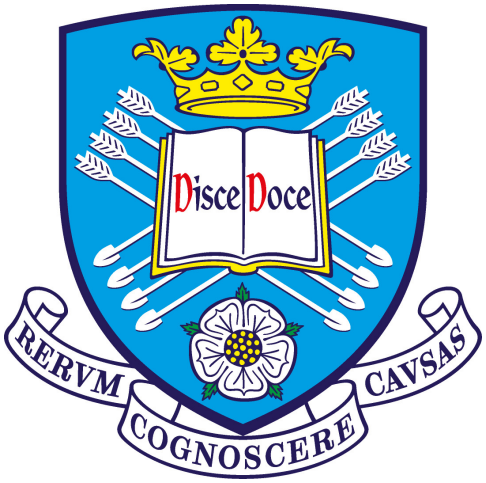
Automated classification of transients in optical time-domain sky surveys

Umar Farouq Burhanudin

Department of Physics & Astronomy
The University of Sheffield



A dissertation submitted in candidature for the degree of Doctor of Philosophy at the University of Sheffield

April 2022

“

The good news about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do.

”

— Ted Nelson

Contents

List of Figures

List of Tables

Declaration

I declare that, unless otherwise stated, the work presented in this thesis is my own. No part of this thesis has been accepted or is currently being submitted for any other qualification at the University of Sheffield or elsewhere.

The work presented in Chapter 4 of this thesis has already been published and can be found in Burhanudin et al. (2021).

Acknowledgments

Firstly, I would like to thank my supervisor Justyn for taking me on to do this PhD. I have learnt a great deal from you, both in scientific know-how and also how to be a good researcher. You have been supportive and encouraging in what has been one the most challenging but also rewarding experiences in my life.

These three and a half years have not been with its ups and downs. Thank you to my postgraduate tutor Jenny Clark and Stu for chatting with me when I was struggling, and assuring me that sometimes being stuck and feeling like you're not getting anywhere is perfectly fine. I guess impostor syndrome is just part of the PhD package and that you just learn to live with it, and eventually realise that you are smart enough to do a PhD!

Most, if not all the work during my PhD could not have been completed without the help of Paul Kerry, the computer wizard. Thank you for installing countless Python packages for me and making sure that they were working properly. I hope I did not cause you too many headaches.

I came into this PhD not knowing anything about machine learning, and now I am proud to say I know some things. Thank you to Maurico and Fariba from the Computer Science department, and all the clever people I met at the Artificial Intelligence in Astronomy conference at ESO for the informative conversations. The best way to learn something you don't know a lot about is to learn from experts.

This PhD has not been an entirely solitary experience (apart from the year or so due to a global historical happening). Thank you to the current and past inhabitants of the E18 PhD office for our shared commiseration - things will be fine. I must also thank my friends from the 'Union' for making working from home a much more fun experience.

A big thank you to my family for your continued support. Ummi, Ayah, Man and Nui, I cannot thank you enough for the love and support you have shown me throughout my PhD. I love, respect, and admire each one of you. I hope I have made you proud to call myself your son and brother.

Last but not least, thank you to my wife Emily. Without you I would not be where I am today. You believed in me when I didn't believe in myself, and helped me to find the strength to keep going when I thought doing a PhD was not the path for me. For that, you forever have my gratitude. None of this would have been possible if it were not for you. I will love you always.

Summary

Repeated sky surveys in the past decade have led to the proliferation in the discovery of transients. This has not come without its own challenges: the rate of discovery from current sky surveys greatly exceeds the human capacity to manually identify and classify newly discovered objects. The use of machine learning approaches to automate the discovery process with little to no human intervention is rapidly becoming a standard practice in surveys. In this thesis we present the use of machine learning to classify objects discovered by sky surveys observing at optical wavelengths.

The Gravitational-wave Optical Transient Observer (GOTO) is a survey with the aim of searching for the optical counterparts to gravitational waves, while also scanning the night sky for transients and variable objects. We use both a machine learning and a deep learning approach to classify objects observed by GOTO using their light curves, and compare the effectiveness and limitations of both methods for photometric classification. We find that using a deep learning approach with recurrent neural networks works best to reliably classify objects using their light curves in real-time.

We investigate the use of Gaussian processes to create uniform representations of supernova light curves from different surveys. These are then used with a convolutional neural network for classification into supernova sub-types. Future surveys will have a lack of labelled data to train classifiers. We use transfer learning to show how data from another survey can be used to train a classifier for a new survey.

Machine learning is a widely used methodology, and has uses in other fields of research. We show how the classifiers developed for GOTO light curve classification can be adapted for other classification tasks, and how they perform on these tasks.

Chapter 1

Introduction

Time-domain astronomy concerns the observation and study of sources that have some variability in time, typically on timescales much shorter than typical astrophysical or cosmological timescales such as galaxy or stellar evolution. The discovery of time-varying sources are often associated with an observed change in brightness or flux. The most widely observed time-varying sources in the optical wavelengths ($\sim 380\text{nm} - 700\text{nm}$) are variable stars and transient events. Transient events (commonly referred to as transients) are some of the most extreme and violent astrophysical phenomena to occur in the universe, and observing them has augmented efforts to understand the universe. The discovery of the period-luminosity relation of Cepheid variables in the Small Magellanic Cloud enabled the use of Cepheid variables as a standard candle to measure extra-galactic distances (Leavitt & Pickering, 1912). Thermonuclear supernovae have been used to reveal an accelerated expanding Universe (Perlmutter et al., 1999; Riess et al., 1998) and continue to be a powerful probe of dark energy (e.g. Betoule et al., 2014). Detections of gravitational waves produced by the merger of two black holes have confirmed one of the predictions of general relativity and marks a new age of multi-messenger astronomy (Abbott et al., 2016). The study of core-collapse supernovae has shed light on the end-points of stellar evolution for massive stars, and the progenitor systems that give rise to these catastrophic explosions (e.g. Smartt 2009).

Discovering transients requires repeated observations of the night sky over the period of days to weeks. Observing transients can be a challenge due to the fact that they are 'one-off' events, and discovering them in the past was a serendipitous process. However, the emergence of synoptic all-sky surveys such as the Palomar Transient Factory (PTF) (Rau et al., 2009), Pan-STARRS (Kaiser et al., 2010), and the Catalina Real-Time Transient Survey (CRTS) (Drake et al., 2009) have made these once rare events more common. Current transient surveys now face a new challenge: the large volume of data collected exceeds the available human resources to analyse them. This problem highlights the need for an automated framework, based around a machine-learning approach where

the discovery and identification of new transients is executed without (or with very little) human supervision.

1.1 Explosive transients

Transients are markedly different from other typically observed objects such as stars and galaxies, in terms of their observed properties and the physical mechanisms that power them. They are different to variable objects, as transients are 'one-off' events that can increase in brightness by several magnitudes and then fade away. They are host to some of the most extreme environments where physics occurs, and some transients represent the endpoints of stellar evolution. Transients are visible on timescales ranging from minutes to months, and their transient nature arise from some physical process of the progenitor system. Photometric studies of transients utilise their *light curves*, obtained by measuring their flux (and usually converting them to magnitudes) over time at regular intervals. The different types of transients discussed in this section can be visualized in a transient phase space, shown in Fig. 1.1. The transient phase space visualises the diversity of observed transients, showing the range of luminosities and timescales over which they are visible.

1.1.1 Novae eruptions

Classical novae are the result of thermonuclear burning of hydrogen-rich material accreted onto the surface of a white dwarf (WD) star in a close binary (see Starrfield et al., 2016 for a detailed review). Hydrogen-rich material is accreted onto the surface of the WD in a close binary system from a companion star. Compression of the layer by the surface gravity of the WD causes it to become electron degenerate. Once temperatures on the burning surface are high enough, a thermonuclear runaway process ignites the hydrogen-rich layer causing an outburst.

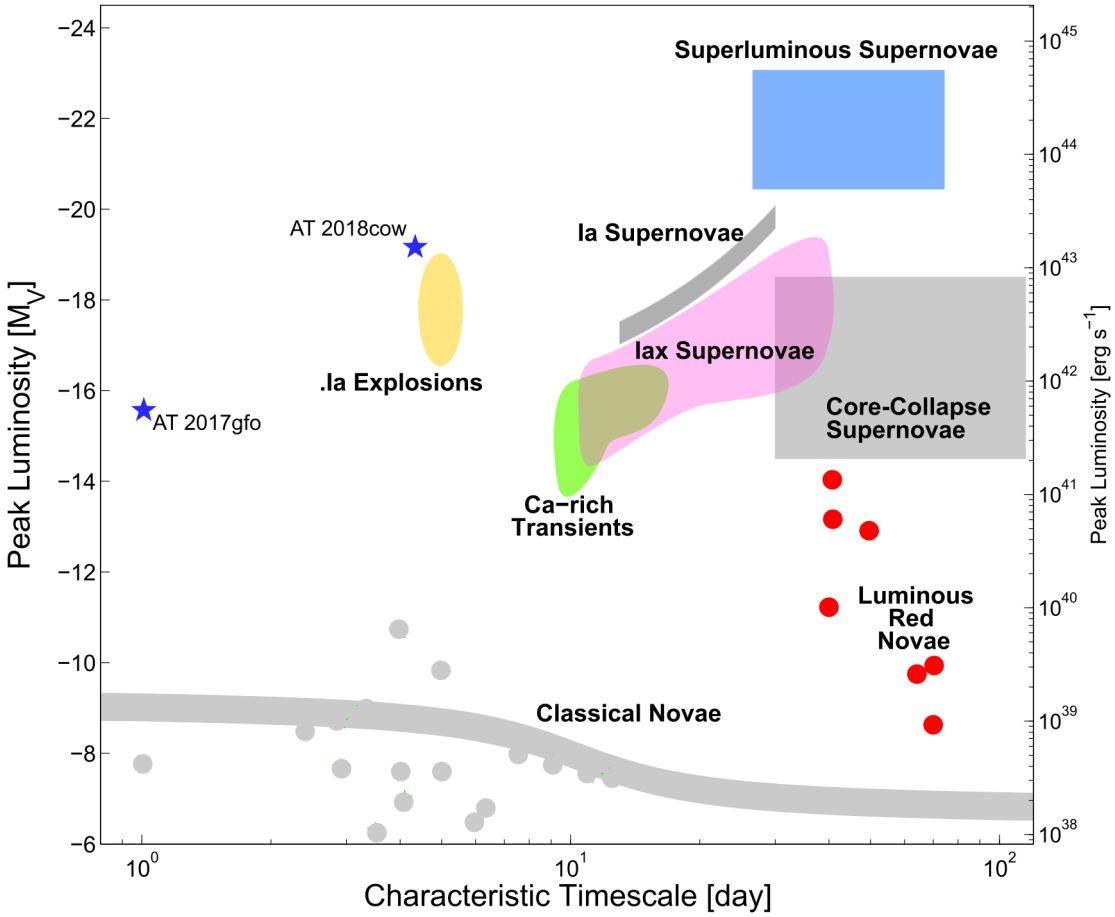


Figure 1.1: Transient phase space diagram, adapted from Kasliwal (2011) and updated with data from Foley et al. (2013) for SNe Iax, Valenti et al. (2017) for AT 2017gfo, the kilonova associated with gravitational wave detections from a binary neutron star merger, and Perley et al. (2019) for AT 2018cow. Peak luminosity (in the V band) is plotted against characteristic timescale (the time taken to decline in brightness by ~ 1 mag from peak).

Material produced by the hydrogen-burning nucleosynthesis of classical novae explosions are injected into the interstellar medium (Gehrz et al., 1998). Classical novae reach peak absolute magnitudes of $M_V \sim -7.5$ with decay times (the time taken for magnitude to become fainter by 2-3) ranging from days to hundreds of days; peak absolute magnitude and decay time are related, with brighter novae having shorter decay times (Hernanz, 2005, and references therein). This relationship can be seen for classical novae in Figure 1.1

1.1.2 Luminous red novae

Luminous red novae are transients with peak absolute magnitudes $M_R \sim -12$ (brighter than novae but fainter than ordinary supernovae) with red colors ($g - r \sim 1$) (Kulkarni et al., 2007; Kasliwal et al., 2011), and occupy the 'luminosity gap' between classical novae and supernovae (see Fig 1.1). Luminous red novae have double-peaked light curves (an initial rapid blue peak followed by a longer duration red peak), and are proposed to arise from massive binaries or the common envelope ejection of a binary system (Pastorello et al., 2019).

1.1.3 Supernovae

Supernovae (SNe) mark the end points of stellar evolution for massive stars and WD in binary systems. Traditionally, SNe have been divided into different classes based on observable features in their optical spectra (see Filippenko, 1997, for a review on SNe taxonomy). The standard SNe classification scheme is summarised in Fig. 1.2.

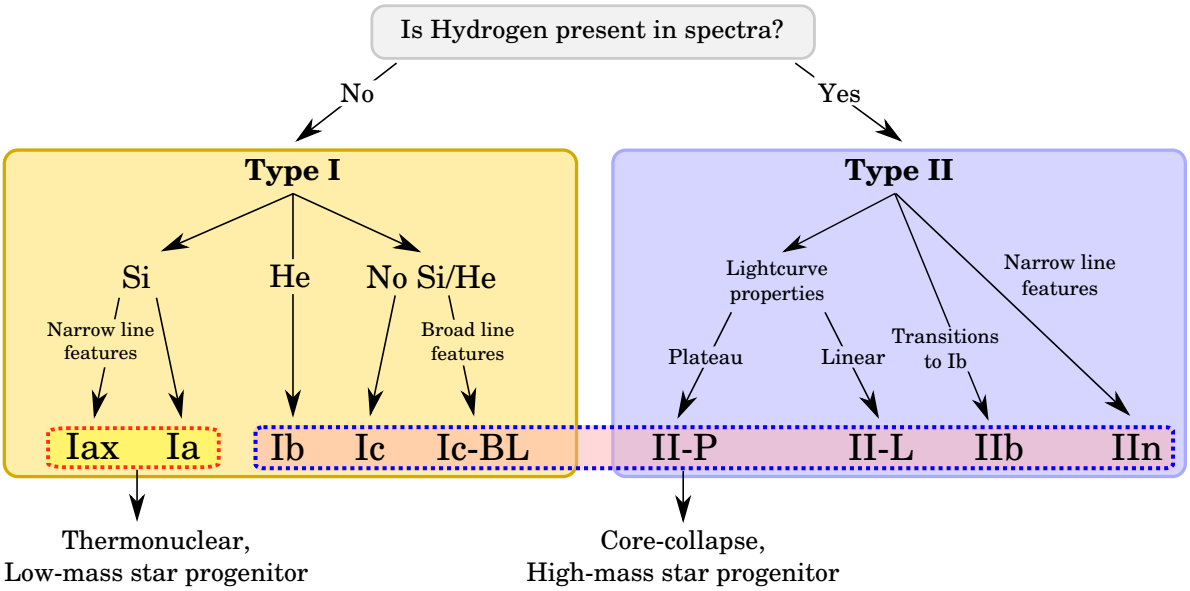


Figure 1.2: Supernovae classification diagram.

Thermonuclear supernovae

Type Ia SNe (SNe Ia) are identified by the lack of hydrogen in their optical spectra, but with observed Si II lines (Filippenko, 1997).

They are thought to originate from the runaway thermonuclear burning of a degenerate carbon-oxygen core of a WD into ^{56}Ni once the WD approaches a mass comparable to the Chandrasekhar mass ($M_{\rm Ch} \sim 1.4\ M_{\odot}$), which is sufficient to unbind the WD. Whether the progenitor consists of a single-degenerate (SD) or double-degenerate (DD) binary system is still debatable (Maoz et al., 2014).

SNe Ia reach peak absolute magnitudes of $M_V \sim -19$ (Hamuy et al., 1996), and there is a correlation between the peak brightness and light curve decline rate (Phillips, 1993), known as the *width-luminosity* relation. The width-luminosity relation for SNe Ia can be seen in Figure 1.1. Using this correlation, SNe Ia can be used as calibrated standard candles for cosmology (e.g. Perlmutter et al., 1999; Riess et al., 1998).

Type Iax supernovae (SNe Iax) form a distinct class of SNe different to SNe Ia, and have spectral features that are similar to SN 2002cx (Li et al., 2003) at similar epochs. They have lower peak luminosities than SNe Ia, correlated to the shape of the light curve similar to the width-luminosity relation of SNe Ia but with a larger scatter. The spectra of SNe Iax at peak brightness show lower ejecta velocities than that of SNe Ia at peak brightness. For a review of observational properties of SNe Ia, see Foley et al. (2013). The width-luminosity relation for SNe Iax can be seen in Figure 1.1, showing a similar relation but at lower peak luminosities. The proposed mechanism to produce SNe Iax is the failed deflagration of a carbon-oxygen (CO) WD, where burning within the WD rises to the surface but the entire WD is not consumed by the burning (Foley et al., 2009).

.Ia explosions ("point Ia") produce roughly ~ 10 % of the luminosity of SNe Ia for roughly ~ 10 % of the typical SN Ia timescale (Kasliwal et al., 2010; Poznanski et al., 2010). They are thought to arise from the thermonuclear explosion of helium accreted

onto a WD in a binary system (Poznanski et al., 2010).

Core-collapse supernovae

All other types of SNe apart from SNe Ia are expected to arise from the core-collapse of massive stars ($> 8\ M_{\odot}$). The resulting explosion releases heavy elements created via nucleosynthesis during the lifetime of the star; the high energies released from these explosions provide feedback for star formation within galaxies, and the end result of core-collapse SNe (CCSNe) leaves behind compact objects such as neutron stars and black holes (Maund et al., 2017, and references therein). Core-collapse supernovae show a wide range of observational properties: there is a large variation in peak luminosities and magnitude decay timescales, which is visualised in Figure 1.1. Classification of core-collapse is typically done based on spectral features.

Type Ib SNe (SNe Ib) show He I lines in their spectra, while Type Ic SNe (SNe Ic) show neither He nor Si absorption lines in their spectra (Filippenko, 1997). The progenitors of these SNe I are thought to be hot Wolf-Rayet (WR) stars or low mass He and CO stars in a binary system (Maund & Smartt, 2005). SNe Ic with broad line spectral features (SNe Ic-BL) have been observed and some have been associated with gamma-ray bursts (GRBs) (Valenti et al., 2008; Galama et al., 1998).

Type II SNe (SNe II) are those with hydrogen observed in their spectra, with further subclassifications made on the basis of light curve or spectral properties: SNe II-P feature a plateau in their light curves while SNe II-L light curves show a linear decline (Filippenko, 1997). The progenitors of SNe II are red supergiants, and different processes throughout stellar evolution (e.g. loss of a hydrogen envelope through mass-loss processes) dictate the type of explosion that occurs (Smartt, 2009).

Observations of transients with peak absolute magnitudes between $-15.5 \geq M_R \geq -16.5$ with late-time spectra dominated by calcium fall into a distinct class of calcium-rich transients (Kasliwal et al., 2012). These transients rise to brightness and transition

into the nebular phase faster than ordinary SNe, with rise times of ∼12-15 days and showing spectroscopic evolution into the nebular phase within ∼1-3 months. Calcium-rich transients are located in the outskirts of their host galaxies ($\geq$ 30 kpc from the centre).

Superluminous supernovae

The discovery of a supernova explosion with a peak absolute magnitude of $M \sim -22$ suggested the existence of a new class of rare and superluminous supernovae (SLSNe) (Quimby et al., 2007). The high luminosities and observed physical properties of SLSNe suggest they have different progenitors to ordinary SNe, though the exact nature of the powering mechanism remains an open question (Gal-Yam, 2012).

SLSNe can be further classified into two subclasses based on their spectra: Type I SLSNe (SLSN-I) are hydrogen poor and Type II SLSNe (SLSN-II) are hydrogen rich. The light curves of SLSNe typically have longer timescales compared to normal supernovae, both in rise and decline. There are a number of proposed powering mechanisms for such high luminosity events, such as magnetars, black-hole accretion, radioactive nickel decay, and circumstellar material interaction (see Gal-Yam 2019 for a review of superluminous supernovae).

1.1.4 Fast blue optical transients

The rise of time-domain surveys that are capable of observing large patches of sky at higher cadences have led to the discovery of a new class of luminous extragalactic transients that evolve on the timescales of days. These transients have luminosities comparable to supernovae but have bluer colours, and have been dubbed 'fast blue optical transients' (FBOTs). One example of this new class of transients is AT 2018cow, with a rapid time to peak luminosity ($M_V \sim -19$) in roughly 5 days (eg. Perley et al. 2019). The power source of AT 2018cow is proposed to be a compact object such as a

magnetar or a black hole remnant resulting from the failed explosion of a blue supergiant (Margutti et al., 2019), though the exact nature or origin of AT 2018cow is still unclear.

1.1.5 Gravitational Waves & Kilonovae

The detection of gravitational waves from a binary black hole merger confirmed one of the predictions of general relativity; the inspiral and merger of two black holes radiate energy in the form of gravitational waves travelling at the speed of light (Abbott et al., 2016). The subsequent detection of gravitational waves from a binary neutron star (NS) merger prompted a search for an electromagnetic counterpart in the same region on the sky, and a fast-fading transient thought to be a kilonova was discovered (Smartt et al., 2017). The kilonova associated with the binary NS gravitational wave had an absolute magnitude $M_r = -15.8 \pm 0.1$ mags, and dimmed by about $\sim$ 1 mag after $\sim$ 1 day (Valenti et al., 2017).

Kilonovae are the proposed optical transients that accompany a binary NS merger, reaching peak luminosities of $\sim 10^{41}$ ergs s^{-1}, a factor of 1000 larger than novae (Metzger et al., 2010). An ultra-stripped supernova, where a massive star stripped of its outer hydrogen and helium envelopes and undergoes core-collapse, are thought to be the progenitors of kilonova systems (De et al., 2018).

1.2 Studying variability

Studying the dynamic nature of transients is primarily done by observing their evolution through time. This can be done by treating the transient as a point source and measuring their brightness as a function of time (photometry), or by observing the spectra of the transient to identify the presence of chemical elements and measure the velocity of moving ejecta (spectroscopy).

Exploration of the transient sky has seen great progress through the efforts of untar-

geted synoptic surveys, with current surveys such as the Zwicky Transient Facility (ZTF) capable of covering the entire Northern sky roughly every three days (Bellm et al., 2019). Determining variability from survey images is done through either *catalogue searching* or *difference imaging* (Bloom & Richards, 2012). In a catalgoue search, sources in an image with fluxes above a threshold are extracted to a catalogue, and cross-matched with detections from previously catalogued sources to identify time-variability. With difference imaging, a reference image is subtracted from a target image to produce a residual, which indicates that a source is varying in brightness over time. A reference image represents an image of a region of the static sky, and a science image is one that contains a new object that shows a change in brightness in the same region. By subtracting the two images, any non-varying sources are removed from the image and only the new varying object remains, from which the magnitude can be measured. This method can allow the discovery of many transient and variable sources in a single image, provided the astrometric alignment between the two images is accurate. Catalogue searches are generally faster, but difference imaging excels at finding variability in crowded fields.

Once a source has been identified, photometry is carried out to measure the brightness of the source. The photon counts f_X measured from an image of the source taken through a filter X, that covers a particular wavelength range, is converted into an apparent magnitude using the relation

$$m_X = -2.5 \log_{10} f_X + m_X, 0. \tag{1.1}$$

where m_X is referred to as the 'X'-band magnitude. Since the magnitude system is calibrated relative to known stars, a zero-point magnitude $m_{X,0}$ is needed to convert the flux into a magnitude. For the Vega magnitude system, the zero point is defined as the apparent magnitude of the star Vega, which is taken to be 0 in all bands. In the AB

magnitude system, the apparent magnitude in a single filter is

$$m_{\mathrm{AB}} = -2.5 \log_{10} f_\nu + 48.60 \tag{1.2}$$

where the constant is chosen so that $m_{\mathrm{AB}} = m_V$ for an object with a flat spectrum. In practice, $m_{\mathrm{AB}} = m_V$ for a flux f_ν measured at 5480Å in units ergs $\mathrm{cm}^{-2}\mathrm{s}^{-1}\mathrm{Hz}^{-1}$ for an object with a smooth spectrum (Oke & Gunn, 1983).

Photometry of the same source can be done repeatedly over time to give a time-series of flux measurements, producing a light curve. The light curves of different transients will vary in shape and length (according to the typical timescale of the transient), and from the light curves it is possible to infer the physical properties of the source.

1.3 Current and future surveys

In Table 1.1, we provide a summary of past, current, and future time-domain surveys operating in the optical wavelengths. We list the etendue $A\Omega$ of each survey, which is given by the total telescope light collecting area times the field of view of the telescope. For some surveys, there may be multiple telescopes used to produce a combined field of view, such as the All Sky Automated Survey for SuperNovae (ASSASN) and the Gravitational-wave Optical Transient Observer (GOTO). We also discuss the following surveys that are relevant to the work presented in this thesis: the Palomar Transient Factory (PTF), the Zwicky Transient Facility (ZTF), GOTO, and the Vera Rubin Observatory Legacy Survey of Space and Time (LSST).

