curves.

The dataset was filtered to retain light curves meeting the following criteria: Time Gap Ratio ≤ 15

Following this filtering, the final dataset included 14,609 light curves from 3,213 stars. The test set, consisting of known exoplanet observations, was filtered similarly, resulting in 2,250 light curves from 706 stars, forming a reliable basis for model evaluation.

5.2.3 Balancing the Dataset

In constructing the training dataset, it is important to ensure a balanced representation of light curves containing no transits, a single transit, and multiple transits. Figure 5.8 illustrates this balance, displaying the distributions of transit appearances, transit midpoints, and orbital periods for three different dataset configurations.

Given that a planet orbiting close to its star will complete its orbit more quickly, multiple transits may appear in a single light curve. On the other hand, planets in wider orbits may exhibit only one or even zero transits in a single observation period. A balanced dataset should have an approximately equal representation of these

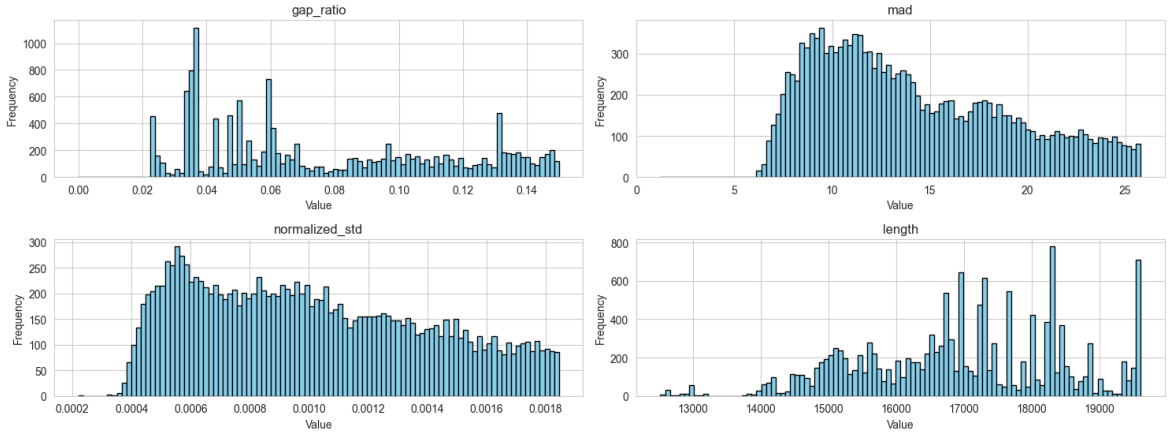


Figure 5.7: Distribution of calculated quality metrics for the filtered set of light curves.

three classes (no transits, one transit, multiple transits) to avoid biasing the model's predictions toward any specific class. This balance is especially important for cases with a single observed transit, as the period is indeterminate in such instances. There is a method for estimating the period from a single transit under specific assumptions [41]; however, this possibility is left for future work.

Balancing both the midpoint and period distributions simultaneously introduces certain constraints due to the inherent relationship between these parameters. For a given orbital period P that is smaller than the observational time span Δt, the transit midpoint t_{mid} is constrained by the inequality

$$0 \leq t_{\text{mid}} \leq \min(P, \Delta t - P).$$

As a consequence, the midpoint distribution tends to concentrate at earlier times for shorter periods, as the close spacing of transits restricts the possible range of initial midpoints. For longer periods, however, the midpoint can be more uniformly distributed, as only a single transit is likely to appear within the light curve duration Δt.

The top row of Figure 5.8 shows the configuration with fully balanced classes, where light curves with no transits, single, and multiple transit observations appear in equal proportions. This "fully balanced" setup results in a midpoint distribution that is skewed towards earlier times in the light curve, and then assumes a uniform distribution after $\frac{\Delta t}{2}$, when only one transit is visible in the light curve. The period distribution shows a uniform spread within a constrained range $[0.1\Delta t, 0.9\Delta t]$.

The middle and bottom rows show the marginal distributions of the midpoint and periods parameters for "one transit" and "many transits" configurations. In the middle scenario, the light curve contain only one transit and the period cannot be calculated so, by convention, it is set to 0.

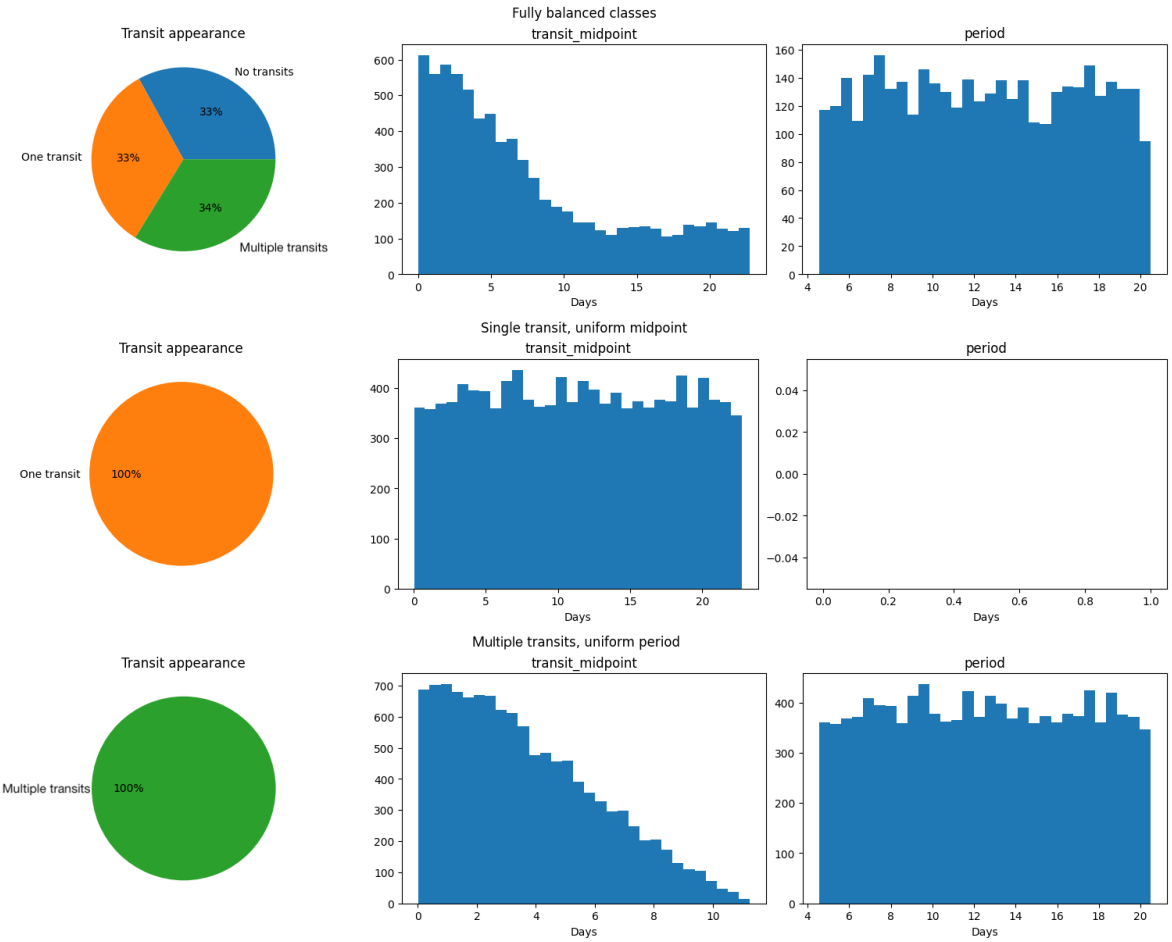


Figure 5.8: Distributions of transit appearance, midpoint, and period for three balanced datasets. Top: Fully balanced classes with no transits, one transit, and multiple transits. Middle: Single transit cases with a uniform midpoint distribution. Bottom: Multiple transits with a uniform period distribution.

In the bottom row, the period is uniformly distributed while the midpoint shows a natural skew towards earlier values due to the constrain explained above. Since at least two transits are visible, the transit midpoint time is always less then half the length of the light curve $\frac{\Delta t}{2}$.

For a visualization of a sample of the final dataset, refer to Appendix A.2

5.3 Model Definition

This section outlines the iterative development process of the final model by progressively building upon a simple base model. Starting with a minimal architecture, additional complexity is introduced in subsequent models with the goal of improving their capability.

5.3.1 Progressive Model Development

Lv0: Baseline Architecture

The Level 0 model introduces the foundational components required for the detection and regression tasks. It incorporates a multiclass classification task to identify the transit type and two regression tasks to predict the midpoint and period of the transit event. Because of the model's simplicity, it is not capable of processing long sequences; therefore, the training and evaluation phases use a generated light curve dataset of fixed length of 1024 points, corresponding to approximately 1.5 days of observations each.

Embedder: The embedder combines time and flux information into a unified representation. The two tensors are first passed through separate point-wise feedforward neural networks and the outputs are summed element-wise to produce a single sequence of 1024 elements. A point-wise network has one single input neuron, one hidden layer of variable size and an output layer of also variable size, separated by a non-linear *tanh* activation function. The size of the hidden layer and the output features are controllable via hyperparameters, but for the baseline model both have size 1. A simplified representation is shown in Figure 5.9.

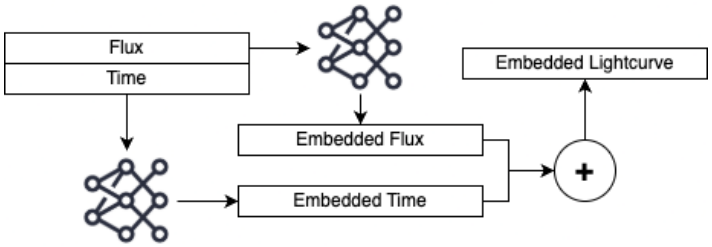


Figure 5.9: Linear embedding of time and flux information.

At this stage, the model performs a batch normalization on the resulted batch of embedded light curves, which aims to reduce the input data covariate shift by normalizing the data over the batch dimension, helping the model to converge.

Encoder: The encoder is a funnel feedforward neural network comprising two hidden layers. It takes the 1024-dimensional input from the embedder and compresses it into a latent representation of size 32. Funnel networks are fully connected neural networks where the size of the hidden layers is linearly interpolated between the input and output sizes, resulting in a structure that progressively compresses information at each layer. This design encourages the model to learn compact representations of the input while also reducing its overall size when the final layer is small. The activation function used

in the network is Snake, a semi-periodic monotonic function that better approximates periodic signals compared to classic activation functions like ReLU, Tanh, and Sigmoid [52]. The equation for Snake is

$$\text{Snake}(x) = x + \frac{\sin(\alpha x)^2}{\alpha}$$

where α is a tunable parameter, left to 1 by default.

Task Networks: The latent representation is processed by task-specific networks. A multiclass classifier uses a funnel feed forward network with a variable number of hidden layers (specified as hyperparameter) that takes the 32 dimension latent representation and outputs 3 values, corresponding to the prediction logits for each class. The regression tasks for Midpoint and Period use a similar architecture, but the output layer only produces one result. Each task has its own instance of the funnel network.

Lv1: Introducing Convolutional Layers

Building on Lv0 model, the Level 1 model replaces the feedforward encoder with a series of 1D convolutional layers. Each convolutional layer is followed by a max-pooling operation with a kernel size of 2 and stride of 2, halving the input dimension at each layer and doubling the channels. This enables the model to capture local features in the data. The convolutional layers support hyper-parametrization to allow more degrees of freedom during model optimization:

- **Embedding Dimension:** Controls the dimensionality of the embedder output (e.g., $N \times D$ instead of $N \times 1$).
- **Convolutional Layers:** Number of layers and kernel size.
- **Task Network Depth:** Number of hidden layers in the classifier and regression networks.