1.3.1 The Palomar Transient Factory

The Palomar Transient Factory (PTF) was a survey that set out to populate the transient phase space by exploring the transient optical sky. The survey camera was mounted on

Survey	AΩ	Cadence	Δλ	Depth	References
CRTS[1]	∼ 3	6-12 days	300- 000	∼ 20	Drake et al. (2009), Djorgovski et al. (2011)
PTF[2]	8.9	3-5 days	300-700	∼ 21	Rau et al. (2009)
Pan-STARRS 1	50	∼ 7 days	400-1000	∼ 24	Kaiser et al. (2010)
ATLAS[3]	4.1	2 days	420-975	∼ 19	Tonry (2011), Tonry et al. (2018)
La Silla-QUEST	7.8	2 days	300-900	∼ 20	Baltay et al. (2013)
ASSASN[4]	∼ 130	2-3 days	470-700	∼ 17	Shappee et al. (2014), Kochanek et al. (2017)
DES[5]	12	7 days	400-1050	∼ 24	Dark Energy Survey Collaboration (2016)
SkyMapper	7.4	<5 days	300-1100	∼ 21	Scalzo et al. (2017)
HSC[6]	2	4-6 days	400-1100	∼ 26	Aihara et al. (2018)
ZTF[7]	53.1	3 days*	400-900	∼ 21	Bellm et al. (2019)
GOTO[8]	∼ 40**	∼ 3 days	380-700	∼ 20	Steeghs et al. (2021)
Future surveys					
LSST[9]	319	3-4 days*	320-1050	∼ 24	Ivezić et al. (2019)
SiTian	-	30 minutes	300-850	∼ 21	Liu et al. (2021)

[1]Catalina Real Time Survey, [2]Palomar Transient Factory, [3]Asteroid Terrestrial-impact Last Alert System, [4]All Sky Automated Survey for SuperNovae, [5]Dark Energy Survey, [6]HyperSuprimeCam, [7]Zwicky Transient Facility, [8]Graviational-wave Optical Transient Observer, [9]Vera Rubin Observatory Legacy Survey of Space and Time

*These surveys also have higher cadences for a subset observing strategies, for more details see the corresponding references listed in the table.

** Calculated for the current configuration of 8 unit telescopes.

Table 1.1: A list of time-domain sky surveys. For each survey, we provide its etendue AΩ in units of m^2deg^2, cadence, wavelength range Δλ in nm and the typical depth in magnitudes. Surveys listed under 'Future Surveys' are still under construction (LSST) or yet to be constructed (SiTian).

the 48-in Samuel Oschin telescope (P48), in California, and provided a 7.9 deg^2 field of view in two broad-band filters (R, g) covering a wavelength range of $\sim 300\text{nm} - 700\text{nm}$. PTF was capable of observing transients on a range of timescales, with a cadence ranging from minutes to days, reaching a depth of ~ 21 mag in both R and g filters. The P48 telescope was accompanied by the 1.5 metre P60 telescope for dedicated follow-up of transients, capable of providing multi-colour light curves in g, r, i, z filters (Rau et al.,

2009; Law et al., 2009). The PTF survey ran from 2009 to 2012, and was succeeded by the intermediate Palomar Transient Factory (iPTF) (Kulkarni, 2013), with a focus on exploring cadence strategies and rapid processing and follow-up of transient events (Bellm, 2018).

1.3.2 The Zwicky Transient Facility

The successor to the iPTF is the Zwicky Transient Facility (ZTF), which uses the same P48 telescope as PTF but has an upgraded field of view of 47 $\deg^2$ (Bellm et al., 2019). Like its predecessor, ZTF searches for optical transients but also facilitates science for variable objects, small bodies in the Solar System such as asteroids and comets, and also Target-of-Opportunity follow-up observations with instruments capable of observing at wavelengths beyond the optical (e.g. infrared and x-ray).

ZTF is equipped with three filters, *gri*, with a wavelength coverage of 400nm - 900nm, and is capable of reaching a depth of ~ 21 in the g band with a 30 second exposure. The public ZTF survey is allocated to 40% of the observing time, and is conducted following two strategies: a galactic plane survey and a Northern sky survey with a 3 day cadence. For both survey strategies, each field is visited in g and r bands with at least 30 minutes between the two visits. ZTF generates $\sim$ 1TB of uncompressed data per night and provides transient alerts in real-time (Mahabal et al., 2019), allowing for the testing of discovery, classification, and transient alert systems for larger scale future surveys such as the Vera Rubin Observatory Legacy Survey of Space and Time (LSST).

1.3.3 The Gravitational-wave Optical Transient Observer

The Gravitational-wave Optical Transient Observer (GOTO) is a ground-based observatory, with a modular design situated at the Roque de los Muchachos Observatory on La Palma, Canary Islands (Steeghs et al., 2021). GOTO will consist of multiple nodes, with

each node hosting an array of up to eight 40 cm diameter unit telescopes (UTs) providing a combined 40 deg^2 field of view in a single pointing. The current configuration consists of a single node in La Palma, with a plan to add another node in La Palma and another two nodes at the Siding Spring Observatory in Australia. When fully complete, GOTO will have two nodes in La Palma (GOTO North) and two nodes in Australia (GOTO South), for a total of 4 nodes and 32 UTs and the ability to have constant coverage of the night sky. The primary science aim of GOTO is to search for optical signatures following a gravitational wave detection from detectors such as the Advanced LIGO (Abbott et al., 2009) and Advanced Virgo (Acernese et al., 2015) facilities. When GOTO is not searching for gravitational wave counterparts, it conducts an all-sky survey enabling the discovery of new variables and, in particular, new transients.

Each GOTO UT is equipped with four Baader filters: an L filter (covering 400 – 700 nm), and three R, G, B filters (covering ∼380 - 700 nm). The transmission curves for the GOTO filters are shown in Figure 1.3, which has been taken from Dyer (2020).

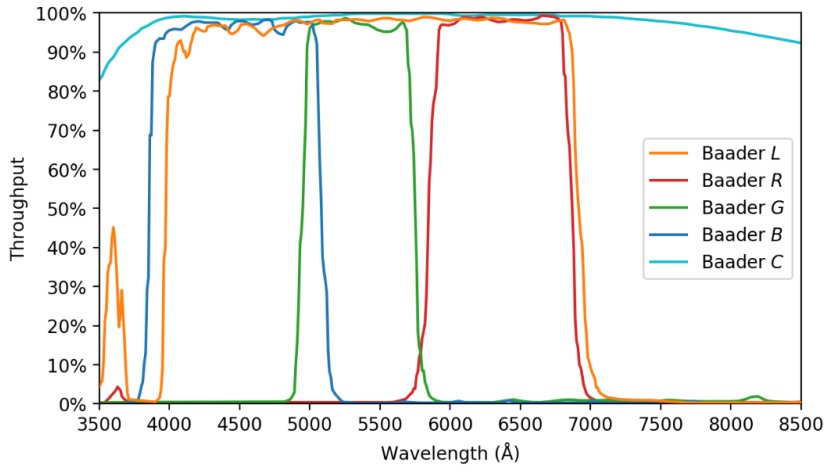


Figure 1.3: Transmission curves for the filter set used by GOTO. Also shown is the transmission curve for the clear glass filter (C).

GOTO currently conducts all-sky surveys with the L filter, and is able to achieve a depth of up to 20.5 mag in a 60 second exposure under dark conditions. New transient and variable detections are obtained through a difference imaging pipeline (Steeghs et al.,

2021). On average, $\sim 1.5 \times 10^3$ new sources were identified through difference imaging per night between the end of 2019 to the end of 2020 when GOTO was operating with a single node.

1.3.4 LSST

The Vera Rubin Observatory Legacy Survey of Space and Time (LSST) (Ivezić et al., 2008, 2019) is a wide field optical telescope with a 9.6 deg^2 field of view currently under construction at Cerro Pachoń Chile. LSST aims to conduct a photometric survey over $30,000$ deg^2 of sky within 10 years with a deep, wide, and fast cadence strategy, and is expected to discover a few million explosive optical transients over its lifetime.

LSST will serve a range of science goals, ranging from probing the nature of dark energy and matter, detailed studies of the solar system and Milky Way, and exploring the optical transient sky. Up to 18,000 deg^2 of the sky will be observed in six optical bands, *ugrizy*, with a wavelength coverage of 320nm to 1050nm. The LSST reference design allows for an average revisit time of 3-4 days for 10,000 deg^2 of sky with two visits per night. With an exposure time of 30 seconds, LSST has a depth ranging from 21.7 in y to $\sim$ 24.5 mags in g in a single visit. A deep-wide-fast survey will use 90% of observing time (the 'main' survey), and 10% will be used to for deeper observations at different time sampling to the main survey in pre-selected 'deep drilling fields'. Approximately 15TB of raw imaging data is expected to be generated every night, and over the 10 year survey lifetime the cumulative processed data is estimated to be 500PB.

1.4 Challenges of time-domain surveys

Surveys such as the ZTF collect large amounts of data throughout their campaigns, putting a strain on their data processing pipelines. The large number of images obtained through these surveys exceeds human capability for manual identification and

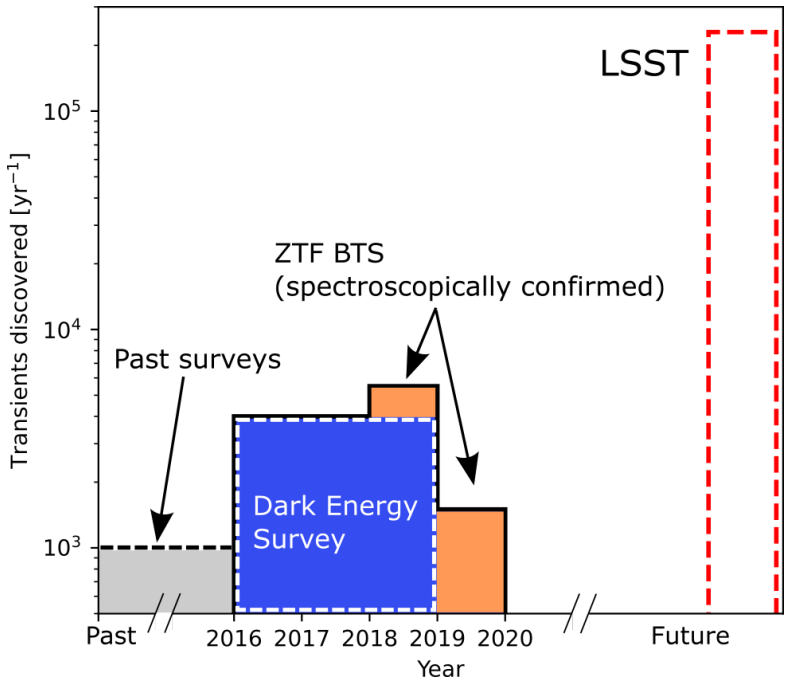


Figure 1.4: The transient discovery rate per year for recent surveys and the expected yearly transient discoveries for LSST, plotted on a log scale histogram. The estimated rates for the Dark Energy Survey were obtained from Smith et al. (2020), for the Zwicky Transient Facility Bright Transient Survey (BTS) from Perley et al. (2020), and for LSST from LSST Science Collaboration et al. (2009). The transient discovery rate for the ZTF BTS survey was estimated using the total number of spectroscopically confirmed transients discovered. Rates for past surveys were obtained from Sullivan (2013) for supernovae discovered up to 2013.

classification of sources (Ball & Brunner, 2010; Bloom & Richards, 2012). Errors in difference imaging pipelines (e.g. due to bad astrometric alignment between images) can introduce artifacts which may be flagged as real time-varying sources, creating the task of separating real sources from 'bogus' subtraction artifacts. The real-bogus classification problem has been well studied and applied to surveys, and is becoming a standard part of automated discovery pipelines (Killestein et al., 2021; Mong et al., 2020; Duev et al., 2019; Lin et al., 2018; Gieseke et al., 2017; Wright et al., 2015; Brink et al., 2013).

Once a source has been identified as real, the next step is classifying the source. Various taxonomies exist in astronomy, with broad categorisations such as consistently varying sources versus transient events, and classifications based on spectroscopic fea-

tures for supernovae sub-types (see section 1.1 for a summary). Usually, confidently classifying new optical transients relies on additional observations with spectroscopic facilities. However, in the era of large scale sky surveys, spectroscopic follow-up of all new discoveries is not guaranteed and is prohibitively time-consuming. As a result, alternative methods of classification relying on photometric and image data are needed.

We discuss in detail the motivation for using machine learning for automated discovery and classification to deal with this challenge in Chapter 2. In Figure 1.4 we plot the annual transient discovery rates for two contemporary surveys and the expected rate for LSST. From Figure 1.4, we see that the current rate of transient discovery from surveys already surpasses that of previous surveys, and future surveys such as LSST will discover an order of magnitude more transients every year compared to current surveys. With such a substantial increase in the rate of discovery, the use of techniques such as machine learning for discovery and classification in transient surveys becomes a necessary step to maximize scientific return.

1.5 This thesis

The focus of this thesis is the use of machine learning (and a subset of machine learning, called deep learning) techniques for the task of classifying objects discovered by time-domain surveys operating in the optical wavelengths, using their light curves (i.e. photometric classification). Automating this process in time-domain surveys where the rate of object discovery exceeds manual (human) capacity facilitates the study of populations of known classes of objects, as well as identifying new objects of interest for further study.

In chapter 2, we provide a review of machine learning and how it is applied in the context of photometric classification of transients in time-domain surveys. We discuss machine learning algorithms, deep learning, and the neural network architectures used

in the work presented in this thesis, and how to evaluate the performance of machine learning and deep learning on classification tasks.

In chapter 3, we apply the random forest machine learning algorithm to classify transient and variable objects from GOTO, using a combination of features extracted from light curves and other object contextual information. We discuss how to deal with a dataset that has class imbalance, the classification performance of the random forest, the importance of features used to make the classification, and the limitations of this approach.

In chapter 4, we improve upon the work presented in chapter 3 and use deep learning for the classification of supernovae, active galactic nuclei, and variable stars on a much larger dataset of light curves from GOTO. We use a recurrent neural network to process variable length light curves to provide 'real-time' classifications that can be updated as new observations of an object are made. We investigate the classification performance and how it improves with more observations and the impact of including additional contextual information about objects.

In chapter 5, we address a limitation on contemporary studies of machine learning and deep learning for the classification of supernova light curves. We create a dataset of real supernova light curves obtained from multiple different surveys, and show how they can be standardised using a Gaussian process regression. A convolutional neural network is used to classify this dataset. We then use transfer learning, to take a classifier trained on this dataset and apply it to a different dataset of simulated LSST supernova light curves. We discuss the classification performance with convolutional neural networks for supernova light curves and the efficacy of transfer learning for creating classifiers for time-domain surveys.

We conclude with Chapter 6, and discuss how the work presented in this thesis fits into the bigger picture of time-domain astronomy, and some directions for future work.

Chapter 2

Machine learning in time-domain astronomy

2.1 Introduction

The advent of larger telescopes has led to an increase in the amount of data gathered by astronomical surveys, as well as an increase in the complexity or dimensionality of the data (i.e. more information is available per object such as multi-band photometry and high cadence time-series data). Such advancements in time-domain astronomy have necessitated the use of machine learning to automate certain tasks. In particular, machine learning is becoming increasingly used in the discovery pipelines of surveys to automate the task of identifying astrophysical objects, and classifying such objects into distinct classes. In this chapter, we introduce the machine learning 'methodology' and background theory that makes up the work presented in this thesis.

In section section 2.2 we discuss the motivation for classifying objects in astronomy. We introduce machine learning and its applications in time-domain astronomy in sections 2.3 and 2.4, deep learning and examples of how it is used for time-domain astronomy in section 2.5. We discuss overfitting and underfitting in section 2.6, and metrics of performance for machine learning in section 2.7.

2.2 Object classification in astronomy

The goal of classifying astronomical objects is to divide the objects based on the their physical properties (such as mass, temperature, or whether the object is a star or a galaxy) or the origin of their variability (stars that vary in brightness periodically versus one-off transient events). At present, the dominant approach to developing astronomical taxonomies is to use observational measurements (such as the flux, either through photometry or spectroscopy), and use *domain knowledge* to develop various classification schemes for the variety of different objects (Bloom & Richards, 2012). Domain knowledge refers to theoretical models proposed to characterise the physical nature of astronomical

objects based on observational data. In chapter 1, we introduced how supernovae are classified by looking for the absence or presence of specific line features in their spectra. Below, we discuss two other examples of commonly used domain knowledge-based classification.

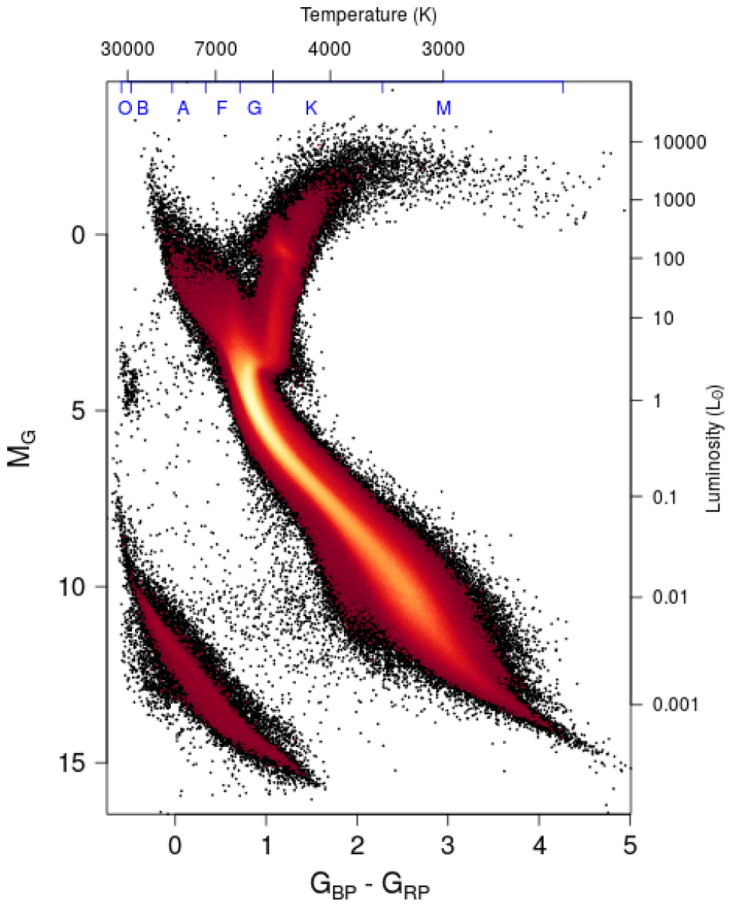


Figure 2.1: GAIA data release 2 colour-magnitude diagram from Gaia Collaboration (2018). The red colour scale represents the relative density of stars in the plot, a brighter colour indicates a higher density. The absolute Gaia magnitude M_G is plotted against the colour $G_{BP} - G_{RP}$, and approximate temperature, spectral type, and luminosity for main sequence stars are shown.

The Hertzsprung-Russell diagram plots the luminosity of stars against their surface temperature, and is useful in visualising the evolution of stars. Typically, the absolute magnitude of the stars is plotted against its colour (the difference in apparent magnitude in two different bands) in a *colour-magnitude* diagram: its luminosity can be derived if

the distance to the star is known, and the temperature can be derived from colour information. Figure 2.1 shows the Hertzsprung-Russell diagram of stars from the second Gaia data release (Gaia Collaboration, 2018). The bright diagonal feature across the diagram represents the main sequence, where stars spend most of their lifetime. Stars then evolve into the giant branch: the arm moving off into the upper right from the main diagonal in the diagram. White dwarfs, the end-points for main sequence stars of Sun-like mass, occupy the bottom left region of the diagram.

A scheme for dividing galaxies into either star-forming or active galactic nuclei was developed by Baldwin et al. (1981) (BPT diagram), using the ratio of emission line intensities in the spectra of galaxies. In star-forming galaxies, emission lines are powered by massive stars whereas active galactic nuclei are powered by more energetic photons which make collisionally excited lines more intense relative to the recombination lines. Figure 2.2 shows the BPT diagram for SDSS galaxies, dividing the galaxies into star-forming (blue) and active galactic nuclei (red). The dashed lines in Figure 2.2 represent two different theoretical demarcations between star-forming galaxies and active galactic nuclei, and the galaxies in grey represent those that lie between these two boundaries.

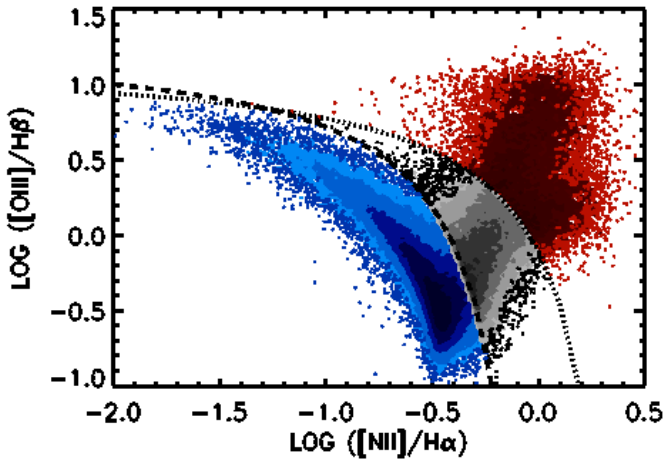


Figure 2.2: BPT diagram for SDSS galaxies from Trouille et al. (2011), where the ratio of line intesities log([OIII]/Hβ) is plotted against log([NII]/Hα). Star-forming galaxies are show in blue, and active galactic nuclei are shown in red.

An alternative approach to domain knowledge-based classification is to use a *feature*-based classification scheme. Features are metrics that can be derived from observational data, such as statistical measures (e.g. mean and standard deviation) from time-series data, or contextual information such as the on-sky location of the object. The features created from the data may be simple (summary statistics) or complex (e.g. period-fitting). For a given dataset (e.g. the light curves of multiple objects), a set of M features can be derived for each sample (a single object) and a M-dimensional feature space can be created. Machine learning is well suited to develop models that can operate in high-dimensional feature space and create decision boundaries to classify the data. Throughout this thesis, we refer to models developed using machine learning to classify data as *classifiers*.

2.3 Machine learning

2.3.1 Defining machine learning

We define machine learning as an approach to create a computer program that can improve its performance on some task by learning from experience. To provide a formal definition, we use the one given in Mitchell (1997):

'A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. '

For example, we can define a machine learning task to be classifying light curves of astronomical objects, the performance measure as the accuracy at which it is able to correctly separate objects into different classes, and the experience as a dataset of light curves. In this case, machine learning is used to create a model that can classify light

curves of objects into different classes by learning from the data (light curves of objects belonging to different classes).

The types of tasks that can be used with machine learning can be divided into three broad types (Bishop, 2006):

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

Supervised learning problems involve the application of machine learning to create a model that can map a set of input data to a set of corresponding target data. The expected mapping of input to target is known beforehand (hence the name supervised learning). Classification is an example of a supervised learning problem where the target data is grouped into a finite number of discrete classes, and regression is an example where the target data may consists of continuous variables. We have outlined an example of a classification task above (light curve classification); an example of a regression task may be estimating the redshift of an object with photometric measurements in place of examining spectroscopic measurements.

Conversely, unsupervised learning involves the use input data without any corresponding target data. Machine learning in this case may be used to identify some correlation between variables in the data. Clustering is an example of an unsupervised learning problem, where a machine learning algorithm is tasked to identify groupings of similar objects within the data, or to identify outliers. This is useful in anomaly detection, where the aim is to identify peculiar or rare objects that are of interest within a dataset (e.g Pruzhinskaya et al. 2019). Other examples include dimensionality reduction where the aim is to project data in high-dimensional space into two or three dimensions so they can be easily visualised (e.g. t-distributed stochastic neighbour embedding (t-SNE), van der Maaten & Hinton 2008).

Reinforcement learning is the problem of identifying actions to take in a specific situation in order to maximise some reward. Common examples of this type of problem involve using machine learning to play video games (see Justesen et al. 2017 for a review).

The focus of this thesis is applying machine learning in a supervised learning context for classification, in particular the classification of transient and variable objects into different classes by using their light curves and other additional information.

2.3.2 Supervised learning in the time domain

In the supervised learning paradigm, a set of input data is used to adjust the parameters of a model. The task involves adapting the model (by changing its parameters) so that it is able to map the input data to a set of target data, where the expected mapping from input to target is known. We call the set of input-target pairs the *training set*. A machine learning algorithm can be thought of as a function $y(x;\theta)$, where x represents the input data and θ the model parameters. The process of tuning $y(x;\theta)$ by changing the parameters θ so that it matches the expected target is known as the *training* phase (or *learning* phase). For classification $y(x;\theta)$ can represent the predicted class of an object and for regression it can represent a predicted continuous variable.

The optimal parameters for a model, so that its predictions match what is expected, are found during the training phase by minimising some loss function. A loss function describes the *error* between the model prediction and the expected outcome. For a regression task, a loss function could be the mean-squared error (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_i')^2 \tag{2.1}$$

where for a set of N data points, y_i is the expected ith measurement and y_i' is the predicted ith measurement. We introduce a loss function for a classification task later in this chapter.

After a model is trained on the training set, it can then be used to make predictions on new data (that is separate to the training set), called the *test set*. The test set contains a set of inputs where the corresponding target data may be unknown. The goal of machine learning is to be able to create a model $y(x;\theta)$ that can correctly produce target data given input data from the test set. This property is known as *generalisation*; a good machine learning algorithm has good generalisation if it is able to produce the expected target data on a test set after being trained on a training set. The terms algorithm and model are used interchangeably to refer to the mapping $y(x;\theta)$ in the context of machine learning[1].

For classification problems, the training set comprises of *feature-label* pairs, where features may either be the raw data itself, or metrics derived from the data, and labels are the class label for a set of features. An example of a feature-label pair is in the spectroscopic classification of supernovae (Filippenko, 1997); a feature may be the absence of hydrogen and the presence of a Si II absorption line in a spectra, and its corresponding label is a Type Ia supernova.

2.4 Machine learning algorithms for classification

The role of machine learning for the task of classification in supervised learning is to create a classifier that can take as an input a set of features and map the features to an expected label. In contrast to traditional 'fitting the data' approach, a classifier has no specific functional form and a machine learning algorithm may have any number of parameters that are tuned during training.

In addition to the parameters learnt from the data, a classifier may also have a number of *hyperparameters* which are parameters not optimised during training and are defined separately. Alongside the training and test set, a *validation* set may also be used

[1]See Bishop (2006) for an in-depth review of machine learning.

during training. The validation set is a subset of data that is separate to the training and test set, and may be used to identify the optimal hyperparameters of a classifier during training. Once a classifier has been trained on the training set, it can be evaluated on the validation set to find the best hyperparameters before being evaluated on the test set. The role of a validation set is to approximate the degree of generalisation that a classifier has developed during training.

Given a labelled training set, the classifier learns how to map the features to corresponding labels (the *feature-class* relationship), and is then tasked with classifying on new, unseen data (i.e. the test set). There are numerous machine learning algorithms that can be used to build such classifiers. In this section we present some examples and their uses for classification in astronomy.

2.4.1 Tree-based classifiers

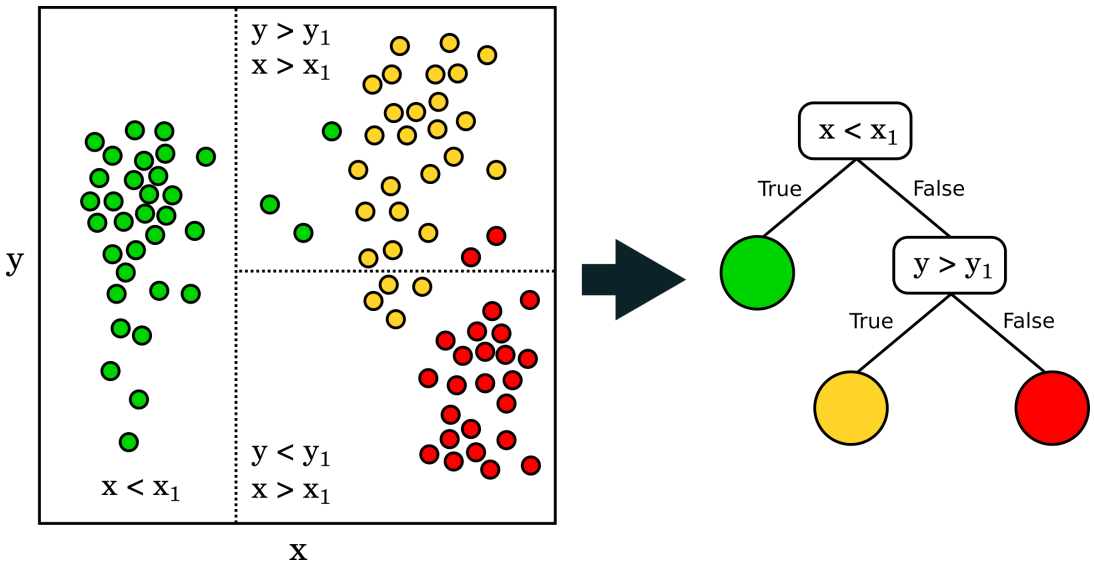


Figure 2.3: Splitting objects of different classes (green, yellow, red) in two-dimensional space by making cuts (left) and a decision tree representation of splitting the objects represented as a series of cuts (right).

A simple and widely used algorithm for classification problems is the decision tree. A decision tree splits the dataset into segments by successively applying selection criteria

such that the number of members from the same class in that segment is maximised (Ball & Brunner, 2010). In other words, the decision tree wants to maximise the *purity* of the sample in a segment. A segment of the data is known as the *terminal node* or *leaf*, which represents samples in the data that have traversed successive selection criteria, which are referred to as the *branches* of the tree. A complex tree structure may arise when attempting to split the data into classes, resulting in several layers of branches and leaves. Figure 2.3 shows how a tree might split three different classes in a two-dimensional feature space.

Decision trees are optimised by minimising a loss function. One such loss function is the Gini impurity, where the decision tree identifies the best splits to make on the training set to create nodes that have high purity by minimising the Gini impurity. The Gini impurity G_i for a node i in the tree is

$$G_i = 1 - \sum_k p_{ik}^2 \tag{2.2}$$

where p_{ik} is the probability of drawing a sample from class k in node i (Breiman et al., 1984). p_{ik} can be estimated as the number of samples from class k divided by the total number of samples in node i. The decision tree minimises G_i, so that by traversing through the branches of the tree, one will arrive at a node where the majority of samples belong to the same class. To make a prediction on an unlabelled sample (i.e. a sample from the test set), the sample is made to traverse through the tree to see which node it arrives at according to which selection criteria apply to it. The node that the sample arrives at is the predicted class of that sample.

Decision trees are however prone to becoming fine-tuned to the training data and not being able to generalise well when classifying unseen data, leading to overfitting. The decision tree may become too *deep*, having too many splits and not choosing the best selection criteria when separating the training data. Ensemble methods, which

create many weak classifiers and aggregate the prediction made by a population of weak classifiers have been used to produce more robust predictions.

The random forest algorithm is an example of an ensemble method which uses an ensemble of decision trees, where each tree is trained on a random subset of the training data, and each split is determined by selecting criteria from a subset of all features, and the final result is obtained through majority voting of all the trees in the ensemble, to give a more robust classification (Breiman, 2001). For a single decision tree, the probability of an object belonging to a class k is the fraction of samples belonging to class k in a node. The random forest can return either the probability of an object belonging to a particular class (by averaging the probabilities across all trees in the forest), or the class of the object (by choosing the class with the highest mean probability).

An alternative ensemble method is the boosted decision tree (Friedman, 2001), which utilises gradient descent (see section 2.5.3 for a discussion on gradient descent) to iteratively add decision trees to a forest by minimising a loss function. Similar to the random forest, a classification probability can be obtained by averaging the probabilities across all trees in the forest.

2.4.2 Support Vector Machines

Support vector machines separate objects into different classes by finding a hyperplane in a multi-dimensional feature space that creates an optimal decision boundary (Cortes & Vapnik, 1995). Data points in feature space are treated as vectors, and decision boundaries are found with respect to support vectors. Support vectors are data points that lie near the decision boundary between different classes. Non-linear relationships between the data can be transformed into a higher-dimensional space so that they become linear (Aizerman et al., 1964), and the hyperplane is found in this space. Figure 2.4 shows how a SVM draws a decision boundary in a two-dimensional space, and Figure 2.5 shows

the use of a kernel trick to find a hyperplane in a higher dimension when the data has a non-linear relationship.

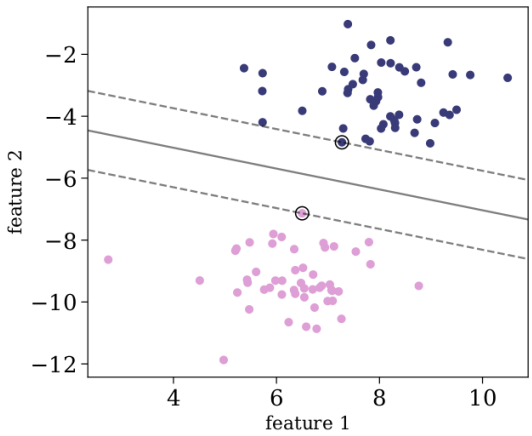


Figure 2.4: The hyperplane for a two-dimensional feature space between two classes (pink and purple circles). The optimal decision boundary (hyperplane) is show as the solid grey line, and the support vectors are circled in black. This figure is taken from Baron (2019)

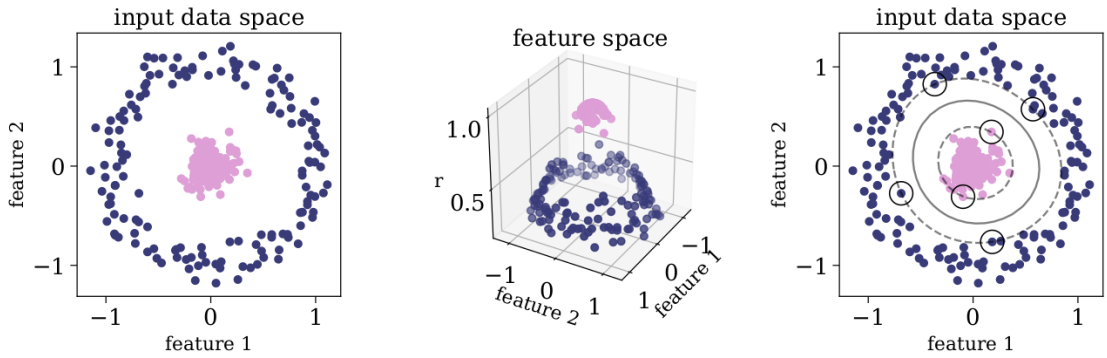


Figure 2.5: The hyperplane for a two-dimensional feature space between two classes (pink and purple circles) for the case when the data has a non-linear relationship. A kernel trick is used to map the data in three-dimensions, where a two-dimensional hyperplane can be used to draw a linear decision boundary. The optimal decision boundary (hyperplane) is show as the solid grey line, and the support vectors are circled in black. This figure is taken from Baron (2019).

2.4.3 Artificial neural networks

Artificial neural networks (also called multi-layer perceptrons) derive their name from attempts to mathematically model biological processes, in particular information processing in neurons (Rumelhart et al., 1986). They can be defined as a non-linear function with a set of adjustable parameters that maps a set of input features to a set of outputs (e.g. for classification these may be class labels, for regression this may be a continuous variable). The parameters for an artificial neural network are found through gradient descent, where the parameters are iteratively updated to minimise a loss function. An in-depth discussion of neural networks and gradient descent is provided in section 2.5. For feature-based classification, the inputs to an artificial neural network are the features derived from the data.

2.4.4 Applications of machine learning for classification in transient astronomy

Feature-based machine learning has been widely used for classification tasks in time-domain surveys. Given a properly trained classifier, the strength of machine learning lies in the ability to perform classification at a faster rate than humans, which allows for rapid follow-up with other facilities for the most interesting objects (Bloom et al., 2012). Discoveries in the time domain are made through either catalogue searching or difference imaging, where a 'reference image' is subtracted from a 'science image'. We introduced discovery methods in time-domain surveys in Chapter 1.

A disadvantage of the difference imaging technique is that errors in the data processing pipeline (e.g. poor image reduction, CCD array defects, poor alignment) can produce 'bogus 'subtraction artifacts which may appear as a time-varying object after subtraction. The first challenge in transient and variable source discovery is identifying the real astrophysical objects from 'bogus 'subtraction artifacts. Machine learning has

been used to classify between real and bogus sources in difference images in time domain surveys. Features derived from images of candidate objects have been used as inputs to a random forest to classify real objects from bogus subtraction artifacts in the Palomar Transient Factory (PTF; Rau et al. 2009; Bloom et al. 2012; Brink et al. 2013) and the Zwicky Transient Facility (ZTF; Bellm et al. 2019; Mahabal et al. 2019). Brink et al. (2013) find that to produce good classifications, it is important to have a large training set and extracting features that describe the data well.

Once a real object has been found, the next step is to identify which class of astrophysical objects it belongs to. Here, we provide examples of how machine learning has been used to classify supernovae in time-domain surveys. The motivation for photometric supernova classification arises from the fact that there will be many more supernovae that are discovered in time-domain surveys than can be followed up with spectroscopic facilities to provide a spectroscopic classification.

Lochner et al. (2016) compared multiple feature extraction methods with a number of machine learning algorithms to classify a set of simulated supernova light curves from the Supernova Photometric Classification Challenge (SPCC) (Kessler et al., 2010) into three classes (Type Ia, Ibc, II). They find that the boosted decision tree produced the best classifications when combined with either highly model-dependent features (such as light curve fitting) of highly model-independent features (performing wavelet decomposition), and that having a representative training set (one that covers the same region of feature space as the test set) improves classification performance. The original SPCC training set emulated a spectroscopically-labelled dataset, where spectra are only available for brighter supernovae. A representative training set was obtained by randomly sampling the same number of supernovae as in the original training from the entire dataset.

Machine learning has also been used to classify real supernova light curves from the Pan-STARRS survey (Pan & Yang, 2010). Villar et al. (2019) used the light curves of 518 spectroscopically classified supernovae to train a classifier to classify the supernovae into

five classes (Type Ia, II, IIn, Ibc, SLSN-I). The training set of Pan-STARRS supernovae was imbalanced, where there were many more samples of supernovae in one class than other classes. Class imbalance can bias classifiers to perform well on classes where there are many examples but perform poorly on classes with few examples. We discuss the class imbalance problem and how to deal with it in Chapters 3 and 4. Data augmentation was used to artificially generate additional samples for classes with the fewest samples (see section 3.4.2 for a detailed discussion). Features extracted from the light curves were used in a support vector machine, random forest, and an artificial neural network to classify the light curves. Random forests were found to be the best performing classifier, and Type Ia supernovae and SLSN-I had the best classification results due to uniformity within the class (Type Ia) and high luminosity and long duration (SLSN-I).

The above examples are well-suited for manually extracted features from photometric time-series data. By manually we mean that the feature extraction process requires domain knowledge to identify the best features that capture information about objects that enable machine learning algorithms to learn how to separate them in feature space. Often, these features are extracted from 'complete' light curves, where the rise, peak, and decline of transients have been observed. The ability to classify supernovae based on just light curves will benefit the studies of cosmology with Type Ia supernovae (e.g. Betoule et al. 2014) and accumulating a large sample of core-collapse supernovae allows for population studies to understand their diversity (e.g. Modjaz et al. 2019). A goal of time-domain astronomy is to enable the study of physical processes that occur early in the photometric evolution (i.e. before peak brightness) of transients, so there is a need for a classifier that can reliably classify objects with just a few observations.

2.5 Deep learning

An alternative to feature-based machine learning is *deep learning*, where a model is trained to learn how to extract features from the raw data itself and also perform the subsequent input-to-target mapping using those extracted features. This has the benefit of not having to manually select and extract features from the data for a machine learning task. In this section, we provide an introduction on deep learning [2].

The basis for deep learning is the use of neural networks, which is essentially a mathematical function $y(x;\theta)$ that has parameters θ and learns to map an input x to an expected target. A neural network is composed of multiple simple functions (or neurons) that applies transformations to the inputs. We begin our discussion of deep learning by introducing the artificial neuron, and how an artificial neural network is constructed from these artificial neurons.

2.5.1 Neural networks

An artificial neuron (or just neuron) is a mathematical function (more precisely a non-linear mapping) that takes an input (which maybe a single valued scalar or a vector of multiple values), and applies a transformation to the output and returns a scalar output. As an example, consider the sigmoid function:

$$\text{sigmoid}(\boldsymbol{x}) = \frac{1}{1+\exp(\sum_i^m x_i)} \tag{2.3}$$

where $\boldsymbol{x}$ is an input in the form of an m-dimensional vector $\boldsymbol{x} = \{x_1, ..., x_m\}$. The sigmoid function transforms the inputs such that the output is in the range (0,1). This can be used for a binary classification problem, where output may represent the probability p of an object belonging to one class (and the probability of it belonging to the other class

[2]For a detailed review of deep learning, see Goodfellow et al. (2016) and Bishop (2006).

is just $1 - p$), given an input $\boldsymbol{x}$. We can introduce some weights $\boldsymbol{w}_1$ and a bias w_0 to equation 2.3, so that it now has the form

$$\text{sigmoid}(\boldsymbol{w}_1^\intercal \boldsymbol{x} + w_0) = \frac{1}{1 + \exp(\boldsymbol{w}_1^\intercal \boldsymbol{x} + w_0)} \tag{2.4}$$

where $\boldsymbol{w}_1^\intercal \boldsymbol{x}$ represents the dot product of the weights vector $\boldsymbol{w}_1$ and input vector $\boldsymbol{x}$. This is known as *logistic regression*, which can be used to produce a classification by returning a probability that an object belongs to a class given some input $\boldsymbol{x}$, and the weights $\boldsymbol{w}_1$ and bias w_0 are optimised so that the output matches the expected target. A neuron is just a form of logistic regression; it takes in a set of inputs and returns a scalar value. Figure 2.6 shows a diagram of a neuron. The function that transforms the inputs is known as the *activation function*, and it is not limited to just the sigmoid function. Other choices of activation functions include the hyperbolic tangent $\tanh(x)$ and the rectified linear unit $\text{ReLU}(x)$. The ReLU function is defined as $y(x) = 0$ when $x < 0$ and $y(x) = x$ otherwise. Figure 2.7 shows the sigmoid, tanh and ReLU activation functions.

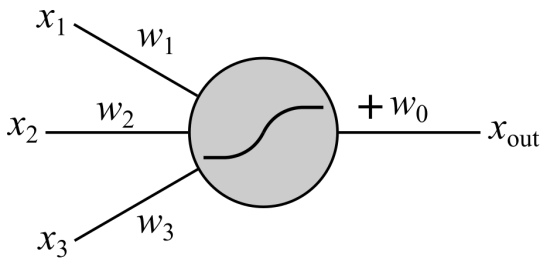


Figure 2.6: An artificial neuron. The inputs x_1, x_2, x_3 with corresponding weights w_1, w_2, w_3 are represented by connections (lines in the diagram) that 'connect' to the neuron, where the sigmoid symbol represents the application of an activation function. The addition of a bias w_0 is shown for the connection going out from the neuron.

A neural network is built up of multiple layers of neurons. Since the output of a single neuron is a scalar, the output of a layer of neurons can be considered as a vector. As an example, consider a neural network consisting of an input layer, one layer of neurons, and

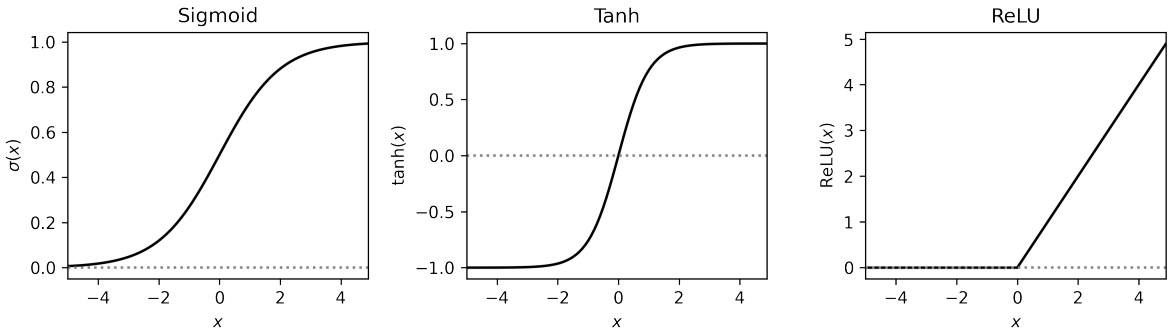


Figure 2.7: The sigmoid (*left*), tanh (*middle*), and ReLU (*right*) activation functions.

a final layer with a single neuron that outputs a value between 0 and 1. This example neural network can be thought of a classifier that takes as an input a vector of features $\boldsymbol{x}$, and outputs a probability p of class membership in a binary classification scenario. The intermediate layer of neurons is often called the *hidden layer*, and a neural network may be made up of multiple hidden layers.

A neural network that is made up of several hidden layers can be considered as a 'deep neural network', and the central motivation of deep learning is to construct such deep neural networks that can take in raw input data (without feature extraction) and are able to learn a representation that allows the deep neural network to map the raw input data to an expected target, hence the term 'deep learning'.

To describe a neural network mathematically, one can consider a layer of neurons as a function of the outputs of the previous layer in the network:

$$\boldsymbol{x_k} = f_k(\boldsymbol{W}_{1k}^{\mathsf{T}}\boldsymbol{x}_{k-1} + \boldsymbol{w}_{0k}) \tag{2.5}$$

where $\boldsymbol{W}_{1k}$ is the weight matrix for the kth layer, $\boldsymbol{x}_{k-1}$ is the output of the $(k-1)$th layer, and $\boldsymbol{w}_{0k}$ are the biases of each neuron in the kth layer. The weight matrix $\boldsymbol{W}_{1k}$ is defined as such since each neuron in a layer takes as input the outputs of all neurons in the previous layer. In the deep learning parlance, these are called 'connections' between neurons across layers. This neural network structure is illustrated in Figure 2.8.

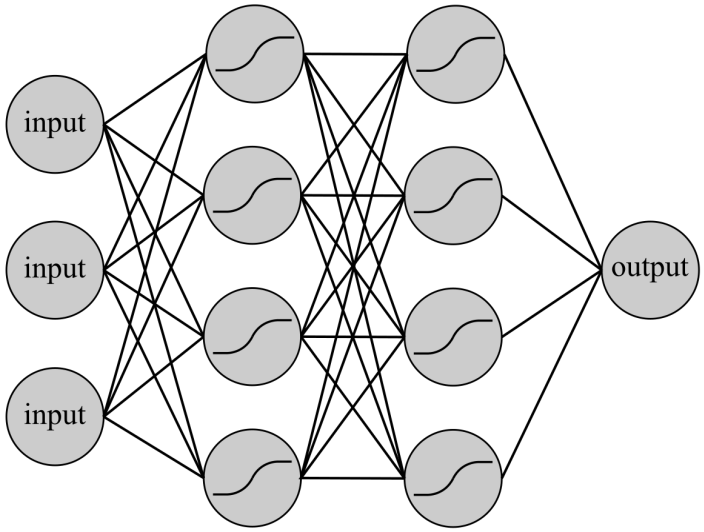


Figure 2.8: A basic neural network. The lines between neurons are referred to as 'connections', and represent the weights and biases applied to the inputs fed into neurons.

In the case of the input layer of a neural network, the layer is a function of the input vector and the layer weights and biases. For a neural network with multiple layers, the first layer is given by

$$\boldsymbol{x}_1 = f_1(\boldsymbol{W}_{11}^{\intercal}\boldsymbol{x}_0 + \boldsymbol{w}_{01}) \tag{2.6}$$

where $\boldsymbol{x}_0$ is the input vector. The next layer is given by

$$\boldsymbol{x}_2 = f_2(\boldsymbol{W}_{12}^{\intercal}\boldsymbol{x}_1 + \boldsymbol{w}_{02}) = f_2\Big(\boldsymbol{W}_{12}^{\intercal}\Big(f_1(\boldsymbol{W}_{11}^{\intercal}\boldsymbol{x}_0 + \boldsymbol{w}_{01})\Big) + \boldsymbol{w}_{02}\Big) \tag{2.7}$$

where $\boldsymbol{x}_1$ is the output of the first layer. Subsequent layers of the network are described similarly, forming a chain-based architecture (i.e. subsequent layers of the network can be described as a function of the previous layer). In the context of neural networks, the term architecture refers to how the network is structured (i.e. how many neurons in a layer, how many hidden layers there are and how are the layers connected to each other).

For classification tasks where there are more than two classes, the softmax activation function can be used in the final output layer, which represents the probability that

on object belongs to one of N classes. Given a vector of layer outputs $\boldsymbol{x}$, the softmax function is defined as

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_j^N \exp(x_j)} \tag{2.8}$$

where the x_i is the ith value of the vector $\boldsymbol{x}$. The final output is a vector of probabilities $[\text{softmax}(\mathbf{x})_1, \text{softmax}(\mathbf{x})_2, ..., \text{softmax}(\mathbf{x})_N]$ that sums to 1, that describes the probabilities of class membership to classes $1, 2, ..., N$.

Now that we have defined what a neural network is, in the following sections we discuss how the weights and biases of a neural network are optimised.

2.5.2 Network parameter optimisation

Given an input vector $\boldsymbol{x}$ and an expected target vector $\boldsymbol{t}$, we want to find a neural network $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{w})$ that produces an output $\boldsymbol{y}$ that matches $\boldsymbol{t}$. To do so, the network parameters $(\boldsymbol{W}, \boldsymbol{w})$ (where $\boldsymbol{W}$ represents the weights and $\boldsymbol{w}$ represents the biases) need to be found such that the error between $\boldsymbol{y}$ and $\boldsymbol{t}$ is minimised. We define a *loss function* as one that describes the error between the neural network output $\boldsymbol{y}$ and the expected target $\boldsymbol{t}$

$$E(\boldsymbol{W}, \boldsymbol{w}) = f(\boldsymbol{y}, \boldsymbol{t}) \tag{2.9}$$

In the case of a multi-class classification task, where an input needs to be mapped to one of N classes, the cross entropy loss function is used for neural networks:

$$E(\boldsymbol{W}, \boldsymbol{w}) = -\sum_{k=1}^{K} \sum_{n=1}^{N} t_{kn} \ln(y_n(\boldsymbol{x}_k, \boldsymbol{W}, \boldsymbol{w})) \tag{2.10}$$

where y_n is the probability of object k with input vector $\boldsymbol{x}_k$ belonging to class n, t_{kn} is the class label for object k (where if the object belongs to class 1 then $t_{k1} = 1$ and $t_{k2,...,N} = 0$), and the loss is obtained by summing over all K objects in the dataset.

To optimise the the network parameters $(\boldsymbol{W}, \boldsymbol{w})$, we want to find the network param-

eters $(\boldsymbol{W}, \boldsymbol{w})$ such that $E(\boldsymbol{W}, \boldsymbol{w})$ is minimised. We first redefine the neural network as a mapping f from an input $\boldsymbol{x}$ to a target $\boldsymbol{t}$, and represent the network parameters $\boldsymbol{W}, \boldsymbol{w}$ as $\boldsymbol{\Theta}$, so the neural network can be expressed as $f(\boldsymbol{x}; \boldsymbol{\Theta})$. Then the problem of optimising the network parameters becomes finding $\boldsymbol{\Theta}$ such that the loss $E(\boldsymbol{\Theta})$ is minimised. To do so, we use the information about the gradient of $E(\boldsymbol{\Theta})$ in the parameter space $\boldsymbol{\Theta}$.