The model still operates on a fixed length of 1024 observations per light curve, to compare the results with the baseline Lv0 model.

Lv2: Long Sequence Processing

To handle longer light curve sequences, the Level 2 model modifies the embedder to process longer inputs through the use of a custom `FluxPatchEmbedder` module. In

short, this module breaks down the input light curve to equally sized patches, embeds them in a lower dimensional space and concatenate them back together.

The input consists of a time-embedded flux sequence of shape (B, N, d_{in}), where B is the batch size, N is the total sequence length, and d_{in} is the embedding dimension of the time-flux data. From Lv2 onwards, N is fixed to 2^{14} points (approximately 23 days). The module begins by dividing the input sequence into non-overlapping patches of size $M \times d_{\text{in}}$; where M is the patch length. The patch length M is a hyperparameter, and for simplicity, it is forced to be a power of two to ensure that the embedded sequence has a length compatible with the convolutional layers in the Encoder.

Each $M \times d_{\text{in}}$ patch is then processed independently by a funnel feedforward network, producing an embedded patch of size $D < (M \times d_{\text{in}})$, which are concatenated together to form the final embedded sequence of size $(N/M) \times D$. This transformation reduces the sequence length, allowing for efficient processing in subsequent layers while preserving the temporal and flux information from the original light curve. The resulting sequence is then passed through the same encoder and task networks as in Lv1.

Lv3: Final Architecture with Conformer

The final and most complex model proposed in this thesis is the Level 3 model, which replaces the linear embedder module for a more sophisticated Time Modulator, and the 1D convolutions with a Conformer model to capture both local and global patterns in the light curve data.

The time modulation process encodes a periodic signal into the flux using sinusoidal harmonics. The harmonics are indexed by an integer vector k of size n_{h}. The normalized time modulation t_m term is computed as:

$$t_m = \frac{2\pi k t}{t_{\max}},$$

where t is the input time, and $t_{\max}$ is the maximum time value in the sequence. For each harmonic k_i, sinusoidal components $\sin(t_m)$ and $\cos(t_m)$ are scaled and biased by learned matrices $S_{\sin}$, $S_{\cos}$, $B_{\sin}$, and $B_{\cos}$, with dimensions $(n_{\text{h}}, d_{\text{embed}})$, where d_{embed} is the embedding dimension. The modulated flux $f'(t)$ is given by:

$$f'(t) = f \cdot (\sin(t_m)S_{\sin} + \cos(t_m)S_{\cos}) + \sin(t_m)B_{\sin} + \cos(t_m)B_{\cos}.$$

Here, $\sin(t_m)$ and $\cos(t_m)$ are also matrices and the scaling and bias terms are obtained by applying matrix multiplications with the respective learned parameters.

The Conformer component, as briefly seen in Section 4.3, is composed of a stack of Conformer blocks, each containing the following modules arranged sequentially:

1. **Feedforward Network:** A single-hidden-layer network using the Swish activation function. Swish, unlike ReLU, is smoothed near zero, allowing small negative outputs and mitigating the dying ReLU problem, where neurons may collapse to always output zero.

2. **Multi-Head Self-Attention (MHSA):** The attention mechanism computes relationships between all elements of the sequence, capturing global dependencies. This implementation incorporates rotary embeddings (RoPE) [44] within the MHSA to encode relative positional information of the input tokens, similar to the AstroConformer approach.

3. **Convolutional Module:** To capture local features, the Conformer block includes a convolutional module with causal padding, ensuring that the sequence's temporal ordering is preserved [15].

4. **Second Feedforward Network:** A second feedforward network follows, further refining the features extracted by the previous modules.

5. **Layer Normalization:** Each Conformer block concludes with a layer normalization step, which stabilizes training and improves convergence.

AstroConformer, compared to the original Conformer design by Gulati et al. [15], removed the feedforward networks to reduce parameter count, arguing that these networks provided minimal benefits. In this work, the feedforward networks were retained as part of the architecture, as the implementation was adapted from an open-source GitHub project*, which closely follows the original Conformer design. Another key difference is that the original Conformer does not include the RoPE mechanism, which I incorporated into the MHSA module by modifying the Conformer implementation with source code adapted and adjusted from another GitHub project†. Although the AstroConformer source code was also available on GitHub, its implementation was rigid and difficult to integrate into my project's workflow. This motivated the decision to adapt the more flexible Conformer implementation while adding RoPE to align with the relative positional encoding advancements presented in AstroConformer.

Residual connections are applied throughout the Conformer block to ensure efficient gradient flow and mitigate vanishing gradient issues. At each step, the inputs from

*https://github.com/lucidrains/conformer
†https://github.com/lucidrains/rotary-embedding-torch

the previous layer are added to the outputs of the current step, enabling the model to preserve information as it propagates through the layers.

The RoPE mechanism used within the Conformer enhances the self-attention mechanism by encoding relative positional relationships between sequence elements [44]. Unlike traditional positional encodings, which rely on fixed or learned vectors added to the input embeddings, RoPE introduces relative position information directly into the self-attention computation. This is achieved by rotating the input vectors in a multidimensional space based on their positional indices. The advantage of RoPE lies in its ability to capture relative positional patterns without being constrained by absolute position indices, making it well-suited for timeseries where the sequence length and periodicity can vary significantly.

The Lv3 model pipeline begins by applying the Time Modulator to the input sequence, producing a modulated tensor enriched with temporal information. This tensor is then divided into fixed-size patches using the same `FluxPatchEmbedder` as Lv2 model. Before invoking the Conformer, a learnable token is appended to the output of the FluxPatchEmbedder. This approach takes advantage of the Transformer's ability of preserving the order of tokens between its input and output. By including this additional token, the model introduces a specific vector that is designed to aggregate and represent high-level information from the entire sequence during processing. Once the extended input is passed through the Conformer, the Lv3 model extracts the processed token, which now encapsulates a summarized representation of the input. This token is then used as input for the downstream tasks.

5.3.2 Software Architecture

To ensure a unified and scalable software architecture design, the multitask models used in this thesis are implemented to inherit from a common parent class, `MultitaskModel`, which itself extends `torch.nn.Module`. This inheritance allows to reuse the same code for training, evaluation, and diagnostic without modifications for each specific model subclass. The parent class defines the shared interface, including `forward()` and `predict()` methods. Derived models (Lv0 to Lv3) define their internal neural network architecture within the class constructor, and each defines a customized implementation of the interface methods. Each model also defines what loss functions to use for each task, and how to generate its hyperparameters configuration given an `optuna.Study` object.

The input to all models is a three-dimensional `torch.Tensor` of size `[batch_size, sequence_length, 2]`. The `batch_size` dimension specifies the number of light

curves processed simultaneously during training, with a default value of 128. The `sequence_length` dimension corresponds to the length of the input light curve, which is set to 2^{10} for Lv0 and Lv1 models and extended to 2^{14} for Lv2 and Lv3 models. The final dimension represents the two features provided for each time step: the time and flux values.

The models output a dictionary, where each key corresponds to a specific task identifier, and the associated value is the tensor produced by the module responsible for that task. This design allows for flexibility in the number and type of tasks that can be incorporated into the framework. Additionally, the outputs of individual tasks may differ in size or structure, accommodating a wide range of objectives. For instance, the classification task outputs a tensor $\hat{y}_l \in \mathbb{R}^3$, representing logits for the three categories of interest. In contrast, regression tasks output single-element tensors corresponding to scalar predictions, midpoint $\hat{y_m} \in \mathbb{R}$ and period $\hat{y_p} \in \mathbb{R}$.

An advantage of this approach is the ability to assign a dedicated loss function to each task, and task-specific weights can be introduced to emphasize certain objectives if required by the experimental design or data distribution. However, in this thesis, all tasks are treated with equal importance, and no task weighting is applied.

5.3.3 Model Training

The training process is implemented using the PyTorch Lightning framework, specifically, the `LightningModule` class. PyTorch Lightning is an abstraction built on top of the PyTorch deep learning library, designed to streamline common tasks such as training loop management, logging and checkpointing. The `LightningModule` is customizable, granting users the possibility to override any step of the training process stage, while still benefiting from the automation of repetitive tasks.

The optimizer used during training is AdamW, a modification of the Adam optimizer that addresses limitations in its regularization behavior by decoupling weight decay from the gradient updates used for minimizing the loss function [25]. It applies weight decay directly to the parameters, independent of the gradient-based update, which improves convergence and simplifies hyperparameter tuning, as the optimal settings for weight decay and learning rate can be optimized independently.

The training loop extends the default PyTorch Lightning implementation to incorporate several additional features: the initialization phase of the `LightningModule` takes as input the `MultitaskModel` to be trained, along with a `TrainParams` dataclass that defines the hyperparameters controlling the training process, specifically, the Learning

Rate and Weight Decay for the optimizer. After the forward pass of each data batch in the training loop, the training module computes the loss for each task and accumulates it into a dedicated data structure, to be used later for model evaluation and visualization of the train/validation loss plot.

To monitor the training dynamics, the gradient norms are recorded after each optimization step. These values are used to construct a gradient heatmap, providing a detailed visualization of how the model updates its parameters over the course of training. Additionally, the learning rate is managed using PyTorchâs `ReduceLROnPlateau` component, which dynamically reduces the learning rate when the validation loss plateaus. Specifically, the scheduler monitors the validation loss and reduces the learning rate by a factor of 10 after 10 epochs of no improvement in the monitored metric. The learning rate is allowed to decrease to a minimum value of 1×10^{-6}.

Hyperparameter tuning also part of the training pipeline, implemented using the Optuna Python package. Optuna is a framework for efficient hyperparameter optimization, employing a tree-structured Parzen estimator (TPE) as its underlying algorithm. By adaptively sampling hyperparameter combinations based on past performance, Optuna accelerates the search for optimal configurations compared to traditional grid or random search methods [1].

The hyperparameter tuning process is managed by a custom `AutoTrainer` class, which evaluates the model's performance after training it for a limited number of epochs with different hyperparameter settings. The training parameters considered during optimization include learning rate and weight decay, while additional model-specific parameters are tuned depending on the model. The `AutoTrainer` accepts two key inputs: the number of trials, specifying how many hyperparameter combinations to explore, and the number of epochs per trial, determining how long each model is trained before evaluation. In this thesis, 8 trials and 12 training epochs per trial are used for the optimization loop. The hyperparameters used for training the final models are reported in the Appendix A.3.

After completing the hyperparameter search, the model configuration achieving the lowest loss of all the tasks combined is selected for an extended training phase. This final round of training spans 100 epochs, allowing the chosen model to fully train under the optimized settings.

For the training phase, six distinct datasets are generated: three for Lv0 and Lv1 models (one dataset per planet type), where the light curves are cropped to a length of 2^{10}, and three more for Lv2 and Lv3 models, where the light curves are cropped or padded to 2^{14} data points. Each dataset entry consists of a bundle containing:

- **X (input data):** A $N \times 3$ `numpy` array storing a single light curve's time and flux data, together with the generated clean transit signals based on the sampled planetary parameters.
- **Y (target data):** The transit midpoint and period used to generate the synthetic transits, as well as the numerical class label indicating the transit category (no transits, one transit, or multiple transits).