The smallest value of $E(\boldsymbol{\Theta})$ will occur where $\nabla E(\boldsymbol{\Theta}) = 0$ at a minimum in the parameter space. $E(\boldsymbol{\Theta})$ typically has a non-linear dependence on $\boldsymbol{\Theta}$, so there may be multiple minima in parameter space where $E(\boldsymbol{\Theta}) = 0$. Where the minimum corresponds to the smallest value for $E(\boldsymbol{\Theta})$ for any $\boldsymbol{\Theta}$ is called a *global minimum*, and any other minima that corresponds to higher values for $E(\boldsymbol{\Theta})$ are called *local minima*. In practice, most neural networks work well without having to find the global minimum. The process of finding $\nabla E(\boldsymbol{\Theta}) = 0$ is usually done iteratively, by moving through the parameter space in incremental steps:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t + \delta\boldsymbol{\Theta}_t \tag{2.11}$$

where t denotes the iteration step.

2.5.3 Gradient descent

To navigate the parameter space to find a minimum, one can use information about the gradient of $E(\boldsymbol{\Theta})$ by taking a small incremental step in the direction of the negative gradient

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \eta\nabla E(\boldsymbol{\Theta}_t) \tag{2.12}$$

where η is known as the *learning rate*, and describes how much of a step is taken during each iteration. During each iteration, $\boldsymbol{\Theta}$ moves in the direction of the greatest decrease of the loss function, so this method is referred to as *gradient descent*. Since $\nabla E(\boldsymbol{\Theta})$ is used, equation 2.12 requires the evaluation of the loss function over all training samples. Gradient descent methods that require the use of the entire training set are called *batch*

gradient descent methods. An iteration over the entire training sample is referred to as one *epoch of training*. The number of training epochs to optimise the neural network parameters can be found through experimentation. To find a satisfactory minimum that minimises the loss, it may be necessary to perform gradient descent multiple times.

A method called *stochastic gradient descent* (LeCun et al., 1989) updates $\boldsymbol{\Theta}$ by taking steps using the gradient evaluated for each training sample. Stochastic gradient descent is defined as

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \eta \nabla E_n(\boldsymbol{\Theta}_t) \tag{2.13}$$

where the loss function $E_n(\boldsymbol{\Theta}_t)$ is the loss for the nth sample in the training set. The update is repeated by going through the training samples sequentially or by randomly selecting training samples with replacement. This approach is useful for loss functions such as equation 2.10 that are comprised of a sum of terms for each training sample. We visualise gradient descent in Figure 2.9.

The drawback of batch gradient descent is that it is evaluated over the entire training sample for each iteration, and it is not guaranteed that the best minimum will be found. Stochastic gradient descent performs gradient descent a number of times equal to the number of samples in the training set. While this may lead to finding a satisfactory minimum, it requires performing gradient descent for each training sample which takes longer to compute. In practice *mini-batch gradient descent* is used for updating neural network parameters, where the training set is split into equally sized batches, and the loss is evaluated for each batch and gradient descent is performed for each batch in an iteration. The *batch size* sets the number of samples in a batch.

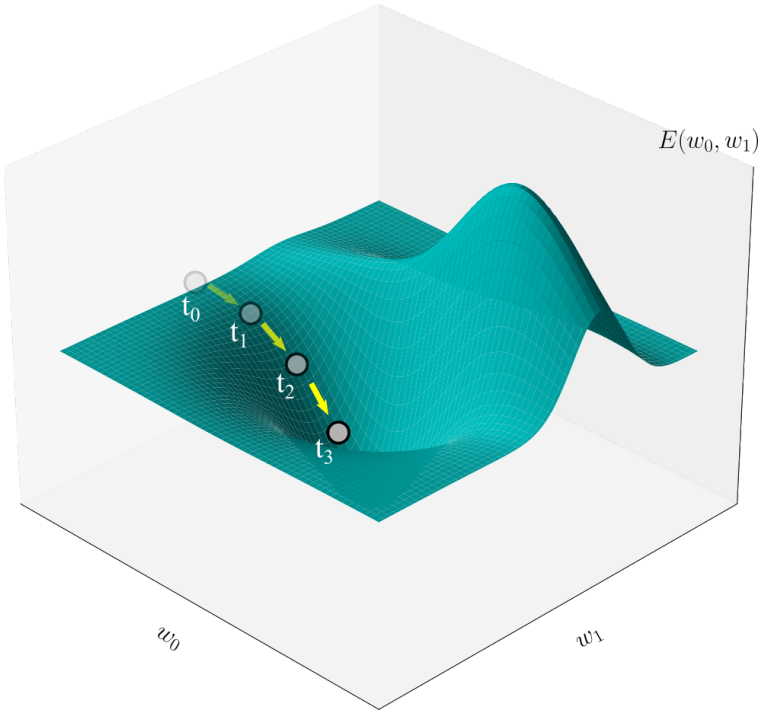


Figure 2.9: A visualisation of gradient descent in a two-dimensional surface representing a loss function $E(w_0, w_1)$ as function of neural network parameters w_0 and w_1. The parameters w_0 and w_1 are iteratively updated over increasing time steps $(t_0, ..., t_3)$ until a minimum for $E(w_0, w_1)$ is found.

2.5.4 Error backpropagation

In this section we briefly describe how the gradient of a loss function for a neural network is computed[3]. The process of training a neural network involves minimising a loss function, by making small adjustments to the parameters in iterative steps. In each iteration, there are two stages:

1. Evaluate the derivative of the loss function with respect to the parameters

2. Use the derivative to calculate the adjustments to be made to the parameters

The technique for passing information through the network to evaluate the gradient of the loss function is known as *error backpropagation*, and is needed in the first stage. In

[3]A detailed discussion of error backpropagation is provided in Chapter 5 of Bishop (2006) and Chapter 6 of Goodfellow et al. (2016)

the second stage, the parameters are adjusted using an optimisation process, such as the gradient descent method.

The process of passing an input $\boldsymbol{x}$ through the neural network and calculating the outputs of all neurons in the hidden and output layers is known as *forward propagation* since information is propagated forwards through the network. After forward propagation, the loss function can be evaluated.

The derivative of $E(\boldsymbol{\Theta})$ can be expressed in terms of 'errors' in the hidden layers and the output layer. An 'error' refers to a small change in the parameters of the hidden layers of a neural network, which contributes to a small change in the output in the final output layer. By knowing the error of the output layer, it is possible to infer the errors of the hidden layers. The error of the output layer is backpropagated to evaluate the errors of the hidden layers. Once these errors are known, it is possible evaluate the derivative of the loss function with respect to the parameters. And once the derivatives are known, adjustments to the parameters can be made using gradient descent methods.

2.5.5 Deep neural networks for classification

So far we have discussed the basis of how neural networks can take a set of training inputs and corresponding targets, and generate a predicted output by adjusting the network parameters so that the loss between the predictions and the targets are minimised. It has been outlined that the motivation of deep learning with neural networks is to have a neural network that can learn to extract features from the data automatically. In this section, we review two deep neural network architectures that are well-suited for processing sequential data and image data. These two particular architectures are used for the work presented in Chapters 4 and 5.

Recurrent neural networks

Recurrent neural networks (RNNs) are a class of neural networks that operate on a sequence of vectors $[\boldsymbol{x}_1, ..., \boldsymbol{x}_t]$, where the subscript indicates the time-step index. This index is not strictly limited to representing time, but can also represent the position of a vector in the sequence (e.g. words in a sentence). There are several possible mappings with recurrent neural networks; producing an output at each time step using information from previous time steps (many-to-many) or producing a single output after all time-steps have been processed (many-to-one). In this chapter we discuss recurrent neural networks in the context of the many-to-one mapping, as this is the architecture used for time-series classification in Chapter 4.

A recurrent neural network uses the same weights across time-steps to learn time dependencies in the data (a process known as *parameter sharing*). Consider a classical dynamical system, where the state of the system at the current time-step t depends on the state at the previous time-step $t-1$, parameters $\boldsymbol{\theta}$, and an external signal $\boldsymbol{x}_t$ at the current time step:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{2.14}$$

To know the state of the system at $t = 3$, we use equation 2.14 and write

$$\boldsymbol{s}_3 = f(\boldsymbol{s}_2, \boldsymbol{x}_3; \boldsymbol{\theta}) = f(f(\boldsymbol{s}_1, \boldsymbol{x}_2; \boldsymbol{\theta}); \boldsymbol{\theta}) \tag{2.15}$$

We see that the state at the current time-step contains information about all previous time-steps, and the parameters are shared across time. Recurrent neural networks use a similar formulation to describe the outputs of the hidden layers, or in the context of recurrent neural networks the hidden state $\boldsymbol{h}_t$ at a time step t given an input $\boldsymbol{x}_t$:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{2.16}$$

The function f describes the activation function, which has the same operation for recurrent neural networks as neural networks in section 2.5.1. In practice, recurrent neural networks will use a number of different methods to retain some information about past hidden states rather than all information of all past states (we discuss these later in this section.)

To describe forward propagation in recurrent neural networks, we first define a hidden state $\boldsymbol{h}_t$. Given an input of a sequence of vectors $\boldsymbol{x}_t$ where the time-step index runs from $t = 1$ to $t = \tau$, the hidden state at time t is:

$$\boldsymbol{h}_t = f(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{b}_h) \tag{2.17}$$

where $\boldsymbol{W}_h$ is weight matrix for the connections between hidden states across time, and $\boldsymbol{W}_i$ is the weight matrix for the connections between the input layer to the hidden layer, and $\boldsymbol{b}_h$ is the bias vector applied to the hidden state. The final output of the recurrent neural network, after reading in all time-steps of the input sequence is given by

$$\boldsymbol{y} = g(\boldsymbol{h}_\tau) \tag{2.18}$$

where g represents an activation function for the final output layer (e.g. a softmax function), and $\boldsymbol{h}_\tau$ is the hidden state of the final time-step (Figure 2.10).

In a many-to-one RNN, the loss takes the form of a loss function that describes the error between the predicted output and the target (e.g. cross entropy loss). The computation of the gradient of the loss function with respect to the parameters involves the forward propagation moving in the direction of increasing time-steps (i.e. forward in time) and then a backward propagation in the opposite direction. The analog of backpropagation in neural networks to recurrent networks is known as *backpropagation through time* (Werbos, 1990)[4]. The parameters of a recurrent neural network can then

[4]See chapter 10 of Goodfellow et al. (2016) for a discussion of how a gradient is computed in a

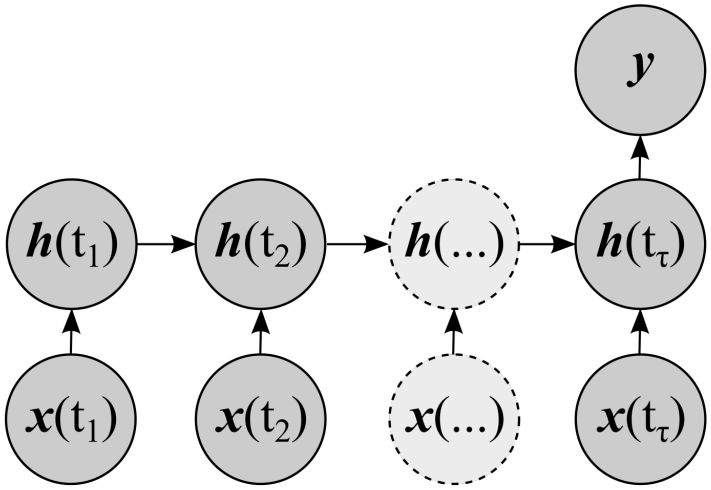


Figure 2.10: A recurrent neural network.

be optimised via gradient descent methods.

From equation 2.16, we see that the hidden state for a long sequence is function of many functions going back to the initial hidden state. The derivative of a function of a function is evaluated using the chain rule which gives a product of partial derivatives. Given a long sequence of nested functions (a function of a function of a function and so on), the derivative of the nested function becomes a product of many terms. Since recurrent neural networks share parameters across time, the gradient is evaluated with respect to the shared parameters at each time step. The derivative of a hidden state for a long sequence contains the product of gradients at each time-step t. This is a problem because if the gradient is small, then the gradient will vanish after successive multiplications, or conversely if the gradient is large then the gradient will grow exponentially after successive multiplications. This problem is known as the *vanishing and exploding gradient problem* for recurrent neural networks. Variants of the 'traditional' recurrent neural network have been developed to overcome this problem.

Successful practical applications of recurrent neural networks utilise *gated recurrent*

recurrent neural network.

neural networks that use connections between neurons that can change at each time-step, and introduce derivatives that do not vanish or explode.

Long short-term memory recurrent neural network

The *long short-term memory* (LSTM) recurrent neural network (Hochreiter & Schmidhuber, 1997; Graves, 2013) is an example of a gated recurrent neural network. The LSTM introduces the 'LSTM cell' which has an internal recurrence, additional parameters, and a series of 'gates' that regulates the flow of information from one hidden state to the next hidden state. Here, we describe the LSTM implementation presented in Graves (2013).

At a time step t, we define an input $\boldsymbol{x}_t$, the hidden state from the previous time-step $\boldsymbol{h}_{t-1}$, and a cell state from the previous time-step $\boldsymbol{c}_{t-1}$. The *input gate* $\boldsymbol{i}_t$ is

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ci}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i\right) \tag{2.19}$$

and the *forget gate* $\boldsymbol{f}_t$ is

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cf}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f\right) \tag{2.20}$$

where σ is the sigmoid function. The biases are denoted by $\boldsymbol{b}$ and the subscripts of the weight matrices represents the connections; $\boldsymbol{W}_{xi}$ are the weights for the connection between the input $\boldsymbol{x}_t$ and the input gate $\boldsymbol{i}_t$, $\boldsymbol{W}_{hi}$ are the weights for the connection between the hidden state $\boldsymbol{h}_t$ and the input gate $\boldsymbol{i}_t$ etc. The cell state for the current time-step is

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh\left(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c\right) \tag{2.21}$$

where $\odot$ is the element-wise Hadamard product[5].

From equations 2.19 and 2.20, we see that the input gate $\boldsymbol{i}_t$ and forget gate $\boldsymbol{f}_t$ are

[5]The Hadamard product of two identically shaped matrices A and B is simply a matrix C where element $C_{i,j}$ of C is the product of $A_{i,j}$ and $B_{i,j}$.

valued between 0 and 1 due to the application of the sigmoid function. In the cell state, the forget gate $\boldsymbol{f}_t$ acts as a weight to control how much information from the cell state in the previous time-step (which contains information about the input from the previous time-step) carries over to the current state. In other words, it is adjusted to 'forget' information from previous timesteps. The input gate $\boldsymbol{i}_t$ acts as a weight to control how much information about the previous hidden state and the input at the current time-step is retained in the current state. The output gate $\boldsymbol{o}_t$ is

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{co}\boldsymbol{c}_t + \boldsymbol{b}_o\right) \tag{2.22}$$

and finally the hidden state $\boldsymbol{h}_t$ is

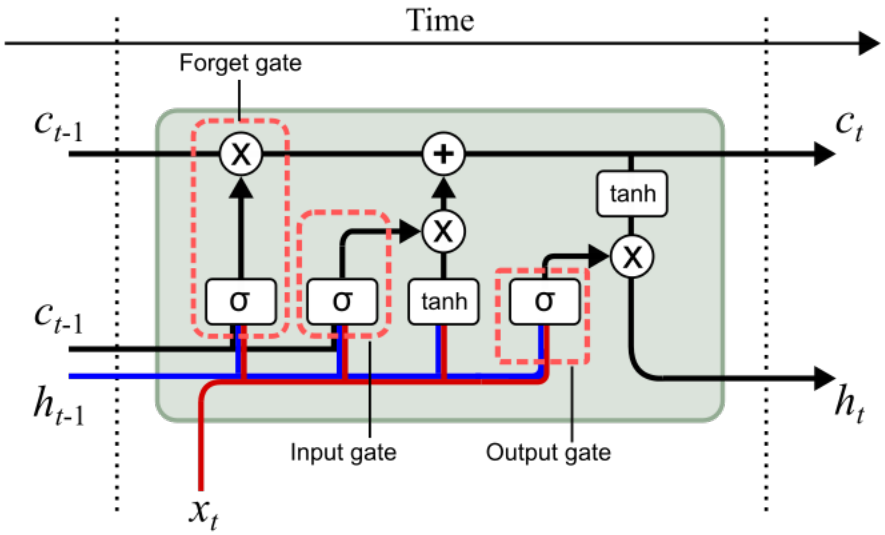
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh\left(\boldsymbol{c}_t\right) \tag{2.23}$$

The output gate $\boldsymbol{o}_t$ weights how much information about the cell state at the current time-step is passed on in the hidden state to the next time-step. We illustrate how information from previous time steps are processed with an LSTM in figure 2.11a.

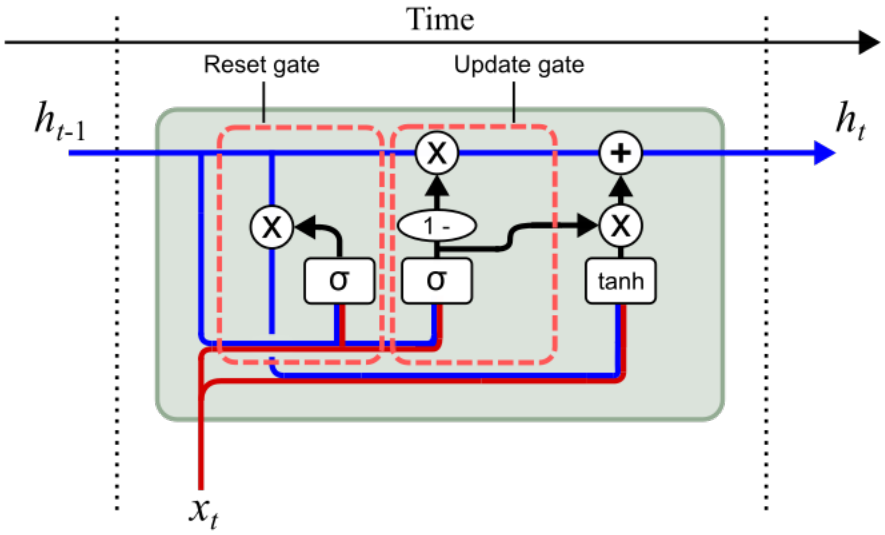
The LSTM allows a recurrent neural network to 'forget' information from time-steps in the past, and it does so by using a set of gates that have their own parameters. During training, these parameters are optimised, and the recurrent neural network effectively learns when to forget information about the past. This allows the LSTM recurrent neural network to learn from long sequences, and deal with the vanishing and exploding gradient problem, since the gradients do not depend on the entire history of the sequence.

Gated recurrent unit recurrent neural networks

Another gated recurrent neural network architecture makes use of gated recurrent units (GRU) (Cho et al., 2014). Similar to the LSTM, the GRU uses gates to regulate the



(a) LSTM cell diagram.



(b) GRU cell diagram.

Figure 2.11: Diagrams showing the flow of information through the LSTM and GRU cells described in equations 2.19 to 2.26. In both diagrams, the 'cross' symbol represents a multiplication operation and the 'plus' symbol represents an addition operation. The 'σ' represents the sigmoid function, 'tanh' represents the tanh and the '$1-$' represents the $(1-\boldsymbol{z}_t)$ term in equation 2.26. The gates in the cell diagrams are highlighted within the dashed line boxes.

flow of information through time but uses a single gate to simultaneously control how much information from previous time-steps is retained and how the current hidden state is updated. This is done by using a *reset gate* and an *update gate*.

Given an input $\boldsymbol{x}_t$ at the current time-step t, and the hidden state of the previous time-step $\boldsymbol{h}_{t-1}$, the reset gate $\boldsymbol{r}_t$ is

$$\boldsymbol{r}_t = \sigma\big(\boldsymbol{W}_{xr}\boldsymbol{x}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}\big) \tag{2.24}$$

and the update gate $\boldsymbol{z}_t$ is

$$\boldsymbol{z}_t = \sigma\big(\boldsymbol{W}_{xz}\boldsymbol{x}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1}\big) \tag{2.25}$$

The hidden state at the current time-step $\boldsymbol{h}_t$ is

$$\boldsymbol{h}_t = \boldsymbol{z}_t\boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t)\tanh\big(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{ph}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})\big) \tag{2.26}$$

where $\boldsymbol{W}_{ph}$ is the weight matrix for the connection between $(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})$ and the hidden layer. From equation 2.24, when the reset gate is close to 0 the hidden state forgets the hidden state from the previous time-step and is only updated with the the input at the current time-step. The update gate controls how much information is passed on to the next time-step. We illustrate how information from previous time steps are processed with a GRU in figure 2.11b. Like the LSTM, this mechanism allows the GRU to learn dependencies over long time scales (i.e. long sequences).

Convolutional neural networks

Convolutional neural networks (CNNs; see Lecun 1989) are a class of neural networks that can process data with a grid-like structure[6]. This can be in one dimension (such as a sequence of measurements in time), two dimensions (an image made up of pixels), or even three dimensions (a colour image decomposed into red, green, and blue channels). This done by applying the convolution operation to the data, (hence the name of the neural network) to identify spatial features (such as edges or corners in an image).

[6]See Chapter 9 of Goodfellow et al. (2016) for a detailed discussion on convolutional neural networks

Consider a two-dimensional image I. The convolution S of I with a kernel K in two dimensions is

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.27}$$

where i and j represent the pixel coordinates, and m and n represents the entries in the kernel K. Here, a kernel refers to a matrix that typically has smaller dimensions than the input image. A convolution is simply a a matrix multiplication of the kernel and a subset of the image with the same dimensions as the kernel (see Figure 2.12 for an example convolution) repeated over the image by shifting the kernel around the image pixel by pixel. The output of a convolution is referred to as a *feature map*.

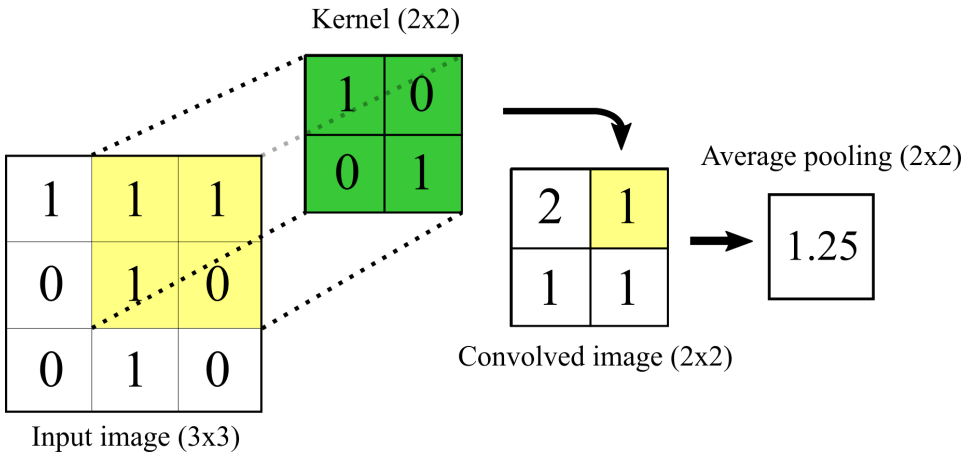


Figure 2.12: An example convolution with of a 2×2 kernel on a 3×3 image, and the application of a 2×2 average pooling window on the convolved output. The application of a 2×2 kernel on a 3×3 image results in a 2×2 output.

By making the kernel smaller than the input, a convolutional neural network has *sparse connectivity*. Each output in the convolution only depends on a small region of the input image which is set by the size of kernel. This is in contrast to fully-connected hidden layers in neural networks where each neuron in a layer is connected to all neurons in the previous layer. Due to this sparse connectivity, a convolutional neural network is able to identify features in small regions of the input image. This is useful when the image

dimensions are large, the size of kernel can be kept orders of magnitude smaller than the image, and a convolutional neural network will still be able to identify meaningful features in the image without a high computation cost.

Like a recurrent neural network, a convolutional neural network also makes use of parameter sharing. The same kernel is applied at each position in the image, so the convolutional neural network only has to learn the parameters of one kernel as opposed to a separate kernel for each position. The form of parameter sharing in convolutional neural networks means that a convolutional layer has a property known as *equivariance*. This means that if the input to the convolution is changed in some way, then the output is also changed in the same way.

The convolution operation is not the sole operation used in a convolutional neural network, it also paired with a *pooling* function. After applying a convolution, the output is passed through an activation function (e.g. ReLU), in what is called the *detector stage*, which is followed by pooling. A pooling function replaces the output of a convolution at a certain position with a summary statistic of values surrounding that position. The size of the pooling window is an adjustable hyperparameter, and defines the size of the window on the convolution output over which pooling is done. Examples of pooling functions are *Max pooling*, where the summary statistic returned is the maximum value of the pooling window, and *average pooling* where the summary statistic returned is the average of the pooling window. Pooling helps to make the representation of the input image approximately invariant to small translations of the input, and it summarises the convolutions over a region. Thus, it is possible to use fewer pooling operations than convolutions.

For both the convolution and pooling operations, it is possible to skip some positions in the input image to reduce computational cost. We can define a *stride* parameter that sets the number of pixels to skip when doing convolution and pooling.

For example, a stride of one for the convolution layer means that the kernel moves

across one pixel at a time and a stride of two means that the kernel moves across two pixels at a time over the image. The stride can also be defined in two dimensions (e.g. kernel moves across horizontally a specified number of pixels at a time and vertically specified number of pixels at a time for a given stride parameter). The number of kernels used is not just limited to one, it is possible to have multiple kernels applied to the same input image, and have multiple convolution outputs from a single image. Each kernel can be randomised, and the parameters of the kernels learnt during training. In Chapter 5, we use the term *filters* to refer to the number of kernels used to generate convolutional outputs. For example, if n different kernels are used then the number of filters is n.

To train a convolutional neural network, the gradients with respect to the kernel parameters need to be computed to find the optimal parameters with gradient descent methods. Convolution can be described as a matrix multiplication, and the use of matrix multiplication operations can be used to back-propagate errors through a convolutional layer to compute gradients[7].

2.5.6 Applications of deep learning for classification in transient astronomy

Deep learning has been successfully used in various classification tasks in time-domain astronomy, with an advantage over feature-based machine learning techniques since deep learning approaches do not require a feature extraction step. The raw data does require some preprocessing before use with neural networks, but this is mainly done to format the data in a specific manner rather than to compute features. In this section we highlight some examples of how deep learning has been used to classify transients in time-domain surveys.

[7]See Chapter 9 of Goodfellow et al. (2016) for an in-depth discussion of back-propagation in convolutional neural networks.

Charnock & Moss (2017) used recurrent neural networks (comparing the traditional RNN, with LSTM and GRU) to classify simulated supernova light curves from the Supernova Photometric Classification Challenge (Kessler et al., 2010). The data was preprocessed so that at each time-step, the inputs to the recurrent neural network were the flux and corresponding errors in four photometric bands (*griz*), the position of the supernova on the sky, dust extinction, and host galaxy photometric redshift. Results showed that classification performance with recurrent neural networks improves with the size of the training set. The use of recurrent neural networks allows for classification of supernovae with few light curve observations without needing the full light curve.

Möller & de Boissière (2020) used recurrent neural networks combined with Bayesian inference (where a prior distribution of network weights is assumed, and a posterior distribution of weights is obtained during training) to classify a set of simulated supernova. They simulated their own set of light curves similar to that of the SPCC dataset, using more than $900,000$ light curves to train and test their classifiers. The inputs to the recurrent neural networks were the fluxes and corresponding errors in four photometric bands (*griz*), time step since the first observation of the light curve, a vector representing the filter, and also host galaxy redshift (both spectroscopic and photometric estimates). The Bayesian implementation of recurrent neural networks provided good classification accuracy for a number of supernova classification tasks (eg. Ia/non-Ia, Ia/Ibc/II, and more detailed classification by spectroscopic type), and also good performance for early classification before supernova peak brightness. When comparing models with and without host galaxy redshift information, the performance improved when redshift information was included.

The GRU recurrent neural network was used by Muthukrishna et al. (2019) to classify a set of simulated Zwicky Transient Facility (ZTF) light curves, and is different to the previous two examples in that classification was done for general transients (including non-supernovae transients). Linear interpolation is used to fill in observational gaps in

two photometric bands (g and r), and at each time step the recurrent neural network is given the fluxes in the two bands, the host galaxy redshift, and dust reddening. The classifier was able to achieve good classifications for most transients within two days of a trigger alert, which improves as more observations are included.

The use of neural networks allows for the processing of different length light curves, and early transient classification, which allows for timely follow-up with ancillary facilities to obtain additional photometry and spectroscopy. These early time observations are important for understanding the physical nature of transient events.

2.6 Overfitting and underfitting

The aim of machine learning is to be able to train an algorithm or a neural network on a set of training data, and have it be able to generalise well on unseen *test* data. During training, we want to minimise a loss function to find the best model parameters to generalise on test data. We refer to this as the *training error*. What is really needed is to also minimise the error on the test data, or the *test error*. There are two criteria to be considered when using machine learning to solve tasks: have the training error be small as possible, and have the difference between the training error and test error be as as small as possible.

In the first criterion, if the training error is large, then the problem is *underfitting*. In the second criterion, if the training error is small but the difference between the training error and test error is large, than the problem is *overfitting*. A number of techniques can be used to prevent both underfitting and overfitting, and these are discussed in the use of deep learning for transient classification in Chapters 4 and 5.

2.7 Metrics of performance

Once an algorithm or neural network has been trained on the training set, we want evaluate its performance on the test set. We can quantify this by using a number of metrics of performance.

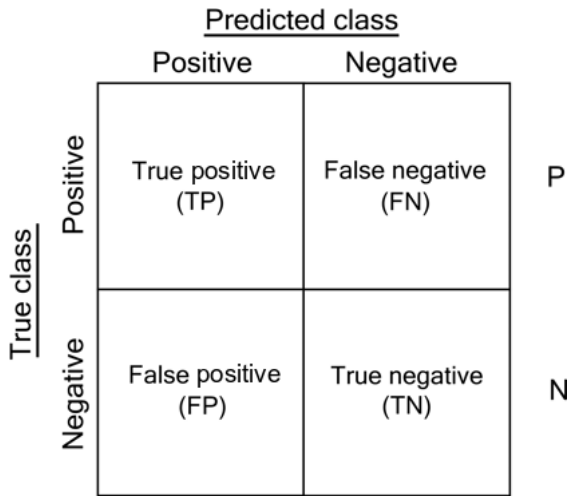


Figure 2.13: A confusion matrix representing the possible outcomes in binary classification for a positive and negative class. Class labels along the horizontal axis are the labels predicted by the classifier, and class labels along the vertical axis are the true labels. The total number of all true positives is denoted by P and the total number of all true negatives is denoted by N.

For a binary classification task where one class is positive and the other class is negative, there are four possible outcomes when evaluating the performance of a classifier. If an example from the positive class is classified as positive, then the result is a *true positive*; if it is classified as negative then the result is a *false negative*. If an example from the negative class is classified as negative, the result is a *true negative*; if it is classified as positive then the result is a *false positive*. Figure 2.13 shows a confusion matrix that represents the possible outcomes from a classifier in a binary classification task. There a number of commonly used classification metrics that can be calculated from the confusion matrix. The accuracy of a classifier is

$$\text{Accuracy} = \frac{TP + TN}{P + N}. \tag{2.28}$$

The F_1 score of a classifier is

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}. \tag{2.29}$$

The true positive rate (TPR) or the recall, is

$$\text{TPR} = \frac{\text{Correctly classified positives}}{\text{Total positives}} = \frac{TP}{TP + FN} = \frac{TP}{P}. \tag{2.30}$$

The positive predictive value (PPV) or precision, is

$$\text{PPV} = \frac{\text{Correctly classified positives}}{\text{Total predicted positives}} = \frac{TP}{TP + FP}. \tag{2.31}$$

The false positive rate (FPR) or the false alarm rate is

$$\text{FPR} = \frac{\text{Incorrectly classified negatives}}{\text{Total negatives}} = \frac{FP}{FP + TN} = \frac{FP}{N}. \tag{2.32}$$

When dealing with imbalanced data (where there are disproportionately more examples of one class than others), metrics such as accuracy and F_1 score are sensitive to the class distribution in the dataset. For example, a classifier predicting on a test set that contains 99% positive examples and 1% negative examples can achieve an accuracy of 99% by predicting all examples as positive. Consider the confusion matrix in figure 2.13. The proportion of positive to negative examples is a relationship between the top row (positive) and the bottom row (negative). Metrics that are calculated using values from both rows (i.e. accuracy and F_1 score) will be sensitive to class imbalance. Alternative metrics for classification are Receiver Operating Characteristics (ROC) graphs and the area under the ROC graph (AUC). The ROC is based on the TPR and FPR, where each

metric is a ratio calculated from values along a row of the confusion matrix, and hence is insensitive to class imbalance. ROC graphs plot the TPR on the y-axis and the FPR on the x-axis.

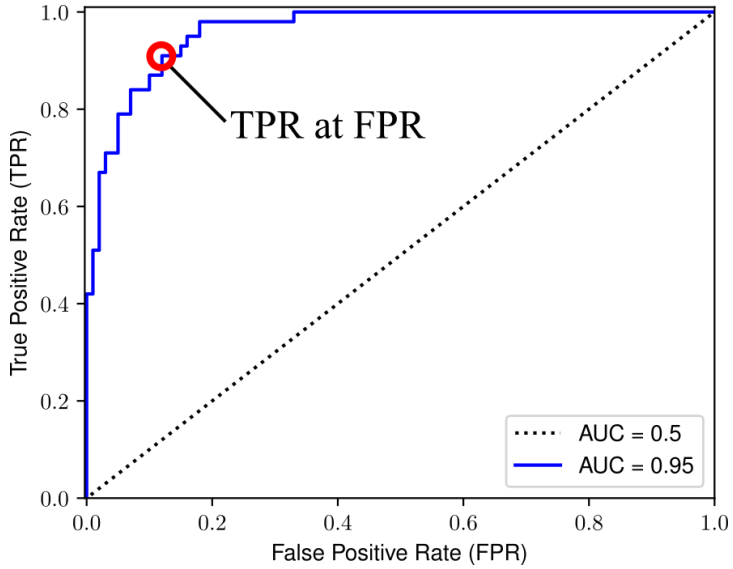


Figure 2.14: Two example ROC curves, with corresponding AUC scores. The blue ROC curve has a higher AUC score than the dotted black line, indicating that it is the better classifier. This is because it achieves a higher true positive rate (TPR) for a given false positive rate (FPR), as shown in the diagram.

A ROC graph shows the trade-off between true positives and false positives - an ideal classifier will return all true positives and no false positives. A classifier that returns discrete classifications (i.e. one that predicts the class labels) will produce a single pair of values for TPR and FPR, and a corresponding point in ROC space. A classifier that returns probabilities can produce a range of TPR and FPR values by varying a threshold: if the probability is above the threshold, then the classifier returns a positive classification, if it is below the threshold then the classifier returns a negative classification. Since different threshold values produce different TPR and FPR values, it is possible to plot a curve in ROC space by varying the threshold.

In order to compare different classifiers, it is convenient to have a single score that represents classifier performance. Since the ROC curve is in two-dimensional space, the

AUC is a fraction of the total area of ROC space, and will always have a value between 0 and 1. For an in-depth discussion of ROC analysis, see Fawcett (2006). In Figure 2.14 we show two example ROC curves and their corresponding AUC scores.

These metrics can extended to a multi-class classification problem: the confusion matrix for n classes becomes an $n \times n$ matrix with the diagonal entries representing correct classifications and the off-diagonal entries representing incorrect classifications. A method for plotting ROC graphs for multiple classes is to plot ROC graphs for each class, treating one class as the positive, and all others as negative. A formulation of the AUC for multi-class classification that is insensitive to imbalanced data was derived by Hand & Till (2001), which calculates the unweighted mean of the pairwise AUC over all classes. Although this formulation of the AUC is insensitive to class imbalance, it is not straightforward to visualise the ROC space with this method. Nevertheless, the pairwise AUC metric is useful to evaluate the performance of multiple classification models during hyperparameter optimization.

Chapter 3

Photometric classification for the Gravitational-wave Optical Transient Observer with machine learning

3.1 Introduction

With the advent of large synoptic surveys such as the Zwicky Transient Facility (ZTF; Bellm 2018), Pan-STARRS (Kaiser et al., 2010), and the Catalina Real-Time Transient Survey (CRTS; Drake et al. 2009), the field of transient astronomy has been greatly advanced. Future surveys such as the Vera Rubin Observatory Legacy Survey of Space and Time (LSST) (Ivezić et al., 2008, 2019) hope to further revolutionize this field by conducting repeat observations of the southern sky over ten years. Contemporary and future surveys now face new challenges: the large volume of data collected exceeds the human capacity for manual processing. The use of machine learning is becoming more prevalent in astronomy to overcome this challenge.

The large number of images obtained through these surveys exceeds human capability for manual identification and classification of sources. Errors in difference imaging pipelines can introduce artifacts which may be flagged as real time-varying sources, creating the task of separating real sources from 'bogus' subtraction artifacts (Bloom et al., 2012). There is also the challenge of determining interesting transients from more common objects such as asteroids or transients belonging to already well-known classes. The use of machine learning for automated discovery and classification has seen rapid growth in the past decade to alleviate these issues.

Before a machine learning algorithm can be employed a necessary step is feature extraction, where the features that encapsulate class membership are extracted from the data. A good feature set is crucial as a machine learning classifier needs to learn the proper feature-class relationship to perform effective classification (Brink et al., 2013).

In this chapter, we present the application of a machine learning algorithm, a random forest classifier, in the classification of light curves from the Gravitational-wave Optical Transient Observer (GOTO) survey.

GOTO is a survey designed to search for the optical counterparts to gravitational

waves, situated at the Roque de los Muchachos Observatory on La Palma, Canary Islands (Steeghs et al., 2021) (see Chapter 1 for a review of the GOTO survey). While not searching for gravitational wave optical counterparts, GOTO operates in an all-sky survey mode, observing a large number time-varying sources to serve a multitude of science goals (such as studying stellar variability, identifying supernovae for follow-up observations to constrain progenitor scenarios, and studying active galactic nuclei).

In section 3.2, we introduce the dataset of GOTO light curves. The feature extraction and data augmentation steps are outlined in sections 3.3 and 3.4. The random forest classifier used to classify GOTO light curves is discussed in sections 3.5 and 3.6. We provide a discussion of the results in section 3.8 and conclude with section 3.9.

3.2 GOTO data

Light curves for objects were created using photometry obtained from the GOTO difference imaging pipeline in the GOTO L filter, between February 2019 - October 2019. Objects with at least three epochs of observations were used to form the dataset, where an epoch is defined as an observation in a single night. A minimum of three epochs are required to be able to calculate time-series features from the light curves for classification. Figure 3.1 shows some summary statistics of light curves in the dataset. The typical cadence for the light curves in the dataset is roughly 7 days.

Objects in the dataset were cross-matched to identify sources that have been observed by other groups and given a classification. First, the sources were cross matched to the SIMBAD astronomical database (Wenger et al., 2000) and then to the Transient Name Server [1]. For the SIMBAD and TNS cross-matching, sources were matched to within 2.5 arcseconds. The classifications given by the cross-matching were assumed to be the true classifications for the observed sources. Next, a search for any nearby galaxies for each

[1] https://wis-tns.weizmann.ac.il/

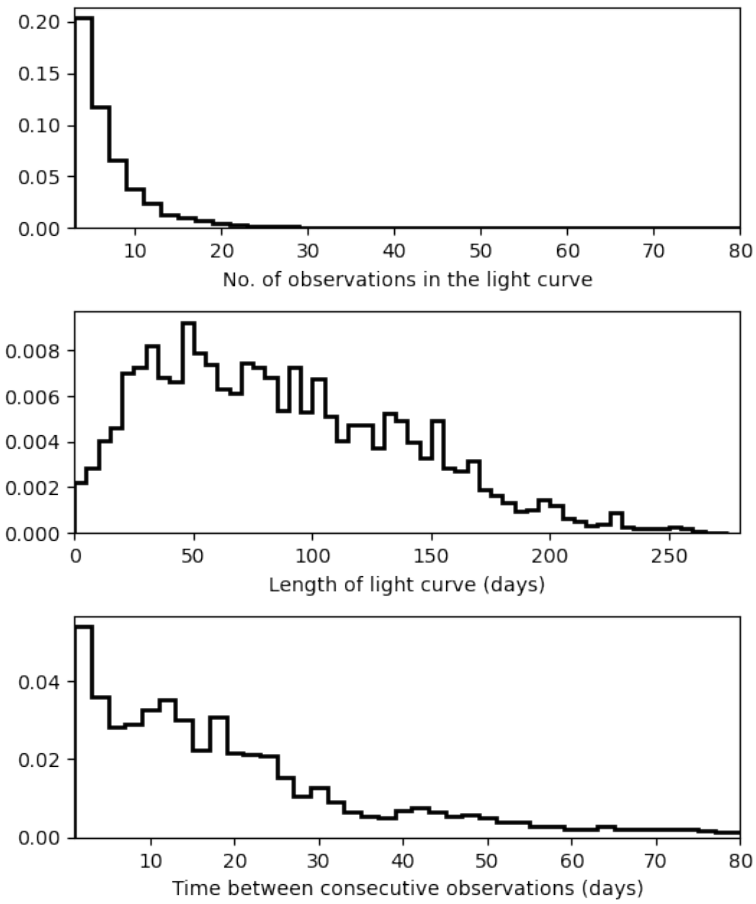


Figure 3.1: Histograms showing the number of observations over all objects in the GOTO dataset in a light curve (top), the length of the light curve in days (middle), and the time between consecutive observations in a light curve (bottom)

source was done, to identify if the source had a host galaxy. This was done by searching the NASA Extragalactic Database (NED) [2] for any galaxies within 1 arcminute of the source. Where possible, if a redshift for a source was available from the SIMBAD or TNS cross-matching, galaxies that lie within 1 arcminute with similar redshifts to the source were counted as a 'host galaxy'. Otherwise, the nearest galaxy within 1 arcminute was assumed to be a potential 'host galaxy'.

The final GOTO dataset consisted of $10,200$ sources. The sources were grouped into six different classes:

[2]The NASA/IPAC Extragalactic Database (NED) is funded by the National Aeronautics and Space Administration and operated by the California Institute of Technology, https://ned.ipac.caltech.edu/

1. **Long-period variables** (LPV): variable stars with periods longer than that of other types of variables, such as Mira stars.

2. **RR Lyrae stars** (RR): a class of periodic variable star named after the prototypical star RR Lyrae.

3. **Eclipsing binaries** (EB): a stellar binary system whose light curves show periodic variation due to one star eclipsing the other.

4. **Cepheid variables** (CEP): a class of periodic variable star with a well known pulsation period and luminosity.

5. **Cataclysmic variables** (CV): a binary system composing of a white dwarf star accreting from a companion star.

6. **Supernovae** (SN): Sources with spectroscopic confirmation of being a supernovae that were reported on TNS. This includes all subtypes of supernovae such as type Ia, Ibc, and II.

Classifications for long-period variables, RR Lyrae stars, eclipsing binaries, cepheid variables, and cataclysmic variables were obtained following the SIMBAD object classification scheme. The classifications were made to divide the data into distinct astrophysical types of objects, and so that each class had enough samples to allow the classifier to learn the differences between the different classes. Table 3.1 shows the distribution of objects in the different classes.

The dataset was then split into a training and test set. 70% was used for the training set, and 30% for the test set. Each split was done to preserve the same proportion of samples in each class, so that the training and test set had the same class distributions.

Looking at the number of samples in each class in Table 3.1, it is apparent that some classes have disproportionately more samples than others; there are many more samples for some of the variable and eclipsing binary classes than Cepheids, cataclysmic

Class	No. of samples	Training	Testing
LPV	3,958 (38.8%)	2,802	1,156
RR	3,232 (31.7%)	2,235	997
EB	2,728 (26.7%)	1,902	826
CEP	99 (1%)	70	29
CV	92 (0.9%)	64	28
SN	91 (0.9%)	67	24

Table 3.1: The number of samples in the dataset for each class: long-period variables (LPV), RR Lyrae stars (RR), eclipsing binaries (EB), cepheid variable (CEP), cataclysmic variables (CV), and supernovae (SN). In the second column, the number of samples for each class is also shown as a fraction of the total dataset.

variables, and supernovae. The dataset is imbalanced, which makes training a classifier difficult since the classifier may be biased towards classes that have more samples. We discuss methods to deal with imbalance datasets in section 3.4

3.3 Feature extraction

3.3.1 Features

To classify objects belonging to separate classes, a classifier needs to use *features*. Features are metrics derived from the data such as summary statistics (e.g. median, range), properties of the objects (e.g. mass, luminosity), or some form of dimensionality reduction done on the raw data (e.g. principal component analysis). During the training process, a classifier will attempt to discover a feature-class relationship, and identify features that discriminate between different classes of labelled objects in the training set. To make a prediction on new objects (i.e. the test set or unlabelled objects), the classifier will look at the features of the unlabelled sample, and make a prediction based on its knowledge of the feature-class relationship in the training set.

To classify GOTO light curves, two sets of features are used, The first are time-series features derived from the light curves that attempt to capture information about the

shape of the light curves. The second are contextual features that do not vary with time, such as the position of the source in the sky.

For the time-series features, Feature Analysis for Time Series (FATS) (Nun et al., 2015) features are used. FATS was designed for the analysis and feature extraction of astronomical time-series data. Standard time series features from the FATS library and additional custom features were used to extract features from the training and test sets. The python implementation of the FATS library `feets` (Cabral et al., 2018) was used to do the feature extraction. The time-series features used are listed below. More detailed explanations of the features can be found in Nun et al. (2015) and Richards et al. (2011).

- **Linear trend**: The gradient of a linear fit to the light curve.
- **Standard deviation**: The standard deviation of magnitudes in the light curve.
- **Mean**: The mean of the magnitudes in the light curve.
- **Mean error**: The mean of the error in magnitudes in the light curve.
- **Mean variance**: The ratio of standard deviation to the mean of magnitudes.
- **Magnitude range**: The range of the magnitudes in the light curve.
- **Peak magnitude**: The brightest magnitude measurement in the light curve.
- **Amplitude**: The half of the difference between the median of the maximum 5% and the median of the minimum 5% magnitudes.
- **Median buffer range percentage**: The fraction of points within 10% amplitude of the median magnitude.
- **Above 20th magnitude**: The number of magnitudes brighter than 20th magnitude in the light curve.

- **Range of a cumulative sum**: The range of the cumulative sum of the light curve.

- **Auto-correlation function length**: For a series of observations $m_1, m_2, ..., m_T$ with mean $\bar{m}$, the sample lag h auto-correlation is given by

$$\hat{\rho}_h = \frac{\sum_{t=h+1}^{T}(m_t - \bar{m})(m_{t-h} - \bar{m})}{\sum_{t=1}^{T}(m_t - \bar{m})^2}. \tag{3.1}$$

  The auto-correlation function length is defined as the lag value h where $\hat{\rho}$ becomes less than e^{-1}.

- **Median absolute deviation**: The median discrepancy of the data from the median, defined as $\text{median}(|m_i - \text{median}(m)|)$ for $m = m_1, m_2, ..., m_i$.

- **Percent amplitude**: The largest percentage difference between the minimum or maximum magnitude and the median.

- **Median minimum difference**: The difference between median magnitude and minimum.

- **Median maximum difference**: The difference between median magnitude and maximum.

- **Skew**: The skew of the magnitudes in the light curve.

- **G-skew**: A median-of-magnitudes based measure of the skew, where the G-skew is

$$G_{\text{skew}} = m_{q3} + m_{q97} - 2m, \tag{3.2}$$

  where m_{q3} is the median of magnitudes lesser or equal than the 3-quantile, m_{q97} is the median of magnitudes greater or equal than the 97-quantile, and m is the median of magnitudes.

- **Standard deviation slope**: The standard deviation of slopes in the light curve, where slope is the change in magnitude with time between consecutive observations.
- **Interquartile range** Q_{3-1}: The difference between the third and first quartile of magnitudes.
- **Pair slope trend**: Considering time-sorted magnitudes in the light curve, the pair slope trend is the fraction of increasing first differences minus the fraction of decreasing first differences.
- **Rise time**: The time from first observation to peak, if the first observation is the peak then the value is zero.
- **Mean time spacing**: The mean of the times between consecutive observations in the light curve.
- **Standard deviation of time spacing**: The standard deviation of the times between consecutive observations in the light curve.
- **Span**: The length of the light curve in days.

For the contextual features, the galactic longitude l and latitude b of the object, and whether there was a nearby galaxy near the source determined by cross-matching, described in section 3.2 were used. The galaxy matching flag was encoded as a vector, so each object has two features representing if a nearby galaxy was detected: `host_true` and `host_false`. If an object is associated with a nearby galaxy then `host_true`= 1 and `host_false`= 0, and vice-versa. This is done since categorical features need to be represented in this format to be used with the other numerical features.

3.3.2 Features from the data

Here we discuss how some of the features introduced in section 3.3.1 encapsulate the differences between different classes. Firstly, we examine how the mean magnitude and the standard deviation of magnitudes in a light curve varies for objects in the dataset. Figure 3.2 shows a scatter plot of the mean magnitude against the standard deviation of magnitudes, with accompanying histograms.

From Figure 3.2, we can see that the mean magnitude is a useful discriminant between some of the classes. Supernovae occupy the fainter end of the mean magnitude distribution, while Cepheid variables and long-period variables occupy the brighter end of the distribution. Cataclysmc variables and RR Lyrae stars also tend to have fainter mean magnitudes, but not as faint as supernovae. Eclipsing binaries are distributed across a wide range in mean magnitude. Looking at the the standard deviation of magnitudes, all objects roughly follow the same distribution, with long-period variables and Cepheid variables showing higher standard deviations in their light curves.

Secondly, we also look at how the contextual features vary across the different classes. Figure 3.3 shows histograms of the Galactic latitude for the six classes of objects in the dataset. All classes other than supernovae tend to be observed at lower Galactic latitudes ($\lesssim 30°$) , though RR Lyrae stars and eclipsing binaries are also observed at high Galactic latitudes. Cepheid variables in particular are observed close to the Galactic plane, while supernovae are observed away from the Galactic plane. There is a dip in the histograms for long-period variables and RR Lyrae stars near the galactic plane which may be due to difficulty in resolving sources within crowded stellar fields.