To ensure reproducibility, the planetary parameter generation process uses a fixed random seed to guarantee a reproducible dataset creation process across experiments. The generated datasets are stored in a `HDF5` file, a format chosen for its efficiency in handling large numerical datasets and its capability for random data access without loading the entire file into memory. This is particularly advantageous in distributed training scenarios, as they minimize memory usage across multiple processes. Data loading is handled by a `torch.DataLoader`, wrapped within a PyTorch Lightning `LightningDataModule`. The dataset is split into training, validation, and test sets with proportions of 70%, 15%, and 15%, respectively, and a seed is also specified to ensure a reproducible random split. The training dataset indices are reshuffled at every epoch to enhance model generalization. To further optimize input pipelines for GPU training, the `DataLoader` uses multiple CPUs for parallel data loading and uses memory pinning, which improves data transfer efficiency between host and device memory, reducing training times.

Within the training loop, the light curve data is processed by combining synthetic transit signals with the original flux values, with a parameter k controlling the intensity of the perturbation, as detailed in Section 5.2.1. Initially, as part of an earlier hypothesis, k was adjusted dynamically based on the current training epoch, with the idea of introducing noise gradually as a regularization technique. However, this approach resulted in the opposite effect, as the models showcased difficulty in learning effectively, demonstrated by erratic train/validation loss trends. Consequently, the parameter was fixed at a value of 1 for all experiments. After applying the flux perturbation, both the time and flux components are normalized to have zero mean and unit variance. The normalization parameters are preserved to allow de-normalization of the model's output during post-analysis. Additionally, the flux values are inverted (multiplied by -1) to aid the convolutional layers in identifying dips more effectively, as MaxPool operations favor positive values.

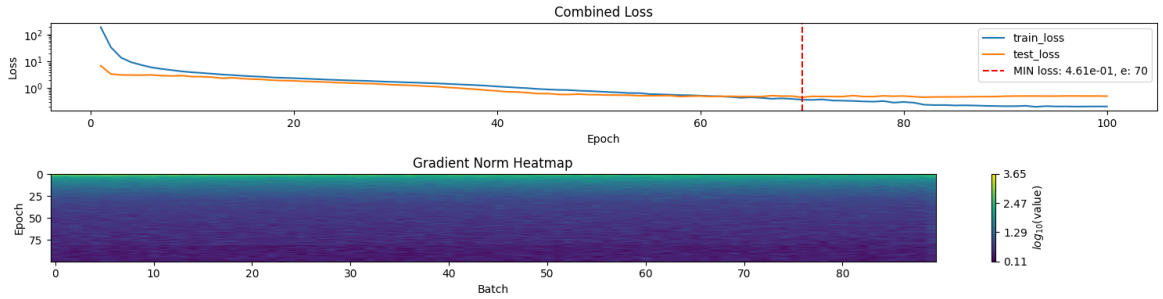
6. Results and Evaluation

This chapter focuses on the evaluation of the fully trained models, Lv0 to Lv3. Key metrics for classification include the accuracy, Area under the Curve (AUC) and F1-score, while regression accuracy is assessed using the root mean square error (RMSE) and residual trends for transit midpoints and orbital periods. Additional information and detailed evaluation plots are reported in Appendix B.

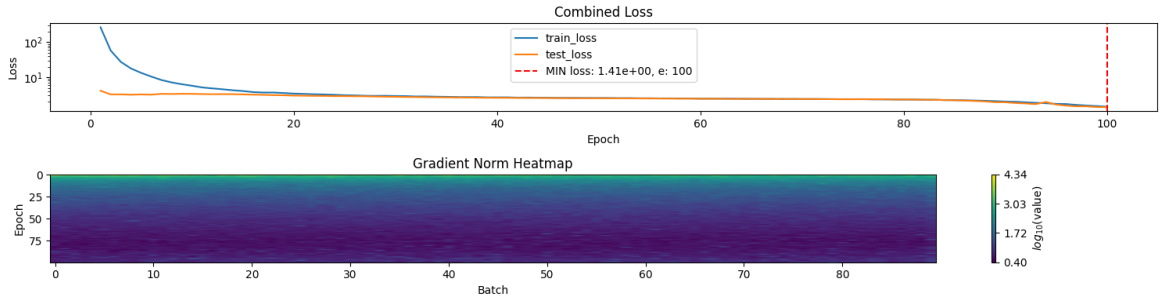
6.1 Train-Validation Loss Analysis

The train-validation loss plot is a fundamental tool for evaluating the performance and generalization of machine learning models. It provides insights into whether a model is underfitting, overfitting, or generalizing well by analyzing the trend of loss values computed at each epoch for the training and validation datasets. Ideally, both losses should decrease smoothly over the epochs, maintaining close alignment and avoiding abrupt spikes or divergences. In addition to loss trends, the analysis of gradient norms offers valuable information about the optimization dynamics. Smooth and stable gradients indicate effective learning, while excessively high gradients can signal instabilities, and overly small gradients may point to slow convergence. By combining these metrics, the train-validation loss plots and gradient heatmaps allow for a comprehensive assessment of model behavior during training. Figures 6.1, 6.2, 6.3, and 6.4 summarize the results for Lv0, Lv1, Lv2, and Lv3 models, respectively.

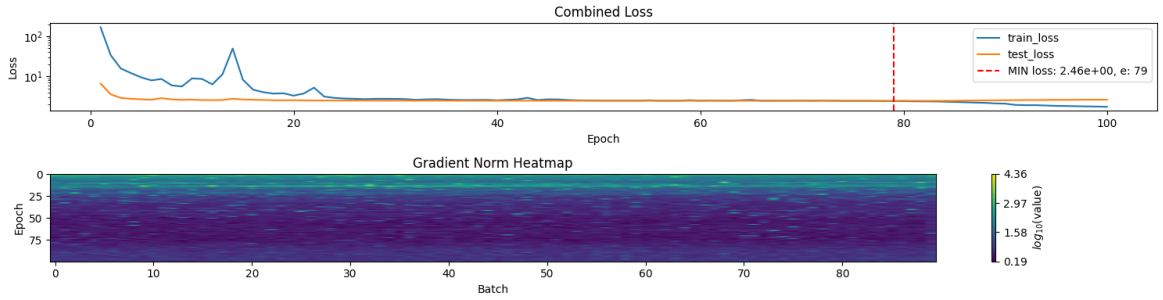
The loss curves for the Lv0 models in Figure 6.1 show rapid loss reduction during the first 15 epochs, followed by slower convergence and eventual stabilization. The Lv0 Jupiter model (Figure 6.1a) steadily reduces training loss until epoch 70, after which validation loss diverges, suggesting overfitting. The Lv0 Neptune and Mini-Neptune models stabilize earlier, but the early plateau in validation loss and generally high loss value shows lack of generalization and underfitting. However, the Neptune model exhibits a loss reduction around epoch 90, indicating further optimization potential. Gradient norm heatmaps reveal high gradients during early epochs, corresponding to



(a) Lv0 Jupiter model



(b) Lv0 Neptune model

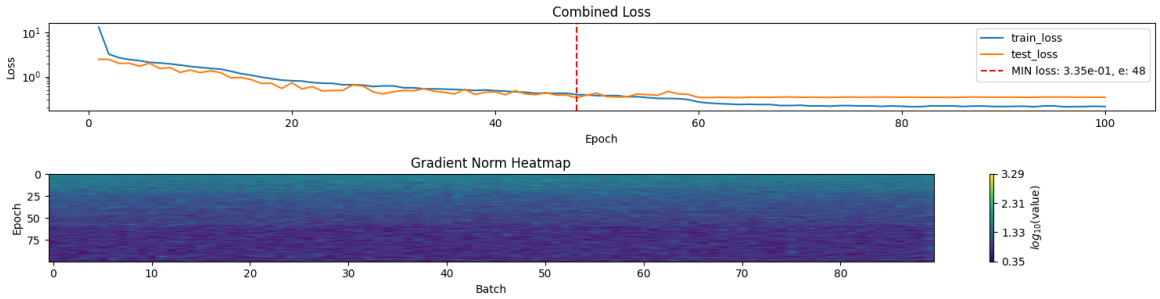


(c) Lv0 Mini-Neptune model

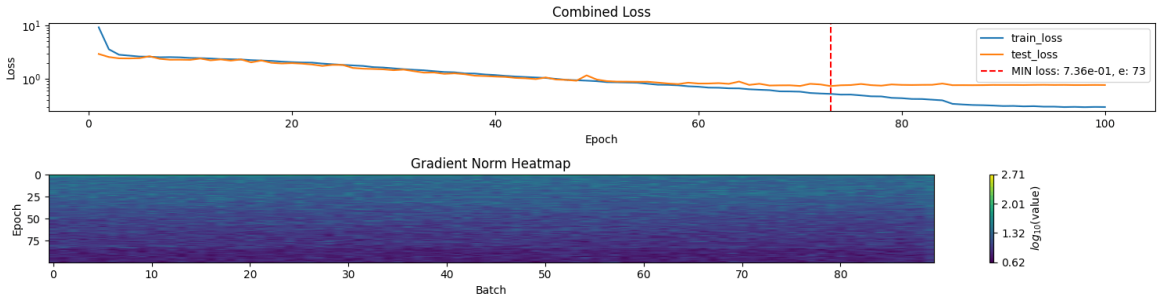
Figure 6.1: Training and validation loss curves for Lv0 models across planet types.

the initial steep loss decrease. Gradients stabilize to smaller values over epochs, but Mini-Neptune gradients remain noisier.

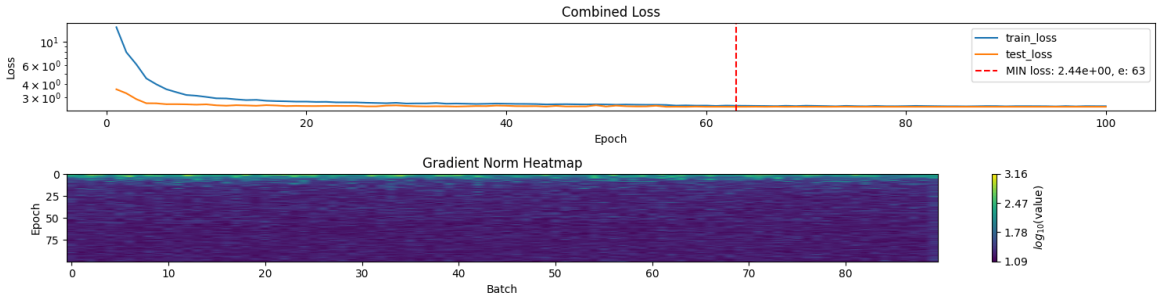
The Lv1 models exhibit varying behaviors in terms of overfitting and stabilization across planet types (Figure 6.2). All models show a rapid initial loss decrease before epoch 10, followed by slower convergence. The Lv1 Jupiter model displays a steady learning up to epoch 48, after which the losses plateau and slightly diverge. In the Neptune model, overfitting is evident after epoch 60, as training loss continues to decrease while validation loss stabilizes, diverging visibly. The Lv1 Mini-Neptune model demonstrates early stabilization, with both training and validation loss plateauing around epoch 40. Like Lv0, the early plateauing of validation loss indicates poor learning and generalization capabilities. The gradient norm heatmaps for all models reveal smooth transition of gradients throughout the training epochs, with higher