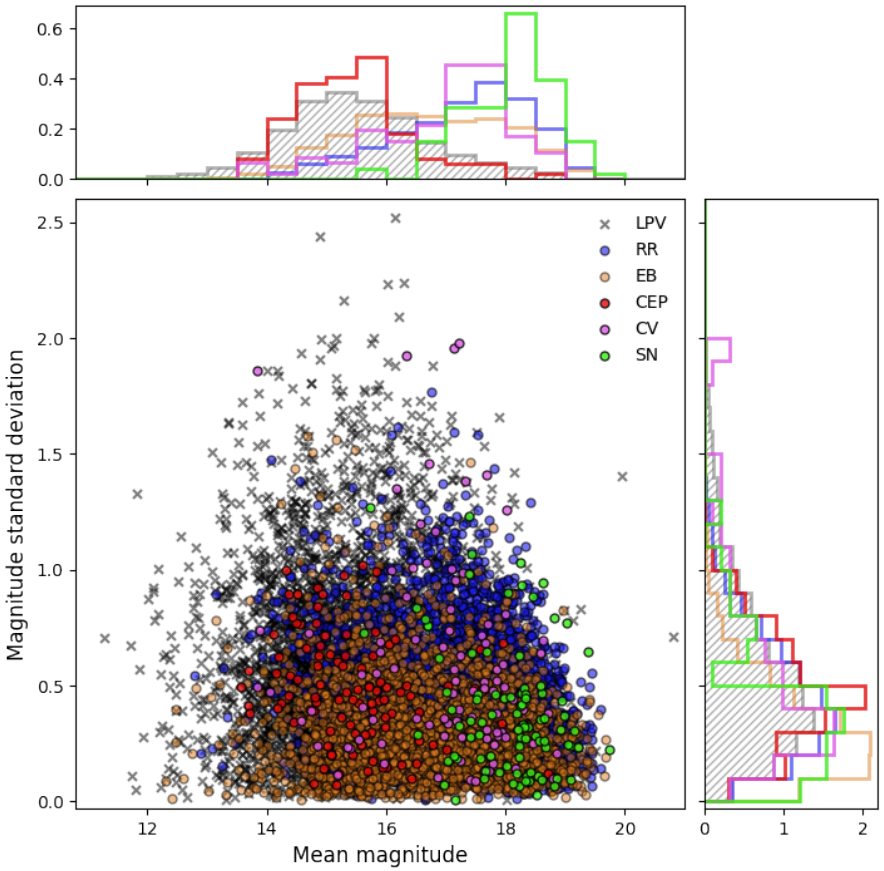


Figure 3.2: A scatter plot of the mean against standard deviation of magnitudes in a light curve, for all objects in the dataset, divided by class. The histogram for the mean magnitude (top) and standard deviation (right) are plotted alongside. The y-axes for the histograms are plotted as normalised counts.

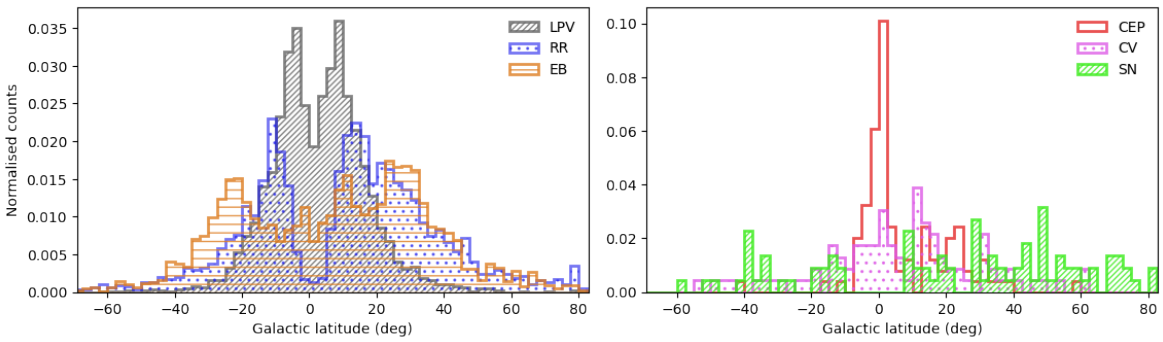


Figure 3.3: The histograms of galactic latitudes of all objects in the dataset, divided by class.

Table 3.2 shows the fraction of objects in each class that have a nearby galaxy when

cross-matched to the NED catalogue (Section 3.3.1). The majority of supernovae have an associated galaxy, while all other classes of objects have a low fraction that are associated with a galaxy.

The features derived from light curves (mean magnitude and standard deviation in magnitude in Figure 3.2) show that these two measures show some separability between distant extra-galactic objects (supernovae) and nearby galactic sources. The contextual features are very useful in differentiating between supernovae and the other classes of objects, showing a clear difference in values for supernovae from the other classes.

Class	**Fraction with a nearby galaxy**
LPV	1.3%
RR	11.1%
EB	13.2%
CEP	2.0%
CV	8.7%
SN	82.4%

Table 3.2: A table showing the fraction of each class in the dataset that are associated with a galaxy when cross-matched with the NED catalogue.

3.4 Data augmentation

3.4.1 Data challenges in time-domain surveys

There is a challenge when trying to make a classifier for a new survey: a lack of labelled data to train the classifier with and classes with large differences of the number of samples (i.e. an imbalanced dataset). To develop a classifier, labelled data is needed so that the features extracted from the data can be mapped to a corresponding class label. Typically, machine learning algorithms used for classification assume that there is an equal distribution of samples across all classes. In many real-world applications, however, the data may be imbalanced, where there are disproportionately more examples

in one class (the majority class) than in another class (the minority class). Extreme imbalance can occur when the minority class contains significantly fewer examples than the majority class. Classifiers will tend to misclassify examples from the minority class, and will be optimised to perform well on classifying examples from the majority class. In many classification tasks, the class of interest is usually the class with the smallest number of samples since the aim of classification is to find rare examples (e.g. looking for a small number of rare transients in a large dataset).

Previous work for building classifiers for new and upcoming surveys such as the ZTF and Vera Rubin LSST (e.g. Muthukrishna et al. 2019; The PLAsTiCC team et al. 2018) rely on models that emulate the observing conditions of real telescopes to generate training samples. This is convenient since it is possible to generate a desired number of examples to train a classification model with. Simulations may not, however, be representative of data obtained with real observations. Using real data to train a classifier removes the need to use models to simulate observations. The caveat is that the classifier will only have knowledge on what it is trained on, so it may struggle to characterise peculiar instances of known objects, or completely new and yet undiscovered classes of objects. In this work, we use a classifier to classify objects into well-known and established classes. The challenge in working with real data is that it is often imbalanced, and may have classes that have a small number of objects.

There are three main approaches for dealing with class imbalance in a dataset with machine learning (Krawczyk, 2016):

1. *Data-level methods*: reduce imbalance by modifying the data with resampling methods

2. *Algorithm-level methods*: modify the algorithm to reduce bias towards examples from the majority class

3. *Hybrid methods*: combine both data resampling and algorithm-level methods

Here we present a data-level method where the labelled data used to train a machine learning model is altered to deal with class imbalance. A rudimentary method of dealing with imbalance is to create multiple instances of samples from the minority class so that there are artificially more minority samples (a method known as oversampling), or inversely, removing samples from the majority class so the class distribution in the dataset is more balanced (a method known as undersampling). Although this method can be successful, the drawback is that there is either a loss of information (with undersampling) from the dataset or no real increase in information (with oversampling) in the dataset.

3.4.2 Data augmentation with SMOTE

We perform data augmentation on the training set and create new training samples by sampling in feature space with Synthetic Minority Over-sampling Technique (SMOTE; Chawla et al. 2002). The purpose of augmenting only the training set is to allow the classifier to learn from more examples of the minority classes, and to see if it is able to generalise on the unseen test set of real light curves. SMOTE has previously been used for data augmentation in classifying a small sample of supernovae from the Pan-STARRS1 survey (Villar et al., 2019) with good results.

SMOTE generates 'synthetic' samples by randomly introducing new samples along line segments originating from a class sample to its k nearest neighbours in feature space, where k is a parameter that can be tuned. The result is that the region in feature space occupied by samples from the minority class is 'filled in' by newly generated synthetic samples, forming a more robust decision boundary between classes in feature space. The limitation of this method is that it can only generate new samples based on available data, so for a class that has a small number of samples the region in feature space that it occupies may be tightly constrained. Using SMOTE on this dataset has the added

benefit of being able to consider the time-series and contextual features simultaneously during augmentation.

We use the `Imbalanced-learn` python package (Lemaître et al., 2017) to implement SMOTE for data augmentation on the training set, setting $k = 5$. The data augmentation is done so that new samples are generated for all classes other than the majority class (the class with the largest number of samples) until all classes have the same number of samples ($N = 2,802$), for an augmented training consisting of $16,812$ objects. We use both the original training set and the augmented training set to train a random forest classifier and compare the performance between the two training sets.

3.5 Random Forest for classification

The random forest algorithm uses an ensemble of decision trees, where each tree is trained on a random subset of the training data. Each split in the tree is determined by selecting criteria from a subset of all features and the final result is obtained through majority voting of all the trees in the ensemble to give a more robust classification (Breiman, 2001) (see Chapter 2 for an introduction to the random forest algorithm). For a single decision tree, the probability of an object belonging to a class k is the fraction of samples belonging to class k in a node. The random forest can return either the probability of an object belonging to a particular class (by averaging the probabilities across all trees in the forest) or the class of the object (by choosing the class with the highest mean probability).

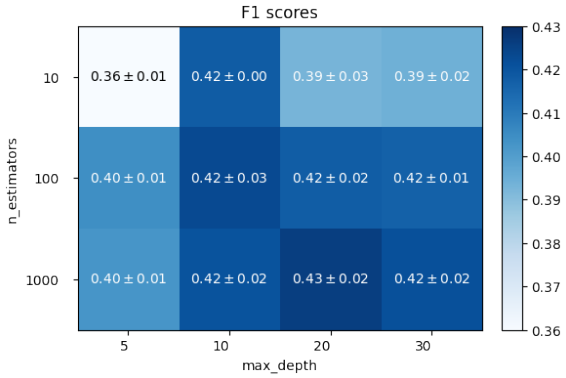
The random forest has a number of adjustable hyperparameters, parameters that are not learnt during training and are instead set outside of training. We vary two hyperparameters: `n_estimators`, the number of decision trees in the random forest and `max_depth`, the maximum number of successive splits performed in a decision tree. We vary the number of trees in the forest to see how increasing the size of the ensemble

affects classification performance, and the maximum depth to see how model complexity (i.e. how many times does a tree split the data in feature space) affects performance. We set the maximum number of features that are sampled each time a split is made as the square root of the total number of features (rounded to the nearest integer). For the classification of GOTO light curves, the random forest classifier from the Python package `scikit-learn` was used (Pedregosa et al., 2011).

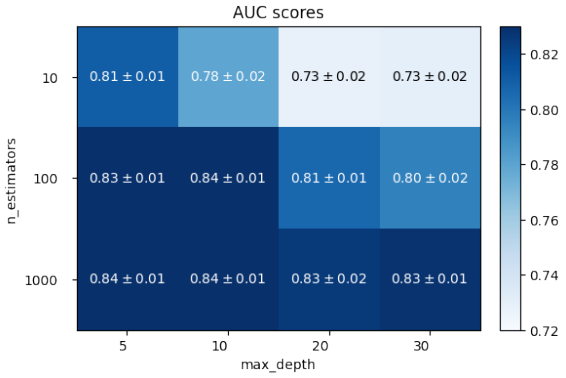
3.6 Hyperparameter optimization

We create a grid of hyperparameters, selecting values $[10, 100, 1000]$ for `n_estimators` and $[5, 10, 20, 30]$ for `max_depth`. To identify the best set of hyperparameters, we use k-fold cross validation. The training set is split into k equally size folds, and k unique groups are created by taking $k-1$ folds as the training set, and the last fold as the test set, so that in each group a different fold is used as the test set. The random forest is trained on $k-1$ folds and then evaluated on the last fold. We calculate the F_1 and AUC scores for $k = 5$ folds, and calculate the average score and take the error as the standard deviation.

We plot the F_1 and AUC scores obtained by each hyperparameter combination as a results matrix for the original training set in Figure 3.4, and for the augmented training set in Figure 3.5. We choose the best set of hyperparameters as the combination that gives the best AUC score, since AUC is a good indicator of how the purity-completeness trade-off is maximised to give the best predictions. See section 2.7 for a discussion on machine learning metrics. The best hyperparameters for the original and augmented training set are then used for a random forest classifier to make predictions on the test set.

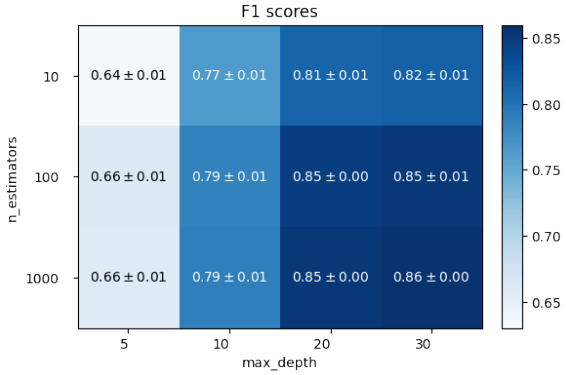


(a) F_1 score matrix, original training set

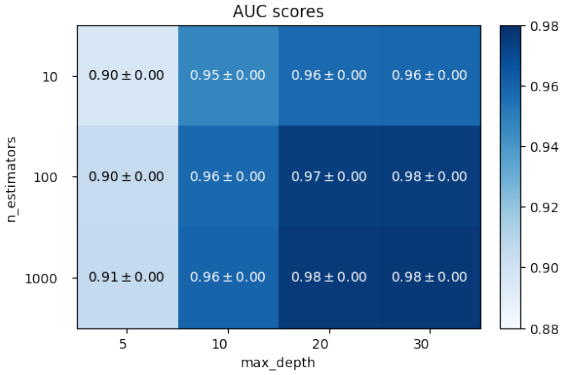


(b) AUC score matrix, original training set

Figure 3.4: Results matrix for hyperparameters, model trained on original training set. The values for `max_depth` are on the x-axis, and the values for `n_estimators` are on the y-axis.



(a) F_1 score matrix, augmented training set



(b) AUC score matrix, augmented training set

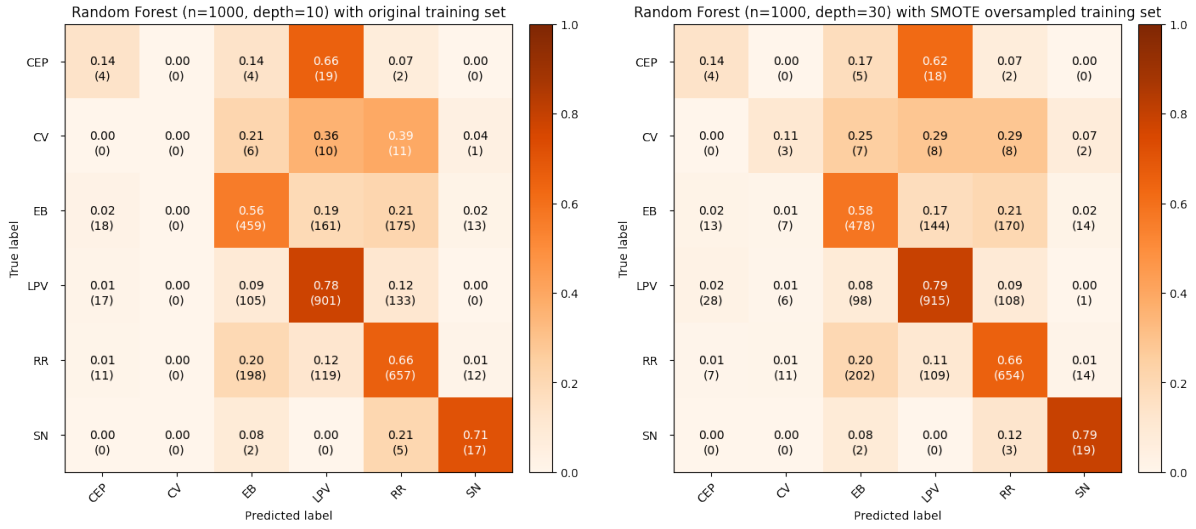
Figure 3.5: Results matrix for hyperparameters, model trained on augmented training set. The values for `max_depth` are on the x-axis, and the values for `n_estimators` are on the y-axis.

3.7 Results

Using the AUC score as a performance metric, the best set of hyperparameters for the random forest trained on the original training set are `n_estimators`$= 1,000$ and `max_depth`$= 10$, and for the random forest trained on the augmented training set `n_estimators`$= 1,000$ and `max_depth`$= 30$. We calculate the F_1 and AUC scores for both random forests when evaluated on the test set, and tabulate the results in Table 3.3, and also plot the confusion matrices in Figure 3.6.

When trained on the original training set, the random forest classifier is able to classify eclipsing binaries, long-period variables, RR Lyrae, and supernovae with accuracies ranging from 56% for eclipsing binaries to 78% for long period variables. The accuracy for Cepheid variables and cataclysmic variables is poor, with only 14% of Cepheid variables being correctly classified, and no cataclysmic variables are correctly classified at all. There is a high degree of confusion between the different classes of variable objects, with 66% of Cepheid variables being classified as long-period variables.

With the augmented training set, there is some increase in classification performance. The classifier is able to correctly classify an additional two supernovae (going from 71% to 79% accuracy), and now correctly classifies 3 cataclysmic variables (compared to none for when the original training set is used). Classification for the other classes remains largely unchanged. The F_1 score increases from 0.435 to 0.461, while the AUC score decreases from 0.843 to 0.838 when the augmented training set is used. We discuss these results in the next section.



(a) Confusion matrix, original training set (b) Confusion matrix, augmented training set

Figure 3.6: Confusion matrices for the random forest classifiers trained with the original and augmented training set, evaluated on the test set. The rows of the matrix show the fraction of correct and incorrect predictions for each class, and where incorrect predictions between classes occur. Below the fractions are the number of objects that have been predicted, in parentheses.

Model	F_1 score	AUC score
Original training set	0.435	0.843
Augmented training set	0.461	0.838

Table 3.3: The F_1 and AUC scores for the random forest classifiers trained with the original and augmented training set, evaluated on the test set.

3.8 Discussion

The use of data augmentation has increased the classification performance for the classes with the smallest number of samples (supernovae and cataclysmic variables), while retaining the same performance for the other classes. The F_1 score increases when the augmented training set is used, reflecting the increased recovery for supernovae and cataclysmic variables. The decrease in AUC score suggest that the data augmentation is creating some overlap in feature space between classes when generating synthetic samples, which may make the classification of objects near the decision boundary in features space harder to classify.

Even without data augmentation, the random forest classifier is able to accurately classify 71% of the supernovae, which suggests that the features selected for classification contained enough information to separate supernovae from the other classes. With a random forest, it is possible to calculate a score that describes how important a feature is for classification (feature importance). We use an impurity based measure for feature importance, where the importance of a feature is calculated as the total reduction of the Gini impurity (Equation 2.2) brought by that feature. A higher value for feature importance indicates that the feature is important for creating decision boundaries in feature space. We plot the feature importance for all features used when training the random forest on the original training set in Figure 3.7. We show the feature importance for the original training set since we want to examine feature importance without data

augmentation, and the fact that data augmentation does not show a significant increase in performance.

From Figure 3.7, we see that some of the most important features are the galactic latitude `b`, the mean magnitude of the light curve `mean`, and `host_false`. In section 3.3.2, we examined how some of the time-series derived features such as the mean magnitude and contextual features were different for supernovae than the other classes, reflecting their potential importance in classifying light curves. Two of the top four important features are contextual features (galactic latitude and association with a nearby galaxy), reflecting the importance of using contextual information alongside time-series information when classifying light curves.

The confusion between the variable object classes suggest that they overlap in feature space, making it difficult for the decision trees in the random forest to draw appropriate decision boundaries. This is also coupled with the fact that two of the variable object classes (Cepheid variables and cataclysmic variables) have a relatively small number of samples (< 100) compared to the other variable object classes ($> 2,500$), limiting the number of samples that the classifier is able to learn from. The small number of samples in these minority classes also make it challenging for the SMOTE augmentation to increase information in the dataset. Since SMOTE works by drawing samples along line segments between objects of the same class in feature space, if there is a small number of objects then the region of feature space occupied by objects of that class becomes constrained. This is the limitation of a small sample of labelled data available for training with a machine learning algorithm, along with class imbalance.

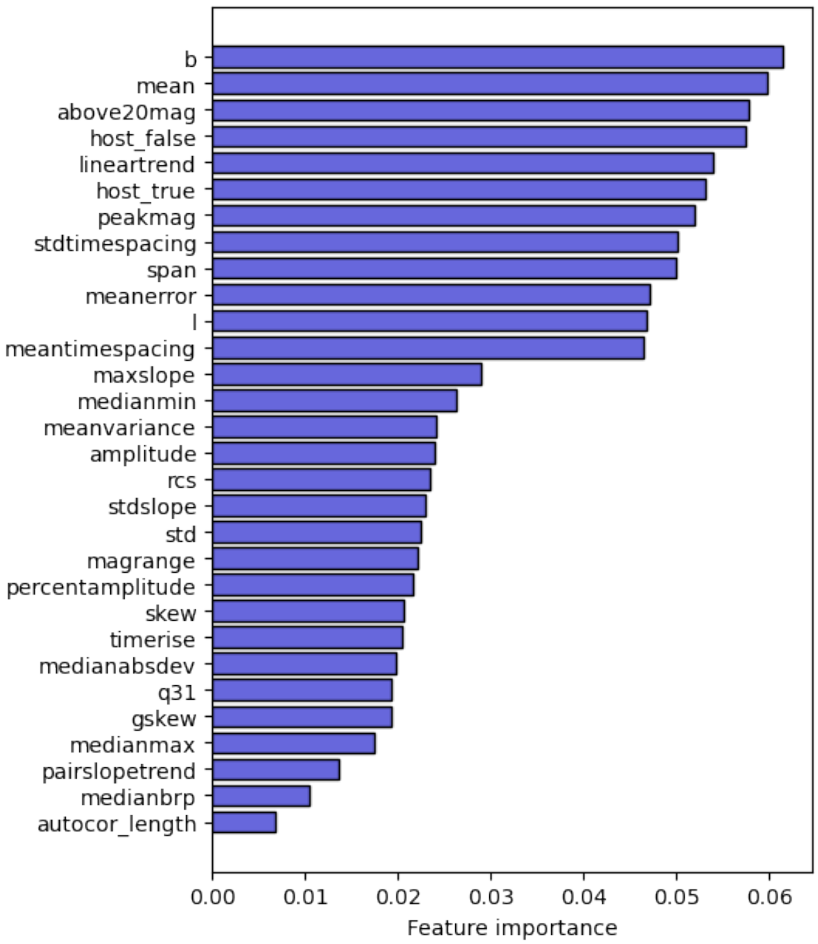


Figure 3.7: Feature importance in a random forest classifier trained with the original training set.

3.9 Conclusion

In this chapter we presented a machine learning approach to classifying light curves from the GOTO survey with a random forest classifier. We use a set of time-series features extracted from the light curves, along with additional contextual features as inputs to the classifier. The random forest identifies the best separations in feature space to split light curve of object belonging to six classes: long-period variables, RR Lyrae stars, eclipsing binaries, Cepheid variables, cataclysmic variables, and supernovae. The dataset comprising the GOTO light curves is imbalanced, with some classes having a much smaller number of samples than others. To deal with the class imbalance present

in the data, we use SMOTE as a data augmentation method and compare random forest classifiers trained on the original training set and the augmented training set.

The choice of features led to good classification accuracy for supernovae, eclipsing binaries, long-period variables, and RR Lyrae stars, but poor performance for Cepheid variables and cataclysmic variables. With data augmentation, there was a slight increase in classification performance for supernovae and cataclysmic variables. The small class sizes and class imbalance in the dataset makes it challenging to classify minority classes even with data augmentation. The need for a representative training set, one that accurately represents the region of feature space occupied by unlabelled samples is important for making a classifier that is able to generalise well.

At this stage, GOTO is still a new survey, and the effort to create a representative labelled training set for training classifiers is an ongoing process. As the survey progresses, more labelled samples will become available, as well as more observations in additional filters that will increase the amount of information available to be extracted from objects to aid classification efforts. The method presented in this chapter utilises features extracted from full light curves that have multiple observations over an extended period of time. To better identify interesting transients early in their photometric evolution (e.g. a supernova before reaching peak brightness), a method to be able to classify light curves with few observations in a short amount of time is required. Methods that utilise deep neural network architectures such as recurrent neural networks (e.g. Muthukrishna et al. 2019; Carrasco-Davis et al. 2019; Möller & de Boissière 2020) that are able to provide 'real-time' classifications on light curves as new observations are made would be useful for this goal.

Chapter 4

Photometric classification for the Gravitational-wave Optical Transient Observer with recurrent neural networks

4.1 Introduction

In this chapter, we improve upon the work presented in Chapter 3. Both feature-based machine learning and deep learning methods have been used to classify astronomical objects into distinct classes. In contrast to feature-based machine learning models, deep learning models are able to learn salient features from the data, and do not require a feature extraction step prior to training. We provide a review of how machine learning and deep learning have been used for classification in time-domain astronomy in Chapter 2.

Recently, the growing number of surveys across the world has facilitated a new era of multi-messenger astronomy. Observing a single event across multiple wavelengths and through different detectors allows for a deeper understanding of the physics behind transient phenomena. In 2017, the gravitational wave signals of a binary neutron star merger (designated GW170817) were detected by the Advanced LIGO and Advanced Virgo gravitational-wave detectors (Abbott et al., 2017). Follow-up observations across the electromagnetic spectrum led to the discovery of the kilonova AT2017gfo, thought to be powered by the radioactive decay of r-process nuclei following a binary neutron star merger (Abbott et al., 2017; Chornock et al., 2017; Coulter et al., 2017; Drout et al., 2017; Shappee et al., 2017; Smartt et al., 2017; Villar et al., 2017).

In search of the optical counterparts to gravitational wave signals is the Gravitational-wave Optical Transient Observer (GOTO) (Steeghs et al., 2021) survey. When GOTO receives an alert for gravitational wave or gamma-ray burst detection from other facilities, it will rapidly begin observing the localised region of sky to look for optical counterparts (Dyer et al., 2020). See Chapter 1 for a discussion on the GOTO survey.

While not in gravitational wave follow-up mode, GOTO conducts an all-sky survey to search for transient and variable sources. The next step in object classification is classifying real discoveries into distinct astrophysical types. Providing object classifica-

tions for real objects will be useful for the GOTO collaboration in helping to identify interesting targets for follow-up and further science goals.

Effective classification by machine learning and deep learning models rely on good representation of the labelled classes in the dataset to learn class separability across the labelled objects. In real-world applications, the data will typically contain one or more classes that have more examples than other classes. This type of data is referred to as 'imbalanced data', and it poses a difficulty for classification as models will be biased towards the class where there are many more examples to learn from. Here, we present a recurrent neural network (RNN) classifier to classify objects discovered by GOTO.

In section 4.2, we provide an overview of the data used to train and test the RNN classifier. In section 4.3, we introduce the RNN architecture, the class imbalance problem, and the approach taken to deal with an imbalance dataset. In section 4.4, we outline the training process, and in section 4.5, we discuss the performance of the classifier and how contextual information plays a role in how models learn to classify. We discuss how the work presented in this chapter can be improved upon, and conclude with sections 4.6 and 4.7.

4.2 Data

The dataset used to train the classifier is obtained from GOTO observations spanning the period of 20 March 2019 to 4 November 2020, during the GOTO prototype phase. Light curves of objects observed during this period are created using photometric measurements derived from the GOTO difference imaging pipeline (Steeghs et al., 2021) in the L filter (see Chapter 1 for a summary of the GOTO survey). The catalog of GOTO objects is then cross-matched to a number of external catalogs to determine objects that have also been observed by other telescopes and surveys, and to obtain classification labels. The list of external catalogs include:

1. The American Association of Variable Star Observers (AAVSO) International Variable Star Index (Watson et al., 2006)

2. The Veron Catalog of Quasars & AGN, 13th edition (Véron-Cetty & Véron, 2010)

3. The Transient Name Server [1]

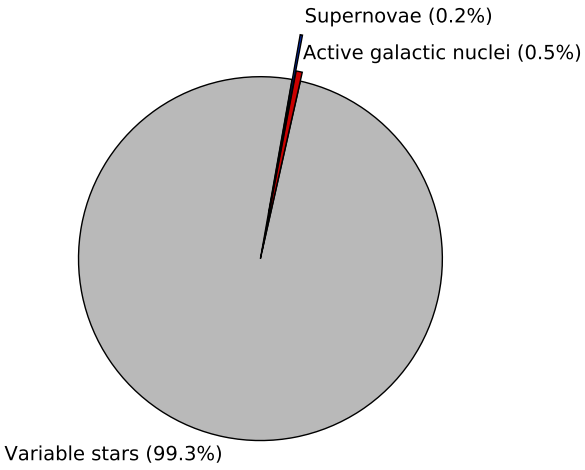


Figure 4.1: A pie chart showing the class distribution in the GOTO dataset, illustrating the high degree of class imbalance present in the data

Additionally, the GOTO objects are also cross-matched against the Galaxy List for the Advanced Detector Era (GLADE) (Dálya et al., 2018) galaxy catalog to identify if there is a nearby galaxy associated with the object. In total, the dataset comprises $99,201$ labelled objects, and are split into three broad classes: variable stars (VS), active galactic nuclei (AGN), and supernovae (SN). The dataset is heavily imbalanced with 99 % $(98,457)$ of labelled objects belonging to the variable star class, and only 543 and 201 belonging to the active galactic nuclei and supernovae classes, respectively (Fig. 4.1). The largest class (VS) contains almost 500 times more examples than the smallest class (SN).

Within the astronomical taxonomy for transient and variable sources, there are a wide range of classification schemes: classifying variable stars by the physical mechanism that

[1] https://wis-tns.weizmann.ac.il/

causes variability (eclipsing binaries, RR Lyrae stars, Cepheids) and classifying supernovae by spectroscopic features (Type Ia, Ib/c, II). The use of 'super-classes' that group together distinct types of objects simplifies the classification task while still providing clear classifications. For rapidly evolving objects that would benefit from early time follow-up observations such as supernovae, it is beneficial to separate them from objects that show photometric variation over longer timescales with just a few observations. Providing a more general classification in real-time acts as a 'first-pass' classification, and further classification into more specific sub-types can be done when additional observations become available.

Figure 4.2 shows the distribution of mean magnitudes and standard deviation in magnitudes, where each statistic is calculated from all measurements in a light curve.

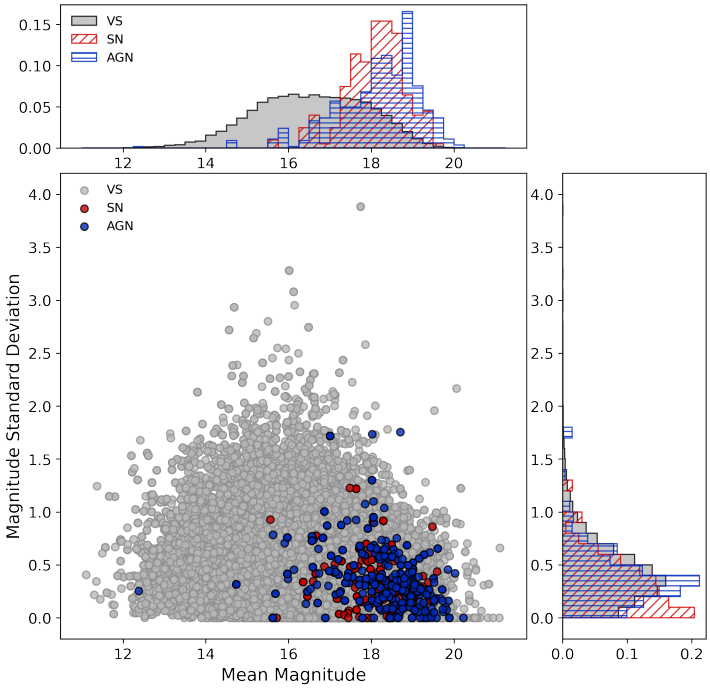


Figure 4.2: A scatter plot of the mean magnitude of light curves against the standard deviation in magnitudes of light curves for variable stars (VS) in grey, supernovae (SN) in red, and active galactic nuclei (AGN) in blue. In the top and right panels are histograms showing the distribution of mean magnitudes (top) and standard deviation in magnitudes (left). The histograms are plotted as normalised counts, for each class.

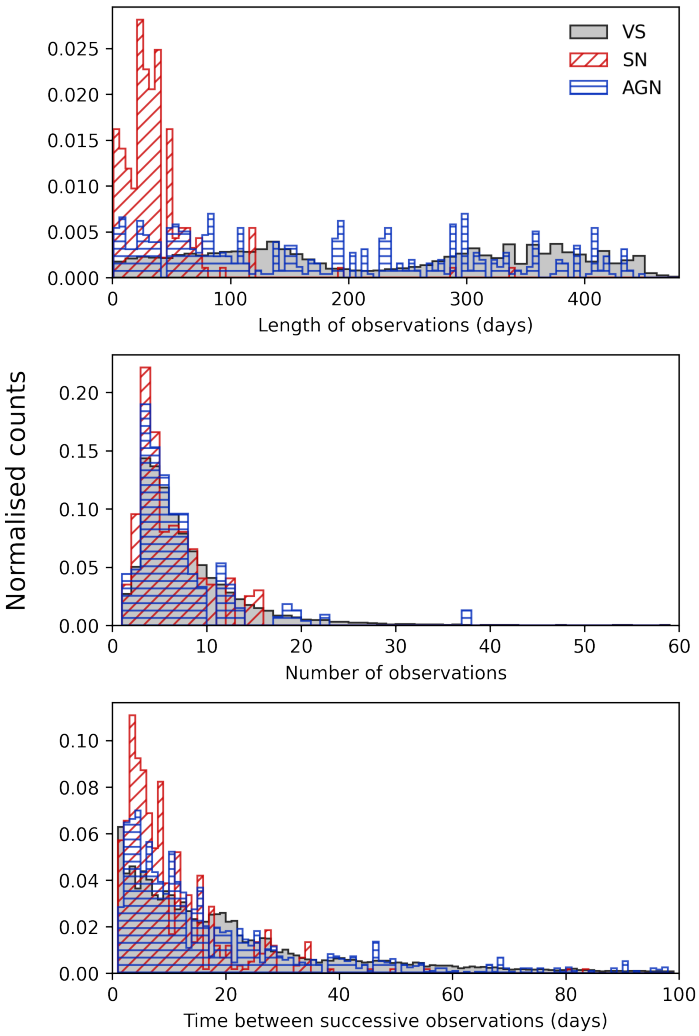


Figure 4.3: Normalised histograms showing different properties of the light curves in the dataset, separated by class. From top to bottom: the time between the first and last observations of the light curves (length of observation), the number of observations in the light curves, and the time between successive observations over all light curves.

VS have a broad range of magnitudes, while SN and AGN tend to be fainter compared to VS. Figure 4.3 summarises the light curve properties within the dataset. SN are typically observed no longer than 100 days, while VS and AGN are observed over longer timescales ($>$ 300 days). The median number of points in a light curve for the dataset is 6, and the median time between successive observations is $\sim$ 11 days. The dataset contains light curves ranging in duration from a single day to a few hundred days. The distribution of number of observations in a light curve and the time between successive observations within a light curve over all classes is fairly similar.

4.2.1 Data preprocessing

Before being used as input into the classifier, the data needs to be preprocessed. For each light curve, the time-series input matrix consists of the times of observation $\mathbf{t}$, the magnitudes $\mathbf{m}$, and the errors in magnitude $\boldsymbol{\sigma}_m$. The time is scaled to the time of the first observation, so that it starts at zero, and subsequent time steps are times since the first observation. The time-series input matrix $\mathbf{X_T}$ for a light curve with n observations is

$$\mathbf{X_T} = \begin{bmatrix} \mathbf{t} \\ \mathbf{m} \\ \boldsymbol{\sigma}_m \end{bmatrix} = \begin{bmatrix} t_1 & t_2 & ... & t_n \\ m_1 & m_2 & ... & m_n \\ \sigma_{m_1} & \sigma_{m_2} & ... & \sigma_{m_n} \end{bmatrix}. \tag{4.1}$$

For each object, the contextual information used are the galactic longitude l and latitude b in degrees, and distance in arcseconds to the nearest galaxy in the GLADE catalog d_G. Objects that have $d_G > 60$ arcseconds have their d_G set to a dummy value. The contextual information input vector $\mathbf{X_C} = (l, b, d_G)$ is used alongside the time-series input matrix $\mathbf{X_T}$ as inputs for the classifier. Figure 4.4 shows the distribution of d_G and b for all objects. AGN and SN are more commonly found to have a nearby galaxy in the GLADE catalog than VS. VS are mostly located close to the galactic plane, while AGN

and SN are usually found $\gtrsim$ 10° away from the galactic plane.

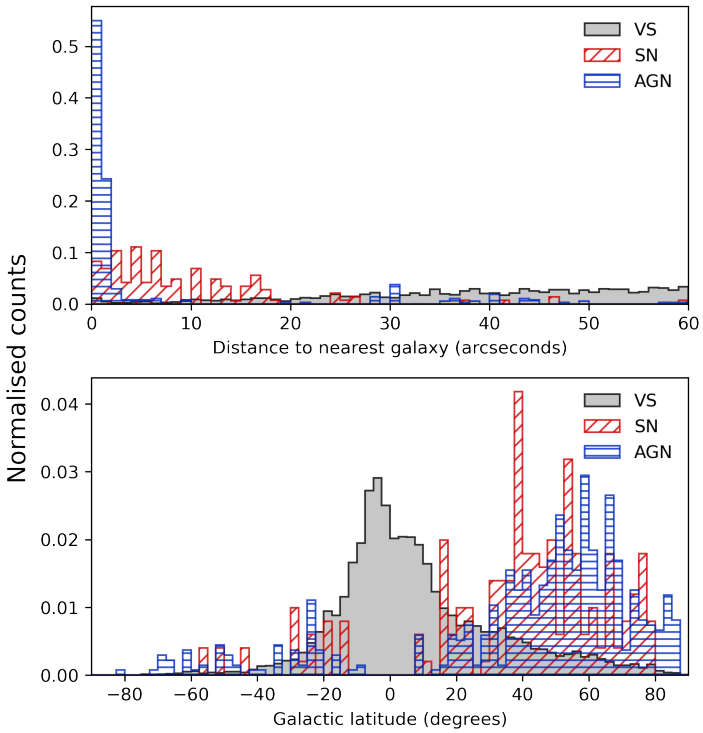


Figure 4.4: Normalised histograms showing how the distance to the nearest galaxy in the GLADE catalog and galactic latitude for all objects in the dataset varies by class.

4.3 Model

4.3.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of neural networks that operate on sequential data. The sequential data can take the form $[\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_\tau]$, where $\boldsymbol{x}_t$ is a vector with time step index t running from $t = 0$ to $t = \tau$ for a sequence with τ time steps. The time step index t does not necessarily have to represent the passage of time, but can also denote the position of a vector in the sequence. RNNs make use of 'hidden states' (similar to the hidden layers of a deep neural network) that incorporate information from the hidden state at the previous time step. For an in-depth review of recurrent neural

networks, see Chatper 2.

In the work presented in this chapter, both LSTM and GRU networks are used to process light curve data and produce a set of class probabilities. It should be noted that the values returned by the classifier are not true probabilities, rather they are scores given to an object by the classifier that indicate the level of 'belongingness' to a certain class. In this work, the scores returned by the RNN classifiers are referred to as the class or prediction probabilities. The outputs are produced after reading in the entire light curve, along with additional contextual information.

4.3.2 Mixed input network

In order to utilise both the time-series data from the light curve and contextual information, the neural network model is formed of two branches: the first branch reads in the time-series input matrix $\mathbf{X_T}$ and the second branch reads in the contextual information input vector $\mathbf{X_C}$. Since recurrent neural networks are optimised for processing sequential data, the contextual information is fed to the model separately as opposed to together with the time-series data (e.g. as a vector of constant values), so that the RNN can extract high-level time-series features from the light curve. We use a similar approach to Muthukrishna et al. (2019) to construct the RNN branch of the network, since they find that their configuration works well for classifying light curves. Further investigation of different RNN configurations is beyond the scope of the work presented in this chapter.

The first branch acts as a standard RNN, consisting of two RNN layers, and the second branch is just an input layer for $\mathbf{X_C}$. The output of the final layer in RNN branch is concatenated with $\mathbf{X_C}$, and forms the input for two dense layers, the latter of which is connected to the final output layer. The output layer produces a list of numbers which sum to 1, which are the class probabilities of an object belonging to each of the defined classes. Figure 4.5 illustrates the model architecture. To avoid overfitting (where

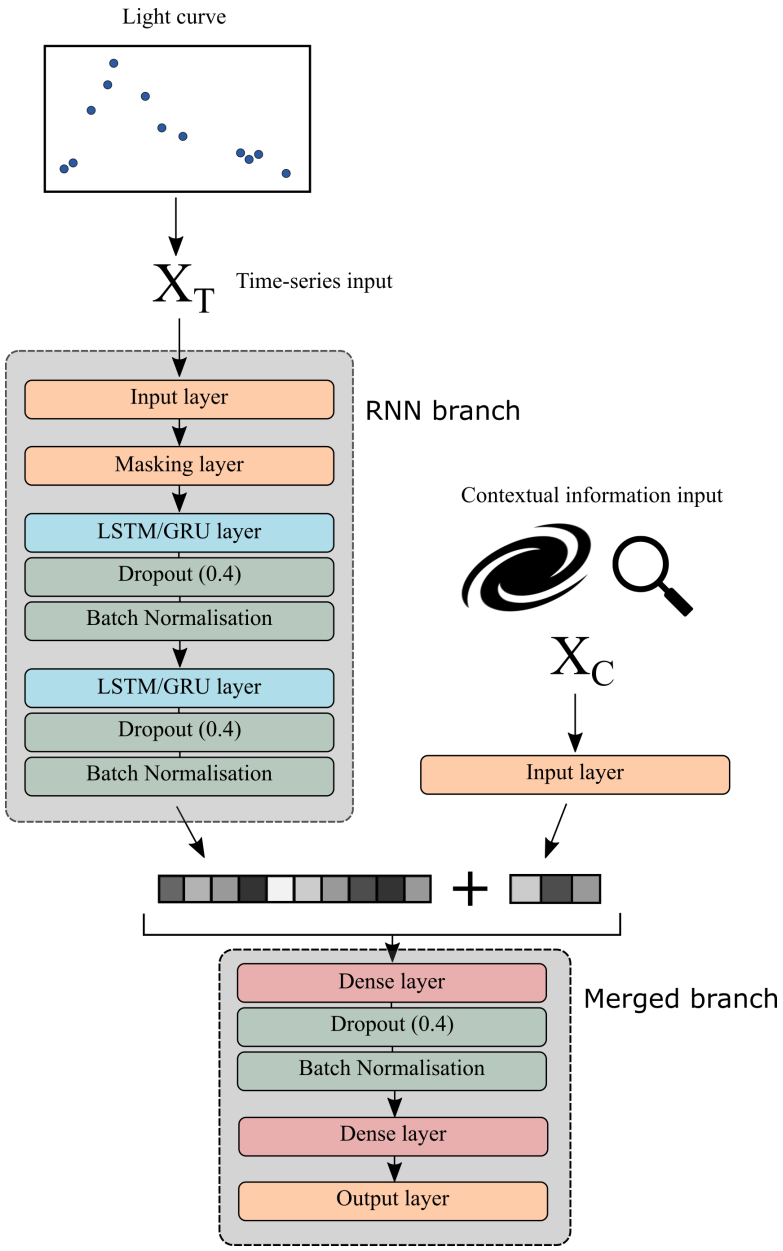


Figure 4.5: Diagram of the mixed input network. The time-series input $\mathbf{X_T}$ is passed to the RNN branch, and the contextual information input $\mathbf{X_C}$ is appended to the output of the final RNN layer, before being passed on to merged branch of the network.

a model performs well on the training set but underperforms on the test set) dropout and regularisation are used within the model. A summary of the different components of the model is presented below.

- **Masking layer**: The masking layer is placed between the input layer of the time-

series branch of the model and the first RNN layer. It applies a mask to a sequence of of time steps, where each time step refers to a column in the vector $\mathbf{X_T}$, and uses a mask value to skip time steps. Subsequent layers will ignore masked timesteps. A masking layer allows the model to process sequences with a different number of time steps.

- **Long short-term memory (LSTM) layer**: The LSTM layer takes in the masked $\mathbf{X_T}$ as input, and applies the LSTM operation. The dimension of the output n_{out} from this layer is an adjustable hyperparameter. Two LSTM layers are stacked in the RNN branch of the model: the first LSTM layer returns a sequence of outputs each with dimension n_{out}, which is passed on to the second LSTM layer that returns a single output with dimension n_{out}.

- **Gated recurrent unit (GRU) layer**: The GRU layer functions in the same way as the LSTM layer and has the same adjustable hyperparameter n_{out}, but it applies the GRU operation to the data instead. Like the LSTM layers, two GRU layers are stacked and the first GRU layer passes a sequence to the second GRU layer, which returns a single output with dimension n_{out}.

- **Dropout**: A dropout layer randomly drops input neurons and their corresponding connections during training, as a method to reduce overfitting (Srivastava et al., 2014). This forces the neurons to derive more meaningful features from the data without heavily relying on other neurons in the network. During testing, the data is passed through the network without dropout. The dropout fraction sets the fraction of input neurons that are dropped.

- **Batch normalisation**: During training the parameters for each layer change, affecting the distribution of the inputs in the preceeding layers in the network. The change in the distribution of inputs in the layers requires the layers to adapt

to the new distribution, a phenomenon known as internal covariate shift. Batch normalisation scales the inputs to a layer for each batch so that the mean value is close to 0 and the standard deviation is close to 1, reducing the impact of internal covariate shift (Ioffe & Szegedy, 2015).

- **Dense layer**: A dense layer is the simplest layer in a neural network: it consists of a fully connected layer of neurons and takes in a fixed size input. The number of neurons in the dense layer is an adjustable hyperparameter. All dense layers in the merged branch have the same number of neurons, which is set by the hyperparameter. The output layer is just a dense layer with three neurons with a softmax activation function that ensures the values returned by the output layer all sum to 1.

- **Regularisation:** Regularisation introduces a penalty term to the loss function as a method to reduce overfitting. The L2 regularisation is used, which adds a penalty term equal to the sum of all the model weights squared, multiplied by a regularisation factor λ. The regularisation factor λ sets the strength of 'weight decay' in the loss function. A larger value of λ forces the weights to have smaller values and helps to reduce overfitting (Goodfellow et al., 2016).

The LSTM and GRU layers serve the purpose of extracting meaningful features from the time-series data. The dense layers then combine the time-series representations and contextual information and further extract features from the combined features to produce a prediction. After each LSTM/GRU layer and the first dense layer, dropout is applied in an attempt to reduce overfitting, and batch normalisation is applied to reduce internal covariate shift throughout the network. We use a combination of three methods as an approach to deal with potential overfitting: dropout, batch normalisation, and regularisation in the loss function.

4.3.3 Class imbalance

In Chapter 3, we used a data-level method with a random forest classifier to deal with an imbalanced dataset. Despite the popularity of deep learning-based classifiers, there is a lack of research into dealing with class imbalance when using deep learning architectures (Johnson & Khoshgoftaar, 2019). Data augmentation for light curves of different astrophysical objects can be a laborious process, as models are needed to generate simulated examples of real observations, and multiple models may be required to simulate objects from multiple classes. In this chapter, we attempt an algorithm-level approach for dealing with class imbalance by using a focal loss function to optimise the RNN classifier, as an alternative to a data-level approach such as data augmentation.

In supervised learning, a model is trained for a prediction task by minimising a loss function. The model adjusts its internal weights to reduce the error by calculating the gradient in weight space that will minimise the loss function via gradient descent. See Chapter 2 for a discussion on loss functions and gradient descent.

For multi-class classification, the cross-entropy loss is typically used. Given a multi-class problem with N classes, the cross entropy loss (CE) for an example i is

$$\mathrm{CE} = -\sum_{j=1}^{N} \delta_{ij} \log(p_{ij}), \tag{4.2}$$

where p_{ij} is the probability of example i belonging to class j, and δ_{ij} is the Kronecker delta function. The loss for the entire dataset is given by summing the loss of all examples.

The focal loss (Lin et al., 2017) addresses class imbalance by down-weighting examples from the majority class. For the same multi-class problem as above, the focal loss (FL) for an example i is

$$\mathrm{FL} = -\sum_{j=1}^{N} \delta_{ij} (1 - p_{ij})^{\gamma} \log(p_{ij}), \tag{4.3}$$

where $(1 - p_{ij})$ is the modulating factor, and γ is the parameter that adjusts the rate at which majority class examples are down-weighted. Increasing γ reduces the contribution from well classified examples to the loss, and increases the importance of improving misclassified examples (Lin et al., 2017). For a misclassified example, p_{ij} is small so the modulating factor is close to 1 and the loss is unaffected. Examples that are well classified will have p_{ij} close to 1, so the modulating factor is small, and the loss from well classified examples will be down-weighted. The focal loss is equivalent to the cross entropy loss when $\gamma = 0$. In practice, a weighted version of the focal loss can be used, which Lin et al. (2017) find to perform better than the unweighted focal loss (eq. 4.3) for imbalanced classification tasks:

$$\mathrm{FL} = -\sum_{j=1}^{N} \delta_{ij} \alpha_j (1 - p_{ij})^{\gamma} \log(p_{ij}), \tag{4.4}$$

where α_j is a weighting factor for class j. In this work, the weighting factor is given by

$$\alpha_j = \frac{1}{n} \times \frac{N}{N_j} \tag{4.5}$$

where N is the total number of examples in the training set, N_j is the number of examples in class j in the training set, and n is the total number of classes which in this case is $n = 3$. For all the above loss functions, the best case is a loss value of zero.

4.4 Method

The GOTO dataset is split into 70% for training, and 30% for testing. Of the training set, 30% is set aside for validation. The data are split so that the training, validation, and test sets all have the same proportions of VS, AGN, and SN. This is done so that there are a good number of samples from each class in the validation and test set so we can evaluate classification performance on the classes with fewer examples (AGN and SN). A

Hyperparameter	**Value**
Batch size	128
Learning rate	1×10^{-4}
LSTM/GRU output dimension	$[100, 150]$
Dropout	0.4
Number of neurons in dense layers	$[100, 200]$
Focal loss γ	$[1.0, 2.0]$
Regularization factor λ	0.01

Table 4.1: Adjustable hyperparameters in the model. Hyperparameters with values in square brackets indicate the range of values used during training.

validation set is useful as it provides a measure of how well a model is able to generalise during training. The RNN is implemented and trained using the TensorFlow 2.0 package for Python (Abadi et al., 2016)[2] with Keras (Chollet et al., 2015) for implementation of network layers.

4.4.1 Hyperparameters

The model has a number of adjustable hyperparameters: parameters that are not derived through training but are pre-defined before the training process. Table 4.1 lists the adjustable hyperparameters of the model. Two hyperparameters are varied during training for all models: the dimension of the LSTM/GRU output and the number of neurons in the dense layers. We select the values of the LSTM/GRU output dimension to be similar to those used in Muthukrishna et al. (2019). For models trained with the focal loss, there is another hyperparameter that is varied: the focal loss γ parameter. We limit the number of adjustable hyperparameters since we want to limit the total number of models to be trained - there are a total of 32 models with all possible hyperparameter combinations. We find that varying the hyperparameters during hyperparameter optimisation does not hugely impact performance on the validation set, and the hyperparameters listed in Table 4.1 produce good classification results.