(a) Lv1 Jupiter model



(b) Lv1 Neptune model

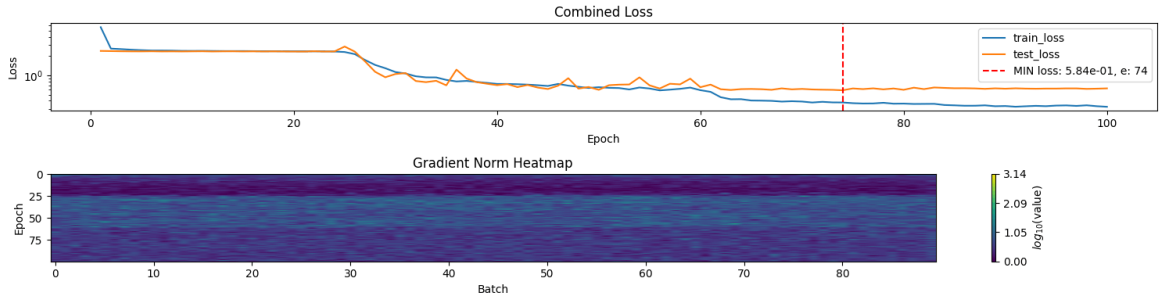


(c) Lv1 Mini-Neptune model

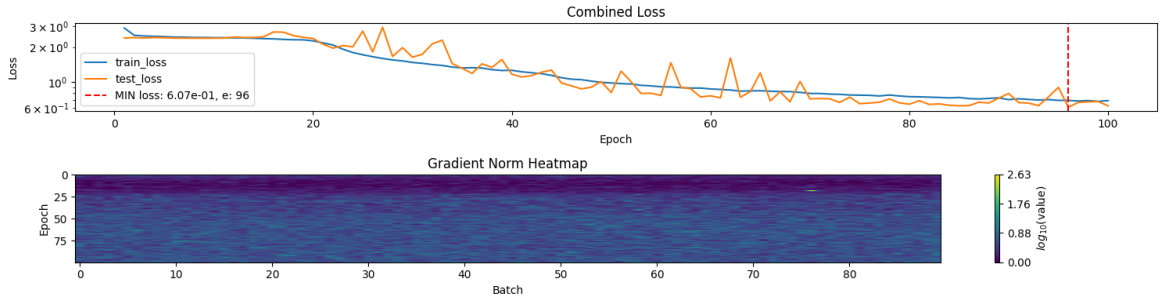
Figure 6.2: Training and validation loss curves for Lv1 models across planet types.

magnitudes during the initial epochs, consistent with the observed loss reduction.

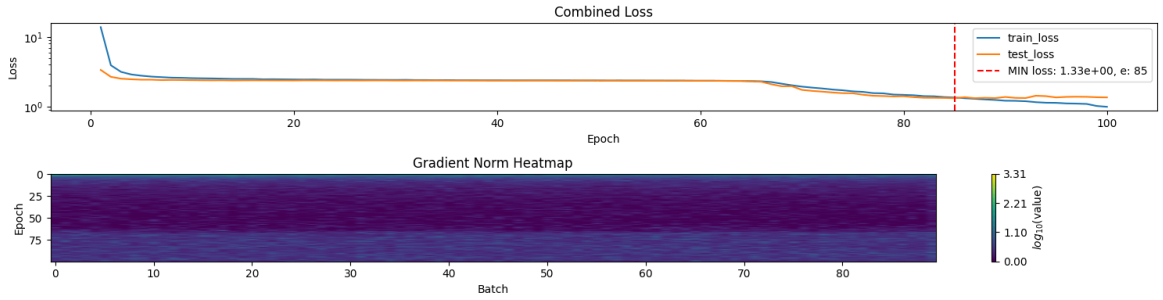
The Lv2 models show two distinct phases of loss reduction: an initial steep decrease followed by a second, smaller decline later in training (Figure 6.3). This behavior aligns with the effect of reducing the learning rate upon reaching a plateau. For the Lv2 Jupiter model, the decreases in loss happen first at epoch 25 and the second around epoch 60. However, overfitting begins shortly thereafter, as indicated by the divergence between training and validation loss. The Lv2 Neptune model demonstrates a smooth and gradual loss reduction starting at epoch 20, which persists throughout training. Despite the validation loss showing a saw-tooth pattern, the spike peaks reduce with the epochs, following the overall descending trend of the training loss. The Lv2 Mini-Neptune model experiences its second loss decrease around epoch 67. Both training and validation losses remain closely aligned, with a small divergence towards the end. However, the



(a) Lv2 Jupiter model



(b) Lv2 Neptune model



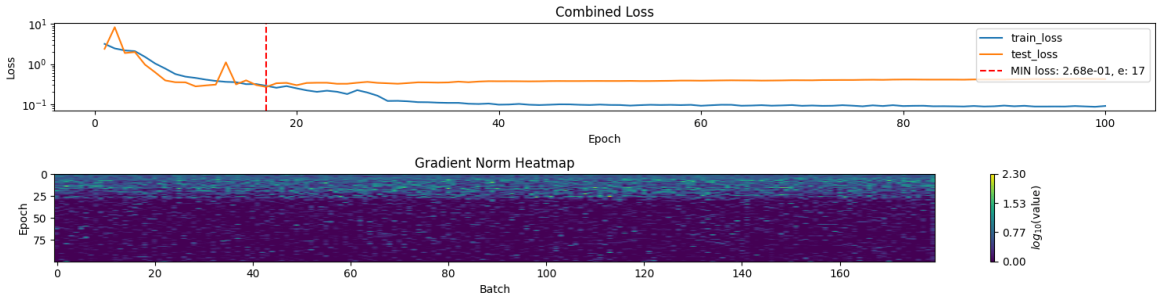
(c) Lv2 Mini-Neptune model

Figure 6.3: Training and validation loss curves for Lv2 models across planet types.

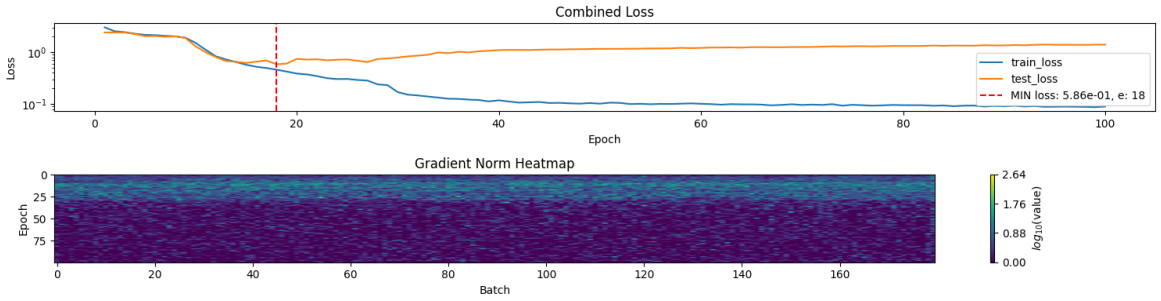
overall higher loss values suggest underfitting. The gradient norm heatmaps for all models show band-like patterns, where periods of higher gradients correspond to epochs of significant loss reduction. This is most prominent in the Jupiter model, with three clearly visible bands aligning with its three learning phases delimited by the jumps in the loss.

The Lv3 models overfit across all tasks and planet types early in training, with divergence occurring before epoch 20 for the Jupiter and Neptune models, and around epoch 30 for the Mini-Neptune model. The only exception is the transit midpoint estimator for the Jupiter model (not included in the plot), where training and validation losses decrease together throughout training, indicating stable performance for this specific task.

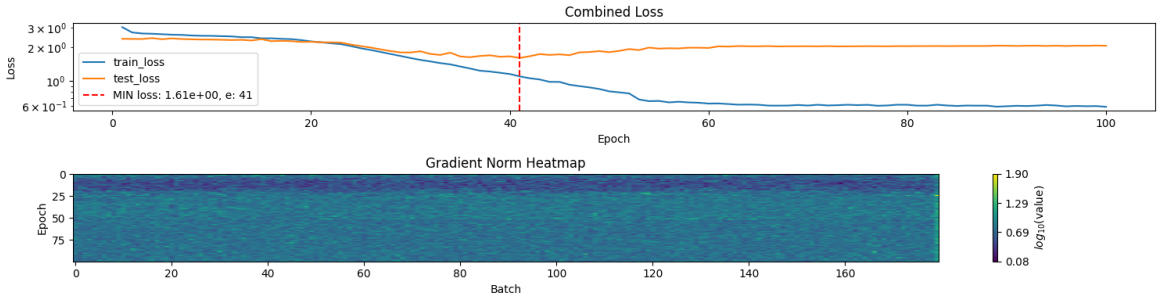
The gradient norm heatmaps for Lv3 models are significantly noisier compared to previous levels, reflecting instability in training. This is particularly evident in the Nep-



(a) Lv3 Jupiter model



(b) Lv3 Neptune model



(c) Lv3 Mini-Neptune model

Figure 6.4: Training and validation loss curves for Lv3 models across planet types.

tune and Mini-Neptune models, where noisy gradients suggest difficulty in convergence and poor optimization. These results highlight the need for stronger regularization techniques or adjustments in learning rate schedules to mitigate overfitting and stabilize training.

6.2 Multiclass Transit Detector Analysis

The evaluation of the multiclass transit detector models reveals strong performance across planet types, with notable differences between larger and smaller planets. The metrics analyzed include accuracy, weighted F1-score, and area under the curve (AUC), as summarized in Figure 6.5. The area under the curve (AUC) is a metric derived from the receiver operating characteristic (ROC) curve. It quantifies the model's ability to

distinguish between classes. An AUC of 1.0 indicates perfect classification, while an AUC of 0.5 suggests random guessing. For multiclass classifiers, the AUC is typically averaged over all classes, providing a single score that reflects overall discriminative ability.

The weighted F1-score is a harmonic mean of precision and recall, weighted by the number of samples in each class. This ensures that the metric reflects performance across all classes, even when the dataset is imbalanced. A higher weighted F1-score indicates balanced precision and recall across the classes, making it a robust measure for evaluating multiclass classifiers.

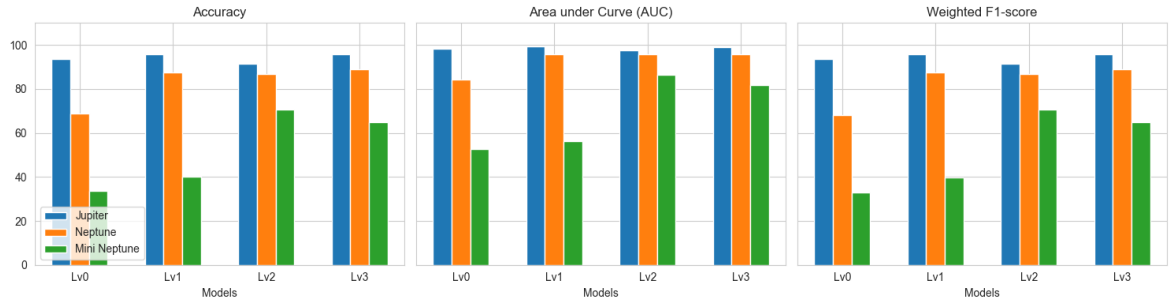


Figure 6.5: Evaluation metrics for the multiclass transit classifier task, including accuracy, area under the curve (AUC), and weighted F1-score, across all planet types and model levels.

Overall, all models perform exceptionally well in detecting whether a light curve contains transit signals for larger planets, particularly Jupiter-like planets. The accuracy for Jupiter models ranges from 93.49% for Lv0 to 95.89% for Lv3, demonstrating consistently high performance across levels. For Neptune and Mini-Neptune planets, the accuracy scores decline, as expected, due to the smaller size and lower signal-to-noise ratio of these transits. The Lv3 model achieves the highest accuracy for both Jupiter and Neptune types, with scores of 95.89% and 89.01%, respectively, closely followed by Lv2 model. However, for Mini-Neptune planet types, Lv2 outperforms Lv3, achieving an accuracy of 70.78%, compared to Lv3's 64.92%.

The weighted F1-scores closely align with the accuracy scores for all planet types and model levels, indicating balanced precision and recall. Similarly, the AUC metric shows that all models are effective in making accurate predictions for Jupiter-like planets, with AUC values exceeding 97% for all levels. The Neptune models also perform well, particularly Lv3, which achieves an AUC of 95.81%. However, for Mini-Neptune models, Lv0 and Lv1 score an AUC of 52.64% and 56.15%, respectively, indicating their inability to reliably detect smaller planets. Lv2 is the most performant, reaching a score of 86.58%, surpassing Lv3's 81.74%, which is an exiting result given the challenging task.

6.3 Comparison of Transit Midpoint and Period Regressors

The performance of the regression tasks, predicting transit midpoint and orbital period, was evaluated using the R^2 metric, root mean square error (RMSE), and median relative error. The R^2 metric measures the explanatory power of the models, where a value of 1 represents perfect predictions, and values below 0 indicate that the model performs worse than simply using the mean of the training data. For this analysis, R^2 scores below 0 are clipped to 0. Figure 6.6 presents the results across planet types and model levels.

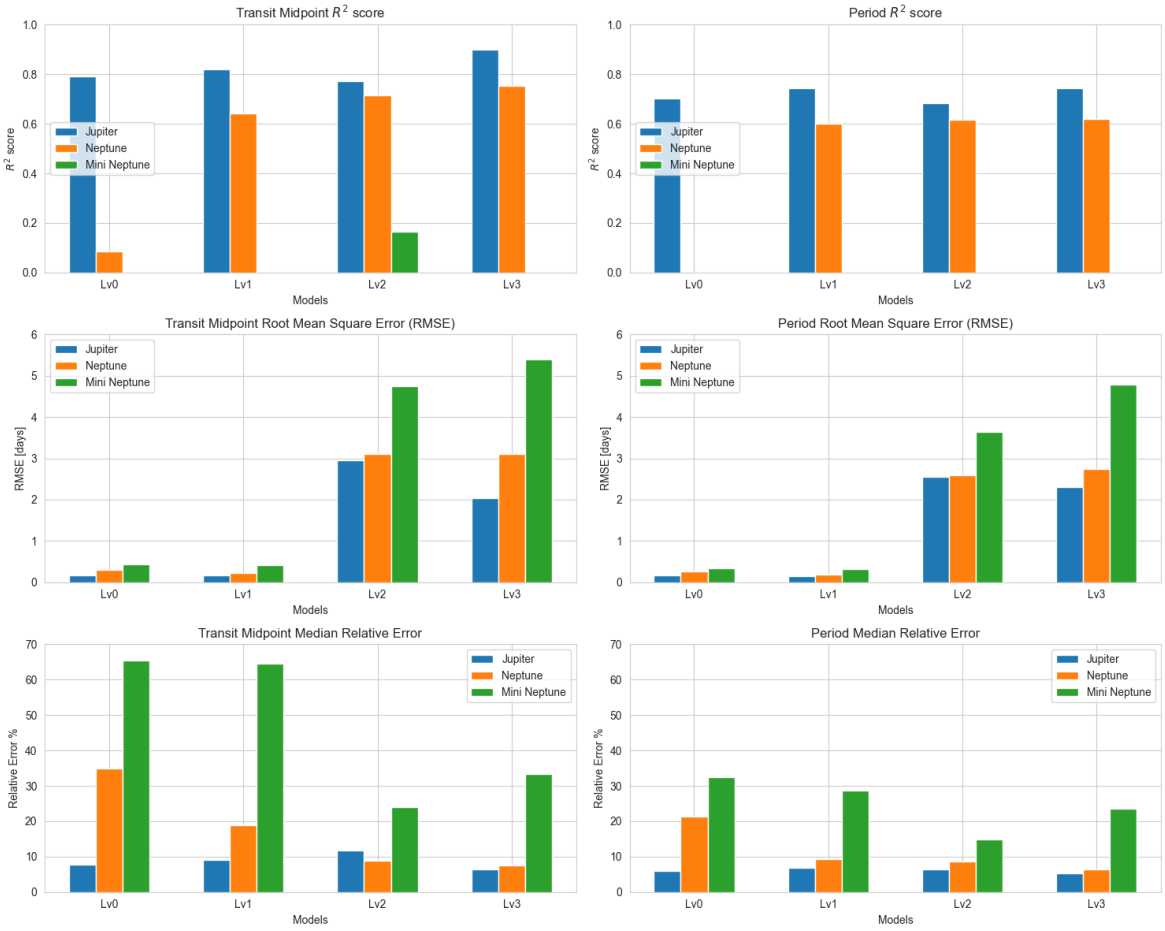


Figure 6.6: Evaluation metrics for regression tasks predicting transit midpoint and orbital period across planet types and model levels. Metrics include R^2 score, RMSE, and median relative error.

The R^2 scores highlight the inability of all models to predict the transit midpoint and period for Mini-Neptune planets, with scores cropped to 0 in all cases except for Lv2, which achieves a marginally positive R^2 score of 0.17 for the transit midpoint task. This indicates very limited explanatory power. In contrast, models demonstrate better

performance for Jupiter and Neptune planets. Transit midpoint predictions achieve higher R^2 scores, with Lv3 scoring 0.90 for Jupiter planets and 0.75 for Neptune planets. Period predictions are consistently less accurate, with R^2 scores generally below 0.75.

The disparity between transit midpoint and period predictions can be attributed to the dataset distribution. Only one-third of the data points contain multiple transits, limiting the training data available for the period regression task. In contrast, two-thirds of the data points include at least one midpoint, providing more data for training the transit midpoint regressor. Adjusting the training process to weight the loss of the period regressor task more heavily could help address this issue.

The RMSE values support the R^2 findings, showing significantly higher errors for Mini-Neptune predictions compared to Jupiter and Neptune predictions. A clear distinction emerges between models trained on cropped light curves of 1024 data points (Lv0 and Lv1) and those trained on full-length light curves of 2^{14} data points (Lv2 and Lv3). Models using cropped light curves consistently achieve lower RMSE values across all tasks and planet types. For example, Lv0 achieves an RMSE of 0.18 days for transit midpoint predictions and 0.17 days for period predictions on Jupiter planets, compared to 2.03 and 2.31 days, respectively, for Lv3. This significant difference can be explained by the difference in scaling and prominence of the transit signals in the cropped light curves: since transits occupy a larger fraction of the input data in shorter sequences, they become easier for the models to identify. Moreover, to ensure consistency in the distribution of transit midpoints and periods between the datasets used to train models with different input lengths, the values of midpoints and periods were scaled according to the maximum sequence length, making their range significantly smaller than the values in the longer sequences, which further impacts the results.

The contrast between RMSE and median relative error reveals that Lv0 and Lv1 models, despite their lower RMSE values, generally have higher relative errors compared to Lv2 and Lv3. This is especially evident for Neptune and Mini-Neptune planets, where Lv0 and Lv1 display high relative errors, while Lv3 achieves lower relative errors, such as 7.43% and 6.48% for Neptune period and midpoint predictions, respectively. For Jupiter planets, relative errors are comparable across all models, with values below 10%, reflecting the larger and more detectable transit signals of these planets.

In summary, the regressors perform well for Jupiter and, to a lesser extent, Neptune planets, while failing to generalize for Mini-Neptune planets. The dataset imbalance between transit midpoint and period tasks significantly impacts performance, particularly for period regression. Future improvements could include rebalancing the dataset or applying task-specific loss weighting during training to address these challenges.

Despite the lower relative error for Lv2 and Lv3, the RMSE values remain too high for reliable predictions, especially compared with classical analysis tools.

6.4 Model Complexity Comparison

The parameter count for each model level is shown in Figure 6.7, highlighting differences in complexity and architectural design across Lv0 to Lv3. The results provide insights into how architectural choices affect model size and performance.

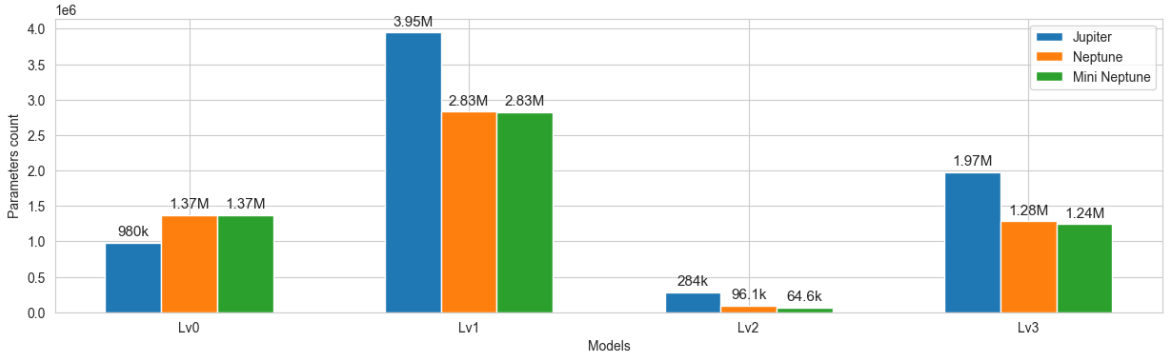


Figure 6.7: Parameter count for models across all levels (Lv0 to Lv3) and planet types. Lv0 and Lv1 are fully connected architectures tied to a sequence length of 1024, while Lv2 and Lv3 use patch encoders to preprocess the input light curve, reducing dimensionality.

The Lv1 model for the Jupiter dataset comprises nearly 4 million parameters, making it the largest among all evaluated models. This substantial size is primarily due to the latent space representation, which retains a dimensionality of 1024 elements after processing by the convolutional network. Consequently, the fully connected architectures of the task-specific models (classifier and regressor) that operate on this high-dimensional input result in a significantly larger number of parameters. In contrast, the Lv0 model embeds the input into a much smaller feature vector of size 32, leading to a far more compact architecture overall.

Lv2, in contrast, is a highly compact model, with parameter counts below 300,000 for Jupiter, and just below 100,000 for Neptune and Mini-Neptune tasks. Despite its small size, Lv2 achieves results comparable to the more complex Lv3. This compactness is achieved through the use of the patch encoder, which preprocesses the input light curve by dividing it into smaller patches and embedding them in a reduced-dimensional latent space. This approach minimizes the reliance on large fully connected layers and reduces the overall model size significantly.

Lv3 also uses a patch encoder but incorporates additional complexity within its conformer architecture. The use of 4 conformer blocks, each containing attention heads,

convolutional layers, and fully connected layers, significantly increases the parameter count. For Jupiter tasks, Lv3 has nearly 2 million parameters, approximately 7 times the size of Lv2. This increase in size, however, doesn't translate to a significant superior performance.

The comparison highlights the trade-offs between model size and architecture. While Lv0, Lv1 are parameter-heavy due to their fully connected architectures, they are also difficult to regularize. Lv2 offers a highly compact design while maintaining competitive performance, making it a promising candidate for further improvements and optimizations. Lv3, with its conformer-based architecture, is only marginally better than the simpler Lv2, suggesting that more refinements in the architecture and training process might be needed to fully benefit from this architecture.

This hypothesis is further supported by Pan et al. [35], the authors of Astroconformer, who achieved significant results by carefully finetuning multiple aspects of their training process. These include optimizing data quality and normalization, incorporating warmup and cooldown stages during training, and employing a specific learning rate scheduling.

6.5 Results Summary

Overall, Lv2 emerges as the most balanced model, combining compactness with competitive performance across classification and regression tasks. Lv3 offers additional capacity for complex tasks but requires stronger regularization and training strategies to mitigate overfitting. For Mini-Neptune detection and regression, all models face significant challenges, indicating a need for improved data representation and task-specific optimization. Future work should prioritize enhancing dataset diversity, particularly for smaller planets, to improve the detection performance. Increasing dataset size with additional light curves could help achieve the goal. Especially for Lv3's transformer-based architecture, further tests could involve reducing the search space for hyperparameters and focusing on key parameters such as learning rate and kernel size. Training for longer periods with more data could improve convergence, as transformer architectures typically require larger datasets. Additionally, investigating more regularization techniques, such as early stopping, and more suitable dynamic learning rate schedulers and warm-up strategies can help stabilize training for larger models like Lv3. Balancing dataset distributions and applying task-specific loss weighting will also address performance disparities between regression tasks. By addressing these challenges, future iterations of the models can achieve greater robustness and accuracy across diverse observational scenarios.

7. Conclusion

The detection of exoplanets has significantly progressed through the study of planet transits in light curve data from missions like TESS and Kepler. This thesis employs deep learning to aid the identification of transiting exoplanets from light curve data via a supervised multi-task learning approach. The estimated parameters, specifically the time of first transit midpoint and the orbital period, can be used to increase the signal-to-noise ratio of the transits through a folding operation applied to the light curve and help further characterize the planetary properties using classical analysis techniques.

To prepare the training dataset, a subset of Sun-like stars with no known exoplanets was curated, sampling from data aggregated from multiple sources, including Gaia Data Release 3, the TESS Input Catalog, NASA Exoplanet Archive, and TESS Objects of Interest. The selection process prioritized light curves with minimal gaps and irregularities to ensure data quality. Synthetic planetary parameters were then generated for three representative planet types (Jupiter, Neptune, and Mini-Neptune) to simulate realistic transit signals, which were superimposed onto the observational light curve data. The resulting supervised training dataset was carefully balanced across three distinct categories: light curves without transits, those containing a single transit, and those with multiple transits.

An incremental approach was adopted in the development of the deep learning models, beginning with a simple architecture and progressively increasing complexity. Four model architectures (Lv0, Lv1, Lv2, and Lv3) were designed and trained on the datasets corresponding to the three different planet types transit signals, resulting in a total of 12 trained models. Lv0, the simplest architecture, employed a fully connected network limited to processing small chunks of the light curve at a time. Lv1 introduced convolutional layers to extract local features, Lv2 incorporated a patch embedder to process the entire light curve sequence, and Lv3 uses a conformer as its core architecture to capture both long and short range patterns in the data.

The training process included a preliminary hyperparameter tuning phase using the

Optuna optimizer. Each model was trained for 12 epochs across 8 different hyperparameter configurations, and the best configuration based on the smallest validation loss was selected for further training. The selected models were then trained for 100 epochs.

The results demonstrated excellent accuracy for the multi-label classification task of detecting transit signals for Jupiter and Neptune-like planets. The regression tasks for estimating the transit midpoint and orbital period showed promising performance, achieving contained relative errors, though the RMSE remained too high for reliable real world applications. For Mini-Neptune like planets, none of the models produced noteworthy results, indicating challenges in detecting smaller or less pronounced transit signals.

In conclusion, this thesis highlights the potential of deep learning in the detection and characterization of exoplanetary transits, demonstrating strong performance in both classification accuracy and parameter estimation. While the models excel at detecting Jupiter- and Neptune-like planets, further refinement is necessary to improve the reliability of regression tasks and extend capabilities to smaller planet types.

Although the Lv2 and Lv3 models may not yet be suited to fully replace the more mature state-of-the-art classical methods, they hold significant promise as efficient preliminary screening tools. These models can quickly evaluate new data and flag particularly promising light curves for further, more detailed analysis. Moreover, their approximate predictions for orbital period and transit midpoint can serve as valuable prior information for subsequent analysis methods. By providing an initial estimate, these models have the potential to significantly reduce the computational time required to analyze light curves from scratch, thus enhancing the overall efficiency of exoplanet detection workflows.

* * *

The journey of exoplanet discovery is far from over. As future missions generate ever-increasing volumes of data, the role of machine learning will only deepen, empowering astronomers to uncover new worlds and, perhaps one day, answer the age-old question of whether we are alone in the cosmos. This work is a small but meaningful step in that direction, reaffirming our unyielding curiosity and the enduring quest to understand the Universe.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. KDD '19, page 2623â2631, New York, NY, USA, 2019. Association for Computing Machinery.

[2] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Van Essen, A. Awwal, and V. Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8:292, 03 2019.

[3] W. Borucki, D. Koch, G. Basri, N. Batalha, T. Brown, D. Caldwell, J. Christensen-Dalsgaard, W. Cochran, E. Dunham, T. N. Gautier, and et al. Finding earth-size planets in the habitable zone: the kepler mission. *Proceedings of the International Astronomical Union*, 3(S249):17–24, 2007.

[4] B. Campbell, G. A. H. Walker, and S. Yang. A Search for Substellar Companions to Solar-type Stars. , 331:902, Aug. 1988.

[5] R. Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.

[6] Chauvin, G., Lagrange, A.-M., Dumas, C., Zuckerman, B., Mouillet, D., Song, I., Beuzit, J.-L., and Lowrance, P. A giant planet candidate near a young brown dwarf* - direct vlt/naco observations using ir wavefront sensing. *AA*, 425(2):L29–L32, 2004.

[7] S. H. Dole. Computer simulation of the formation of planetary systems. *Icarus*, 13(3):494–508, 1970.

[8] P. Dowler, G. Rixon, and D. Tody. Table Access Protocol Version 1.0. IVOA Recommendation 27 March 2010, Mar. 2010.

[9] L. R. Doyle, J. A. Carter, and D. C. Fabrycky. Kepler-16: A transiting circumbinary planet. *Science*, 333(6049):1602–1606, 2011.

[10] D. Foreman-Mackey, R. Luger, E. Agol, T. Barclay, L. Bouma, T. Brandt, I. Czekala, T. David, J. Dong, E. Gilbert, et al. exoplanet: gradient-based probabilistic

inference for exoplanet data & other astronomical time series. *Journal of Open Source Software*, 2021.

[11] Gaia Collaboration and Gaia Data Processing and Analysis Consortium (DPAC). Gaia Data Release 3 (Gaia DR3). Published by ESA on 13 June 2022.

[12] Y. Gao, R. Wang, and E. Zhou. Stock prediction based on optimized lstm and gru models. *Scientific Programming*, 2021(1):4055281, 2021.

[13] A. Ginsburg, B. M. Sipőcz, C. E. Brasseur, P. S. Cowperthwaite, M. W. Craig, C. Deil, J. Guillochon, G. Guzman, S. Liedtke, P. Lian Lim, K. E. Lockhart, M. Mommert, B. M. Morris, H. Norman, M. Parikh, M. V. Persson, T. P. Robitaille, J.-C. Segovia, L. P. Singer, E. J. Tollerud, M. de Val-Borro, I. Valtchanov, J. Woillez, The Astroquery collaboration, and a subset of the astropy collaboration. astroquery: An Astronomical Web-querying Package in Python. , 157:98, Mar. 2019.

[14] N. M. Guerrero, S. Seager, C. X. Huang, A. Vanderburg, A. G. Soto, I. Mireles, K. Hesse, and et al. The tess objects of interest catalog from the tess prime mission. *The Astrophysical Journal Supplement Series*, 254(2):39, jun 2021. Dataset: https://exoplanetarchive.ipac.caltech.edu/docs/API_TOI_columns.html (Accessed: 2024-09-03).

[15] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[16] A. P. Hatzes, W. D. Cochran, M. Endl, B. McArthur, D. B. Paulson, G. A. H. Walker, B. Campbell, and S. Yang. A Planetary Companion to γ Cephei A. *The Astrophysical Journal*, 599(2):1383, dec 2003.

[17] M. Hippke, T. J. David, G. D. Mulders, and R. Heller. Wotan: Comprehensive time-series detrending in python. *The Astronomical Journal*, 158(4):143, sep 2019.

[18] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, 1998.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[20] N. Imara and R. D. Stefano. Searching for exoplanets around x-ray binaries with accreting white dwarfs, neutron stars, and black holes. *The Astrophysical Journal*, 859(1):40, may 2018.

[21] J. M. Jenkins, J. D. Twicken, S. McCauliff, J. Campbell, D. Sanderfer, D. Lung, M. Mansouri-Samani, F. Girouard, and et al. The TESS science processing operations center. In G. Chiozzi and J. C. Guzman, editors, *Software and Cyberinfrastructure for Astronomy IV*, volume 9913, page 99133E. International Society for Optics and Photonics, SPIE, 2016.

[22] N. M. Kostogryz, A. I. Shapiro, and V. Witzke. Magnetic origin of the discrepancy between stellar limb-darkening models and observations. *Nature Astronomy*, 8(7):929–937, 2024.

[23] L. Kreidberg. batman: BAsic Transit Model cAlculatioN in Python. , 127(957):1161, Nov. 2015.

[24] S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, University of Helsinki, 1970.

[25] I. Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[26] Y. Lu, M. Sarkis, and G. Lu. Multi-task learning for single image depth estimation and segmentation based on unsupervised network. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10788–10794, 2020.

[27] S. Mahjoub, L. Chrifi-Alaoui, B. Marhic, and L. Delahoche. Predicting energy consumption using lstm, multi-layer gru and drop-gru neural networks. *Sensors*, 22(11):4062, 2022.

[28] K. Mandel and E. Agol. Analytic Light Curves for Planetary Transit Searches. , 580(2):L171–L175, Dec. 2002.

[29] D. Mateos-GarcÃa, J. GarcÃa-GutiÃ©rrez, and J. C. Riquelme-Santos. On the evolutionary weighting of neighbours and features in the k-nearest neighbour rule. *Neurocomputing*, 326-327:54–60, 2019.

[30] P. Mróz, R. Poleski, A. Gould, A. Udalski, and T. Sumi. A terrestrial-mass rogue planet candidate detected in the shortest-timescale microlensing event. *The Astrophysical Journal Letters*, 903(1):L11, 2020.

[31] NASA Exoplanet Archive. Planetary systems, 2024.

[32] NASA High Energy Astrophysics Mission Data. Target pixel file tutorial for tess data, 2024. Accessed: 2024-09-02.

[33] A. E. P. A. Odunlade. *Transiting exoplanets: characterisation in the presence of stellar activity*. University of Exeter (United Kingdom), 2010.

[34] M. Paegert, K. G. Stassun, K. A. Collins, J. Pepper, G. Torres, J. Jenkins, J. D. Twicken, and D. W. Latham. Tess input catalog versions 8.1 and 8.2: Phantoms in the 8.0 catalog and how to handle them, 2021.

[35] J.-S. Pan, Y.-S. Ting, and J. Yu. Astroconformer: The prospects of analysing stellar light curves with transformer-based deep learning models. *Monthly Notices of the Royal Astronomical Society*, 528(4):5890–5903, 01 2024.

[36] S. J. Prince. *Understanding Deep Learning*. The MIT Press, 2023.

[37] G. R. Ricker, J. N. Winn, R. Vanderspek, D. W. Latham, G. Á. Bakos, J. L. Bean, Z. K. Berta-Thompson, T. M. Brown, and et al. Transiting exoplanet survey satellite. *Journal of Astronomical Telescopes, Instruments, and Systems*, 1(1):014003, 10 2014.

[38] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[39] N. C. Santos, M. Mayor, D. Naef, F. Pepe, D. Queloz, S. Udry, and M. Burnet. The CORALIE survey for southern extra-solar planets VI. New long period giant planets around HD 28185 and HD 213240. , 379:999–1004, Dec. 2001.

[40] M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[41] S. Seager and G. Mallén-Ornelas. A Unique Solution of Planet and Star Parameters from an Extrasolar Planet Transit Light Curve. , 585(2):1038–1055, Mar. 2003.

[42] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[43] STScI. TESS Input Catalog and Candidate Target List, 2022.

[44] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[45] Tess candidate exoplanets, 2024. https://exofop.ipac.caltech.edu/tess/view_toi.php.

[46] A. Tsiaras, I. Waldmann, M. Rocchetto, R. Varley, G. Morello, M. Damiano, and G. Tinetti. pylightcurve: Exoplanet lightcurve model. *Astrophysics Source Code Library*, pages ascl–1612, 2016.

[47] J. W. Tukey. Exploratory data analysis. *Reading/Addison-Wesley*, 1977.

[48] J. T. VanderPlas. Understanding the lomb–scargle periodogram. *The Astrophysical Journal Supplement Series*, 236(1):16, 2018.

[49] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[50] D. P. Whitmire and R. T. Reynolds. Circumstellar Habitable Zones: Astronomical Considerations. In L. R. Doyle, editor, *Circumstellar Habitable Zones*, page 117, Jan. 1996.

[51] A. Wolszczan and D. A. Frail. A planetary system around the millisecond pulsar psr1257 + 12. *Nature*, 355(6356):145–147, 1992.

[52] L. Ziyin, T. Hartwig, and M. Ueda. Neural networks fail to learn periodic functions and how to fix it. 06 2020.

A. Training details

A.1 Data Generation Parameters

Planet Type	Radius Range (Earth Radii)	Density (g/cm^3)
Earth	0.7 – 1.2	5.0
Super Earth	1.2 – 2.5	5.0
Mini Neptune	2.5 – 5.0	5.0
Neptune	5.0 – 7.0	1.0
Jupiter	7.0 – 13.0	1.0

Table A.1: Radius ranges and density values for each planet type.

Planetary Parameter	Generating Distribution	Distribution Parameters
Inclination	Constant	90
Longitude of Periastron	Constant	90
Eccentricity	Gamma[a]	$a = 0.25312$, loc $= 0.001$, scale $= 0.25767$, max $= 0.8$
Limb Darkening u_1	Normal[b]	loc $= 0.5$, scale $= 0.05$
Limb Darkening u_2	Normal[b]	loc $= 0.1$, scale $= 0.01$

[a] scipy.stats.gamma.rvs().

[b] numpy.random.normal().

Table A.2: Planetary parameters and their generating distributions.

The `batman` package provides several pre-defined models for computing limb darkening. I have chosen the simple quadratic limb darkening formula, given by

$$I(\mu) = I_0 \left[1 - u_1(1-\mu) - u_2(1-\mu)^2\right], \tag{A.1}$$

where $\mu = \sqrt{1-x^2}$, $0 \leq x \leq 1$ is the normalized radial coordinate, and I_0 is a normalization constant ensuring that the integrated stellar intensity equals unity.[†]

[†] Source: `batman` documentation

Distance Category	Range of $a/R_\star$
Very Near	10 – 50
Mid	100 – 320
Far	2000 – 6000

Table A.3: Distance categories used for generating the semimajor axis to stellar radius ratio ($a/R_\star$) parameter, used to determine the orbital distances for synthetic transits generated using `batman`. The values are sampled from a uniform distribution within the specified ranges.

where $\mu = \sqrt{1-x^2}$, $0 \leq x \leq 1$ is the normalized radial coordinate, and I_0 is a normalization constant ensuring that the integrated stellar intensity equals unity.*

A.2 Dataset Visualization

This section illustrates three plots, each displaying four training examples from the Jupiter, Neptune, and Mini-Neptune datasets, respectively, for Lv2 and Lv3 models. The plots represent input light curves injected with the corresponding generated transits. While the transit midpoints and periods remain consistent across all planet types, the transit characteristics differ due to variations in the relative sizes of the planets to their stars. The data is normalized to ensure that both time and flux have a mean of 0 and unit variance, which improves the numerical stability of the model and accelerates convergence during training. Furthermore, the flux values are inverted to assist the MaxPool component of the convolutional layers in detecting transit dips, which are transformed into peaks after the inversion for easier recognition.

In the plots, the dotted vertical line indicates the transit midpoint, the red highlighted regions correspond to the transits, and the grey areas represent the padded data added to meet the required sequence length of 2^{14}.

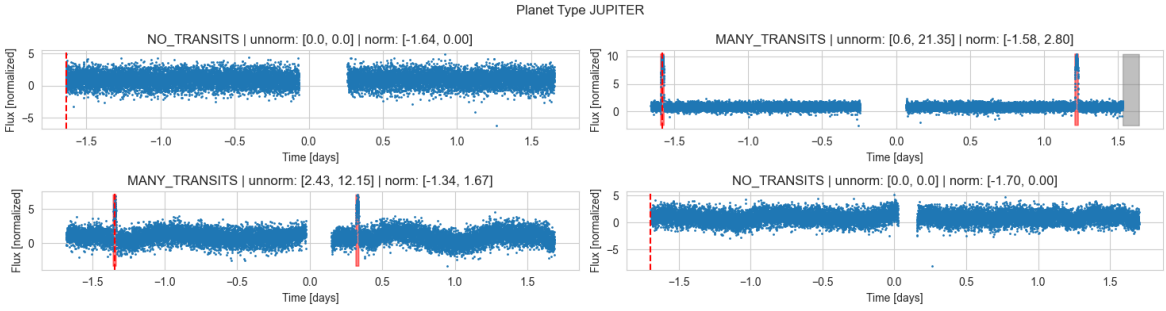


Figure A.1: Visualization of four training examples from the Jupiter dataset.

*Source: `batman` documentation

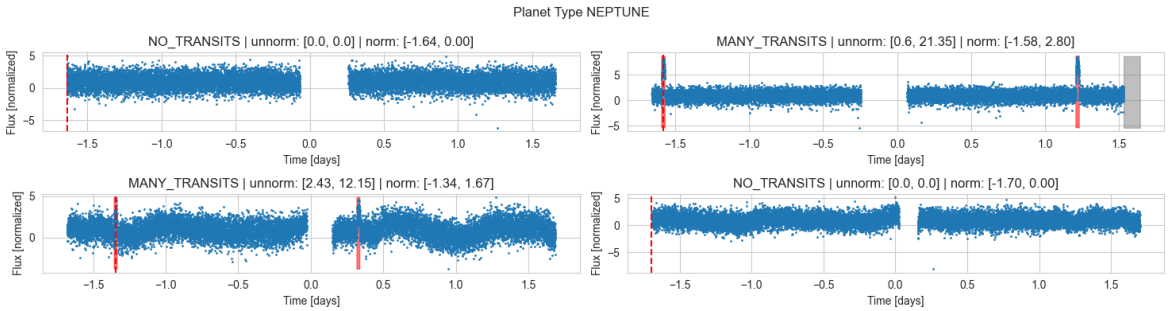


Figure A.2: Visualization of four training examples from the Neptune dataset.

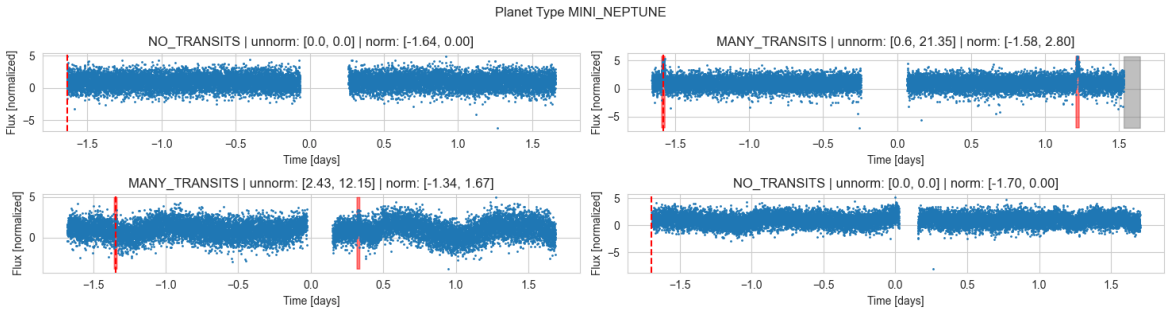


Figure A.3: Visualization of four training examples from the Mini-Neptune dataset.

A.3 Model Hyperparameters

This section reports the hyperparameters used for training each model level for each planet type. The values for learning rate and weight decay are presented up to six significant digits.

Lv0

Hyperparameter	Jupiter	Neptune	Mini-Neptune
Learning Rate	0.002310	0.001695	0.001893
Weight Decay	0.000339	0.000069	0.000008
Encoder Hidden Layers	2	3	3
Decoder Hidden Layers	2	4	3

Table A.4: Hyperparameters used for the Lv0 model for different planet types.

Lv1

Hyperparameter	Jupiter	Neptune	Mini-Neptune
Learning Rate	0.000176	0.000101	0.000022
Weight Decay	0.000071	0.000002	0.000139
Decoder Hidden Layers	3	2	2
Convolutional Layers	3	5	5
Kernel Size	97	33	31

Table A.5: Hyperparameters used for the Lv1 model for different planet types.

Lv2

Hyperparameter	Jupiter	Neptune	Mini-Neptune
Learning Rate	0.006406	0.002195	0.001411
Weight Decay	0.000036	0.000002	0.000558
Patch Length	32	16	128
Patch Embedder Latent Dim	8	4	6
Convolutional Layers	1	3	2
Kernel Size	15	17	33
Flattener Hidden Layers	1	2	2
Detector Hidden Layers	3	3	2
Estimator Hidden Layers	3	2	3

Table A.6: Hyperparameters used for the Lv2 model for different planet types.

Lv3

Hyperparameter	Jupiter	Neptune	Mini-Neptune
Learning Rate	0.001342	0.001798	0.000186
Weight Decay	0.000581	0.000697	0.000043
Patch Length	64	32	32
Time Embedder Latent Dim	16	16	16
Patch Embedder Latent Dim	16	16	16
Patch Embedder Hidden Layers	2	2	2
Kernel Size	601	751	437
Conformer Blocks	4	4	4
Attention Heads	8	8	8
Detector Hidden Layers	2	2	2
Estimator Hidden Layers	2	2	2

Table A.7: Hyperparameters used for the Lv3 model for different planet types.

A.4 System information

Component	Specification
CPU	Intel Core i7-7700K @ 4.20 GHz
GPU	Nvidia GeForce GTX 1080
Ram	16 GB
Operating System	Linux Mint 21.3
Python Version	3.12.4
PyTorch and Lightning Version	2.3.1

Table A.8: System information used for training the models.

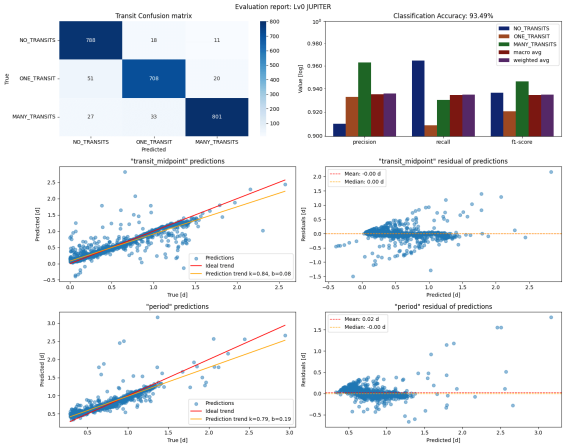
B. Detailed Evaluation Results

This section evaluates the models' performance by planet type, presenting results for three tasks: multiclass detection, transit midpoint regression, and orbital period regression. Each figure consists of six subplots for detailed analysis.

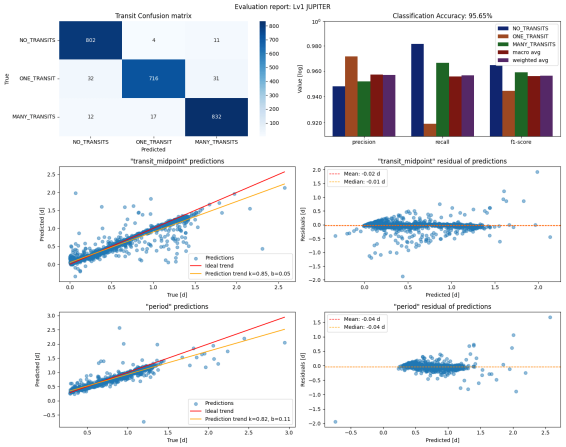
The first row focuses on the multiclass detection task, including a confusion matrix summarizing classification performance and precision, recall, and F1 scores for each label. The second and third rows assess the regression tasks. For each, the left subplot shows a scatter plot comparing predictions to ground truth. Ideally, points should follow a perfectly linear trend, indicating accurate predictions. Deviations from this trend reveal systematic biases, providing insights for future model improvements.

The right subplot in each row presents the residuals, offering a detailed view of the error distribution. These are plotted against the **predicted values** rather than the true values: while plotting against the true values might seem more intuitive, it would convey the same information as the left plot, just rotated 45°, making it redundant. Plotting against the predicted values instead highlights systematic biases and error variability in the model's outputs, providing a more relevant assessment of performance. Together, these plots help evaluate the magnitude and variability of errors, essential for determining the models' applicability in real-world scenarios.

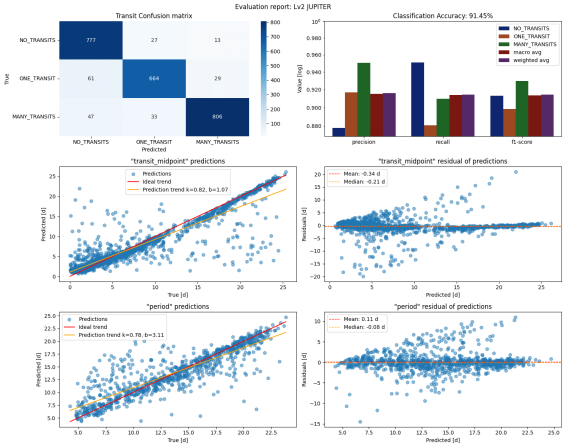
B.1 Jupiter Models



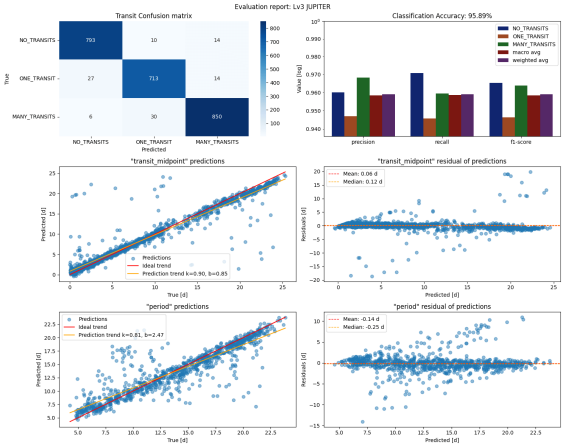
(a) Level 0 Jupiter Model



(b) Level 1 Jupiter Model



(c) Level 2 Jupiter Model

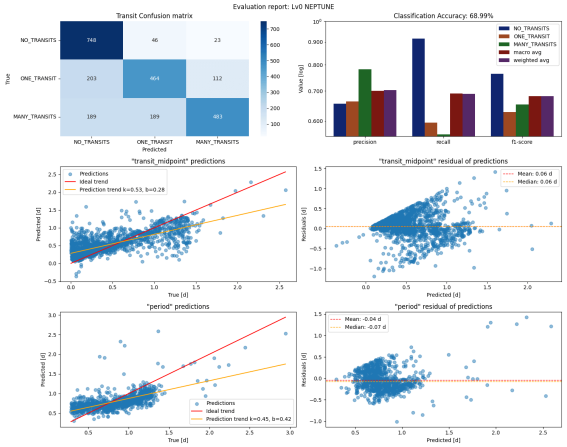


(d) Level 3 Jupiter Model

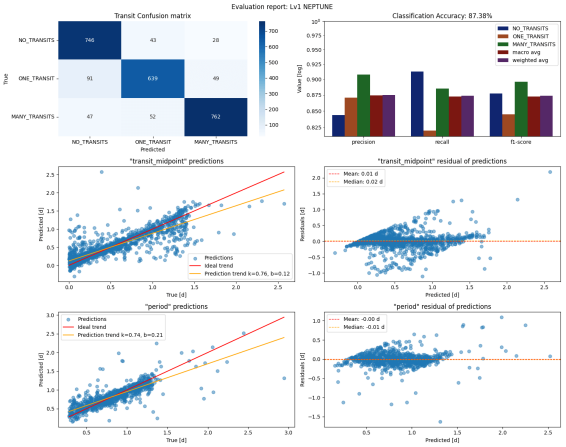
Figure B.1: Evaluation results for Jupiter models.

Jupiter models demonstrates the best performance due to the deeper and more distinct transit signals, making detection more reliable. Notably, the Lv3 transit midpoint regressor achieves the closest alignment with the ideal trend.

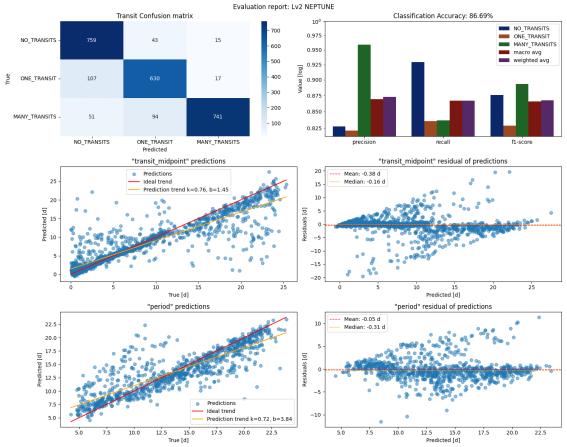
B.2 Neptune Models



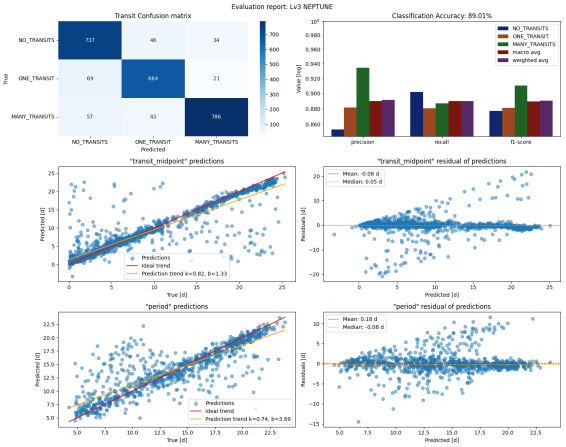
(a) Level 0 Neptune Model



(b) Level 1 Neptune Model



(c) Level 2 Neptune Model

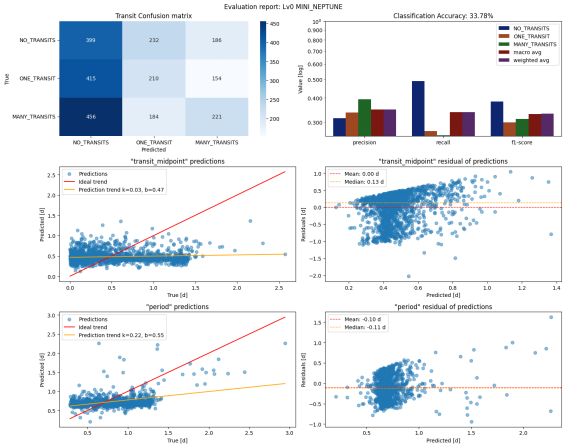


(d) Level 3 Neptune Model

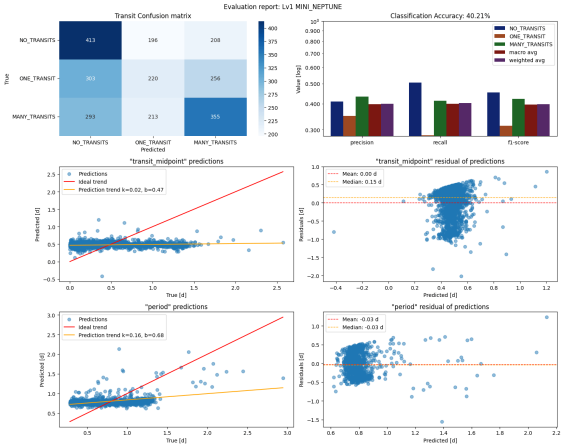
Figure B.2: Evaluation results for Neptune models.

For Neptune models, Lv0 and Lv1 perform below expectations compared to their Jupiter counterparts. However, Lv2 and Lv3 perform well in the multicalss detection task, as indicated by the results in the confusion matrix. Unlike the Jupiter models, there is a more noticeable tendency to assign the "no transit" label to data containing transits. The regressors prediction trends align well with the ground truth, despite noisier predictions as shown in the residual scatter plots. These residuals indicate increased variability, but overall accuracy remains promising.

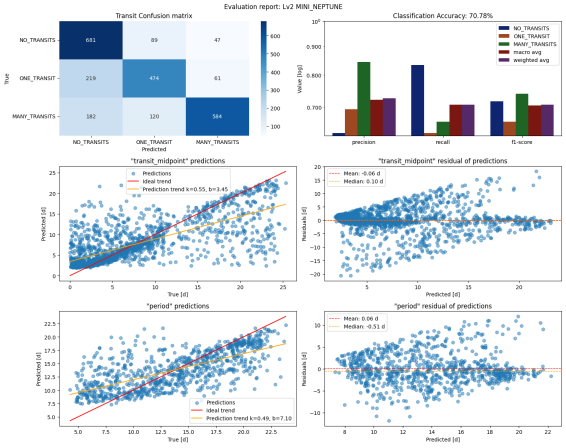
B.3 Mini-Neptune Models



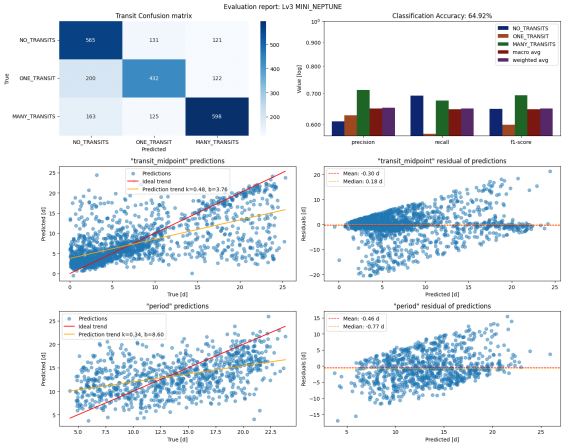
(a) Level 0 Mini-Neptune Model



(b) Level 1 Mini-Neptune Model



(c) Level 2 Mini-Neptune Model



(d) Level 3 Mini-Neptune Model

Figure B.3: Evaluation results for Mini-Neptune models.

For the Mini-Neptune models, Lv0 and Lv1 fail to generate meaningful predictions. Their confusion matrices lack clear patterns, and their regression scatter plots show trends that deviate significantly from the ideal. In contrast, Lv2 and Lv3 show more promise, with Lv2 demonstrating some ability in the classification task. However, their scatter plots reveal noisy predictions and significant variability, making them unsuitable for application.