[2]https://www.tensorflow.org/

Hyperparameters that remain fixed are the batch size, the learning rate, the dropout fraction, and the regularization factor. Batch size sets the number of samples that is passed through the model at each epoch of training before the weights are updated. Given that the data is imbalanced, the batch size is set to 128 to ensure that examples from the minority class are passed through the model while the weights are being updated. We note that this is a rudimentary method, and a more thorough approach would be to employ 'stratified' batching, where each batch has the same proportion of minority and majority examples. Nevertheless, we find that simply setting the batch size to 128 is sufficient to achieve good results (see section 3.7). The learning rate defines the step size taken during gradient descent to determine the optimal set of weights, in other words it controls how much the weights are changed during training. A learning rate that is too small will fail to find a minimum in weight space, and a learning rate too large will result in unstable training and fail to find a minimum. During training, the learning rate is left to the default TensorFlow value of 1×10^{-4}.

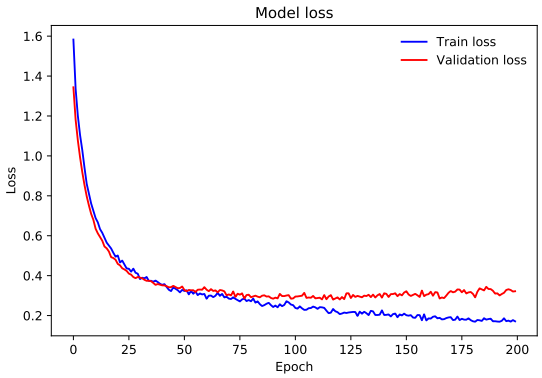
Srivastava et al. (2014) find that setting the probability of retaining a unit in the network between 0.4 and 0.8 optimally reduces test error on a classification task with a deep neural network. In this work, the dropout fraction used in the TensorFlow implementation is defined as the fraction of units that are dropped, so the optimal range found by Srivastava et al. (2014) translates to $0.2 - 0.6$ when expressed as the fraction of units to be dropped. We opt for a dropout fraction of 0.4 in this work. The regularization factor λ is set to the default TensorFlow value 0.01.

The dropout fraction and regularization factor λ could have been used as additional hyperparameters to see if varying their values would impact classification performance, but we choose to keep these values constant as to minimise the total number possible model configurations that need to be trained. The main motivation for the work presented in this chapter is to see how using weighted loss functions affects performance on imbalanced data.

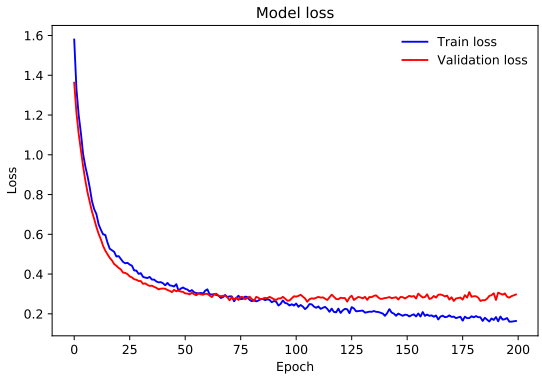
4.4.2 Training process

There are two different RNN architectures used, one with LSTM and the other with GRU, and three different loss functions: an unweighted cross entropy loss (eq. 4.2), a weighted cross entropy loss, and a weighted focal loss (eq. 4.4). In total there are six classes of models, each trained with a range of hyperparameters (Table 4.1). For models trained with the cross entropy loss functions, the focal loss parameter γ is not an adjustable parameter.

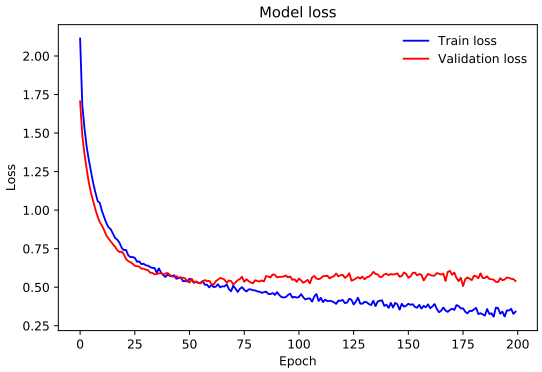
All models are trained for 200 epochs with no early stopping using the Adam optimizer (Kingma & Ba, 2014), and then evaluated on the validation set. The best performing model is selected by choosing the model that has the best AUC score calculated on the validation set. The best models from the six different configurations are then evaluated on the test set. Training was executed on an 8-core CPU. The average time taken to train a single model was approximately one hour at an average of 20 seconds per epoch, and the total time taken to train all models presented in this chapter was $\sim$ 44 hours. The trained models were able to return predictions on the entire test set within $\sim$ 5 seconds. We show how the loss changes with time during training in Figure 4.6



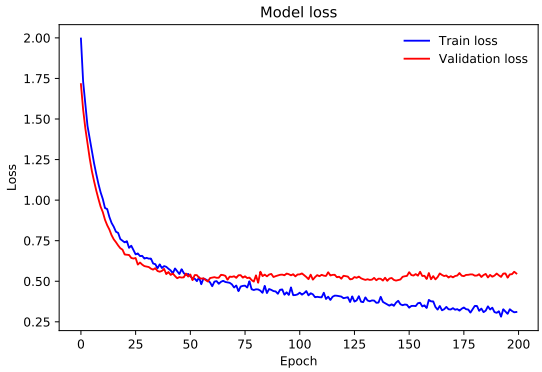
(a) GRU with weighted focal loss.



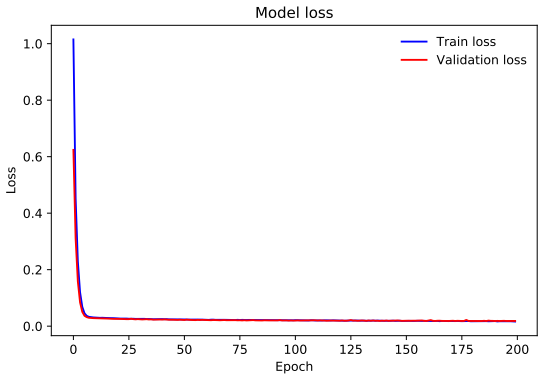
(b) LSTM with weighted focal loss.



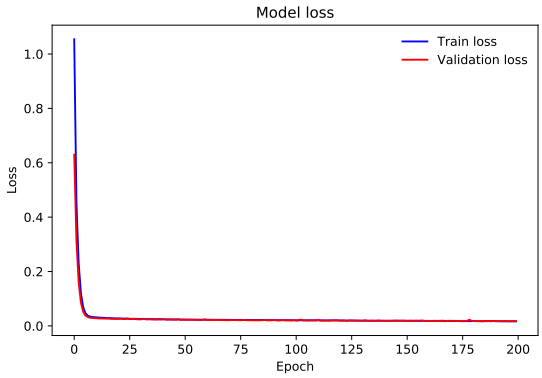
(c) GRU with weighted cross entropy loss.



(d) LSTM with weighted cross entropy loss.



(e) GRU with unweighted cross entropy loss.



(f) LSTM with unweighted cross entropy loss.

Figure 4.6: Evolution of training and validation loss for the best performing models during training. Models with the cross entropy loss converge quickly, but the loss is dominated by contribution from easy to classify examples. Models with the weighted cross entropy loss and focal loss eventually converge within 200 epochs of training, and are also able to account for examples from the minority classes.

4.5 Results

4.5.1 Hyperparameter optimisation

The models are trained with all possible hyperparameter combinations presented in Table 4.1. After training, all models are evaluated on the validation set. The best set of hyperparameters for the six different model configurations are summarised in Table 4.2, with both AUC and the F_1 score shown for each model. All models converge within 200 epochs (Figure 4.6). At each epoch, the model weights are updated through gradient descent such that the loss will be minimized. A model is said to converge once the loss stops decreasing, indicating that the model has reached a minimum in weight space.

The weighted cross entropy models show a small increase of ~ 0.03 in AUC over the unweighted cross entropy models, and the weighted focal loss shows an additional increase of ~ 0.05 in AUC. In this work we used the unweighted F_1 score, which calculates the F_1 score for each class and takes the unweighted mean. Although the unweighted cross entropy loss models both have higher F_1 scores, the models with weighted loss functions perform better overall across all classes - the F_1 score is skewed towards the class with more examples and is not a metric well-suited for imbalanced data. For the weighted focal loss models, the LSTM appears to perform slightly better than the GRU on the validation set.

4.5.2 Test set performance

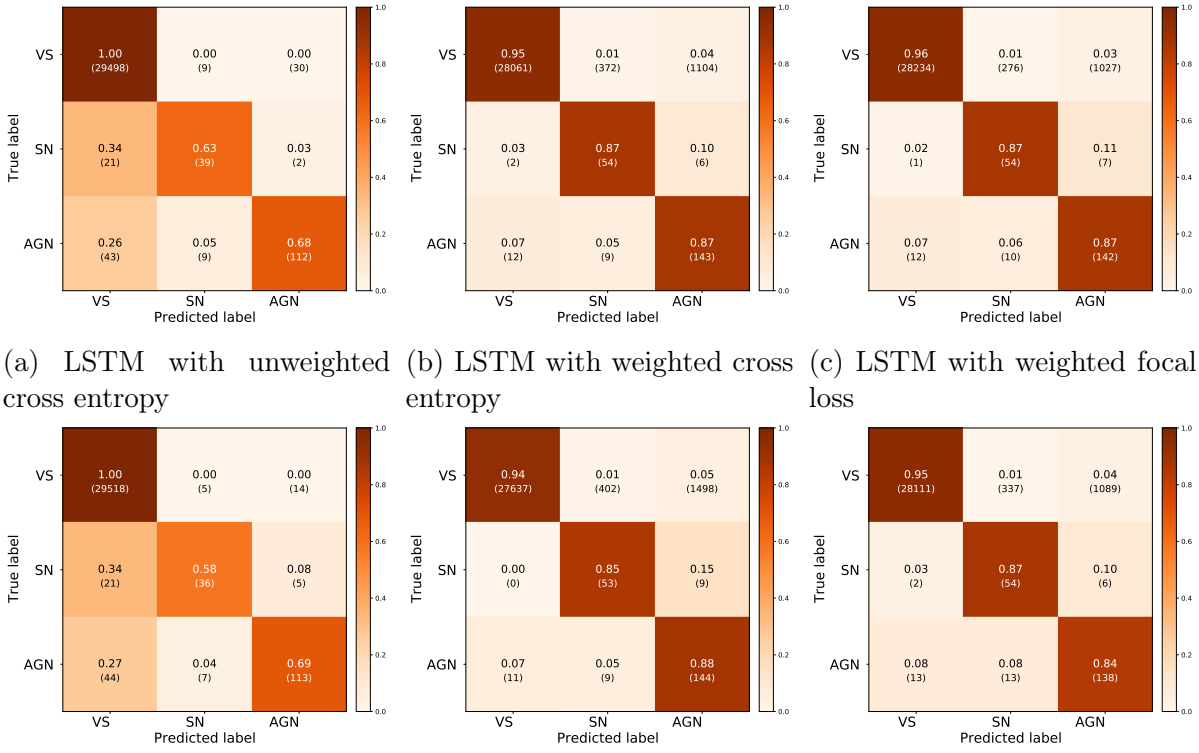
After the best hyperparameters are determined for all the models, the models are then evaluated on the test set. The test set consists of data that the classifier has not seen during the training phase. Columns 8 and 9 of table 4.2 summarises the performance of the six different model configuration on the test set, using all available observations.

As in the results for the validation set, the performance of the models on the test set

RNN type	**Loss**	**Dense neurons**	**RNN output**	γ	Validation		Test	
					AUC	F_1	**AUC**	F_1
LSTM	Weighted focal loss	200	150	1	0.958	0.469	0.966	0.486
GRU	Weighted focal loss	200	150	2	0.947	0.425	0.972	0.468
LSTM	Weighted cross entropy	200	100	-	0.939	0.404	0.967	0.464
GRU	Weighted cross entropy	200	150	-	0.932	0.378	0.968	0.442
LSTM	Unweighted cross entropy	200	150	-	0.899	0.727	0.948	0.794
GRU	Unweighted cross entropy	200	150	-	0.909	0.758	0.937	0.806
GRU NC	Weighted focal loss	100	100	1	0.922	0.322	0.909	0.324

Table 4.2: Results for the best performing models and their best hyperparameters with AUC and F_1 scores, evaluated on the validation and test sets. The dense neurons column denotes the number of neurons in all dense layers preceeding the final output layer, and the RNN output column denotes the dimension of the output of the LSTM and GRU layers. On the bottom row, GRU NC denotes the GRU model with weighted focal loss trained only on time-series data without contextual information.

indicate that simply adding weights to the loss function to account for class imbalance improves performance. Both the weighted cross entropy and weighted focal loss models show an increase of ~ 0.02 in AUC over the unweighted cross entropy. The F_1 scores are also shown for illustrative purposes; although the models with unweighted cross entropy loss have the highest F_1 scores, they have the lowest AUC scores. To better clarify the improvement of weighted loss models over the unweighted cross entropy models, it is prudent to also look at the confusion matrices for each of these models.



(a) LSTM with unweighted cross entropy

(b) LSTM with weighted cross entropy

(c) LSTM with weighted focal loss

(d) GRU with unweighted cross entropy

(e) GRU with weighted cross entropy

(f) GRU with weighted focal loss

Figure 4.7: Confusion matrices for the best performing models on test data. The labels on the x-axis are the labels predicted by the classifier, and the labels on the y-axis are the true labels. Correct predictions are represented by values along the diagonal, incorrect predictions are represented by values in the off-diagonal. The rows of the matrix show the fraction of correct and incorrect predictions for each class, and where incorrect predictions between classes occur. Below the fractions are the number of objects that have been predicted, in parentheses.

Figure 4.7 shows the confusion matrices for all the models evaluated on the test set.

Confusion matrices are useful evaluation tools for multi-class classification problems; they visualise how often a classifier makes correct predictions, and where misclassifications between classes occur. The models with unweighted cross entropy loss perform the worst, and the confusion matrices for these models show the impact of class imbalance on the model.

Using a weighted cross entropy loss function improves performance on AGN and SN, but slightly decreases performance for VS. Figures 4.7b and 4.7e show increased recovery for SN and AGN: up to 87% and 85% accuracy for SN with the LSTM and GRU models, respectively, and up to 87% and 88% accuracy for AGN with the LSTM and GRU models, respectively. The accuracy for VS dropped to 95% for the LSTM model and 94% for the GRU model, with misclassifications occurring in both SN and AGN. There is a small amount of confusion ($< 15\%$) between SN and AGN for the LSTM and GRU models with weighted cross entropy loss functions.

The models using weighted focal loss show similar performance to the weighted cross entropy models, with minor improvements. Figures 4.7c and 4.7f show that both LSTM and GRU models with weighted focal loss are able to achieve 88% accuracy for SN, up to 96% accuracy for VS (with the GRU model achieving 95% accuracy for VS), and up to 87% accuracy for AGN (with the GRU model achieving 84% accuracy for AGN). There is some degree of confusion between SN and AGN, but no more that 11% of examples from these classes are misclassified as the other. Only up to 3% of examples from SN are classified as VS, and up to 8% of AGN are classified as VS. This is a significant improvement over the models with unweighted cross entropy, where up to a third of objects from the minority classes (SN, AGN) are misclassified as the majority class (VS).

Looking at the number of predictions made for each entry in the confusion matrix for the GRU model trained with focal loss (Figure 4.7f), it can be seen that the number of VS predicted as SN and AGN is greater than the number of actual SN and AGN in the

test set. The classifier is designed to produce prediction probabilities, and the predicted class is simply selected by choosing the class prediction that has the highest probability. We can examine how varying the threshold value for the class probability can refine the classification results. By breaking down the three class problem into three binary classification problems, where for each case the positive class is one of VS, SN, or AGN, and the negative class are the other two, we can calculate the true positive rate (TPR) and false positive rate (FPR) for each case. The probability of a negative prediction is simply given as the sum of the probability of the two non-positive classes. We use the predictions given by the GRU model trained with focal loss since it has the highest AUC score on the test set, and compute the TPR and FPR, and produce a ROC curve for each class (Figure 4.8). The ROC curves for all the classes reflects the high AUC score of the GRU model trained with weighted focal loss, covering most of the TPR-FPR space and reaching the top left-hand corner (high TPR at low FPR). See Chapter 2 for a review of machine learning metrics.

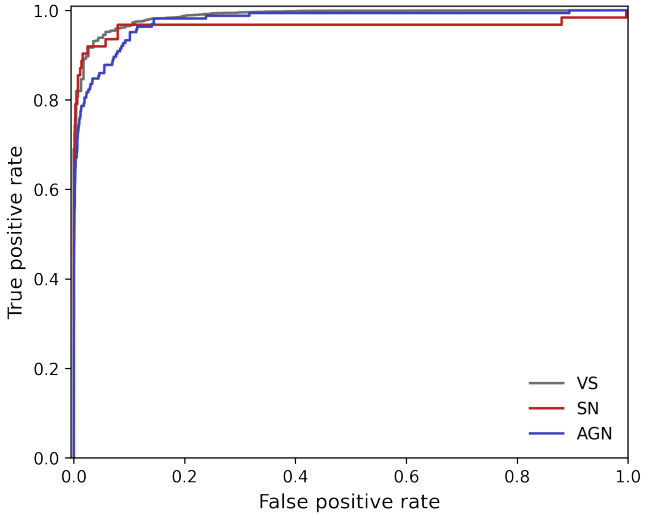


Figure 4.8: Receiver operating characteristic (ROC) curves for the VS, SN, and AGN classes (grey, red, and blue respectively). ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) for a range of threshold values that dictate whether an object is classified as positive or negative. The curve is obtained by considering a separate binary classification case for each class, treating one class as positive, and the rest as negative.

Class	Threshold	TPR	FPR
VS	0.7	86.2% (25,465)	1.8% (4)
SN	0.7	74.2% (46)	0.2% (57)
AGN	0.7	72.6% (119)	0.7% (213)

Table 4.3: The true positive rate (TPR) and false positive rate (FPR) for each class, evaluated on the test set at a threshold for the GRU model trained with focal loss. The positive and negative predictions are obtained by treating each class as positive, and the other two as negative, creating a binary classification problem for each of the three classes. The number of true positive predictions and false positive predictions are shown in parentheses with the TPR and FPR values.

We can examine how varying the threshold can reduce contamination, that is, to reduce the number of false positives in each class. By selecting a 'cut-off' threshold, any objects that have a prediction probability below the threshold can be regarded as negative, and those with a prediction probability above the threshold can be regarded as positive. Table 4.3 shows the TPR and FPR along with the number of true positive and false positive predictions for each class by using a threshold value of 0.7. By selecting a cut-off threshold, the number of false positive predictions for all classes is reduced. At a threshold value of 0.7, the TPR of SN and AGN is $> 70\%$ at a FPR of $< 1\%$. The number of false positives for SN and AGN is still significant, and we note that reducing contamination on minority classes in an imbalanced data setting remains a challenge.

4.5.3 Time-dependent performance

With an RNN architecture, it is possible to take in sequential inputs of different lengths. Hence, it is possible to evaluate the classifiers performance by varying the number of light curve observations used. To do this, the time-series input matrix can be formatted so that it contains only the first n observations, and the remaining values are padded. The models are then evaluated on the test data, using an increasing number of observations from $n = 1$ to a maximum of $n = 30$.

Figure 4.9 shows how the AUC of all models vary as the number of observations

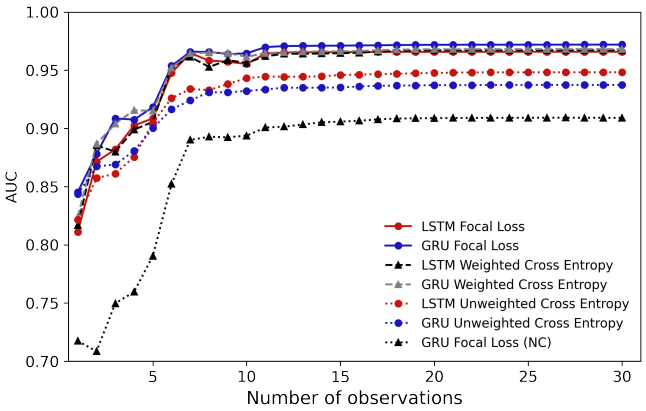


Figure 4.9: AUC scores evaluated for all models on the test data, plotted as a function of increasing number of light curve observations included in the light curve. The black dotted line with triangular markers shows the AUC scores for the GRU model trained with weighted focal loss without contextual information (labelled GRU Focal Loss (NC)).

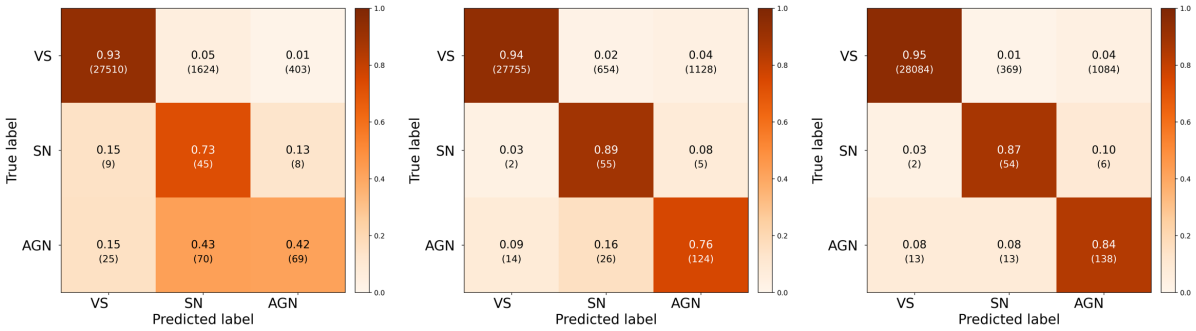
included in the light curves are increased. In this case, the number of observations refers to the maximum number of observations that are included. If the maximum number of observations is m, then a light curve with fewer than m observations will have all its observations included. If a light curve has more than m observations, then only the first m observations are used.

With just one light curve observation, the unweighted cross entropy models, weighted cross entropy models, and the LSTM model with weighted focal loss achieve AUC scores of ~ 0.82. The GRU model with weighted focal loss achieves the highest AUC score with a single light curve observation with 0.84. As more observations are included, all models show an increase in AUC until around ten observations, after which the AUC scores maintain a constant value. Since a majority of the light curves in the dataset only have up to ten observations (see Figure 4.3), it is not surprising that model performance remains constant after a maximum of ten light curve observations are included. The final values of the AUC for all models are shown in column 8 of table 4.2.

We perform some additional analysis on how the GRU model with weighted focal loss performs over time, choosing the aforementioned model since it has the highest AUC

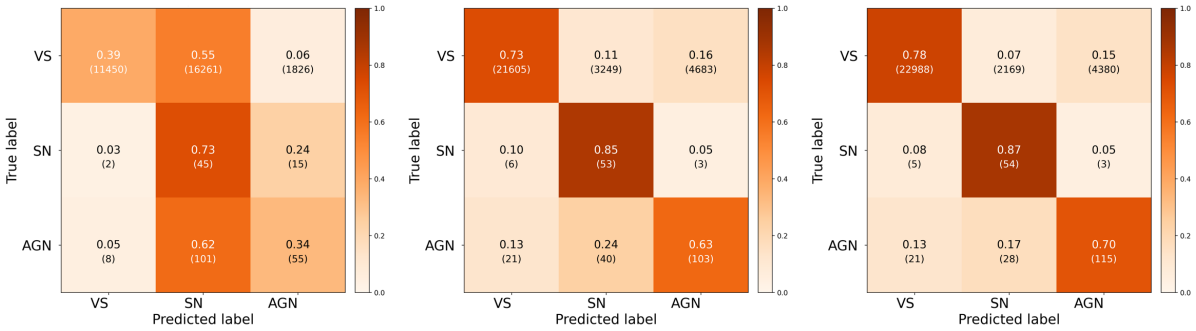
score on the test set. Figure 4.10 shows the confusion matrices for the GRU model with weighted focal loss evaluated with different numbers of light curve observations. With one observation, the model is already able to separate out VS from other objects to a good degree of accuracy, achieving 93% accuracy for VS. At one epoch of observation, the model achieves 73% accuracy for SN, and 42% accuracy for AGN, with 43% of AGN being misclassified as SN. Other than AGN being misclassified as supernovae, there is some degree of confusion between all classes: 13% of SN are misclassified as AGN, and 15% of SN and AGN are misclassified as variable stars.

With up to six light curve observations, the accuracy for SN and AGN improves. 89% of supernovae and 76% of AGN are correctly classified, with some misclassifications between the two ($< 16\%$), and few SN and AGN being misclassified as variable stars. At a maximum of twenty observations, the model reaches the maximum performance. The confusion matrix for the GRU model with weighted focal loss at twenty epochs in figure 4.10c appears similar to the confusion matrix evaluated with all epochs of observations as in figure 4.7f. The SN accuracy drops from 89% at six observations to 87% at twenty observations, suggesting some confusion as more observations are included and light curves appearing similar to each other at longer timescales.



(a) Single epoch observation (b) Maximum 6 observations (c) Maximum 20 observations

Figure 4.10: Confusion matrices for the GRU model with weighted focal loss, evaluated with an increasing number light curve observations.



(a) Single epoch observation (b) Maximum 6 observations (c) Maximum 20 observations

Figure 4.11: Confusion matrices for the GRU model with weighted focal loss trained only on time-series data, evaluated with an increasing number light curve observations.

4.5.4 Importance of contextual information with t-SNE

Imaging surveys such as GOTO will be able to provide contextual information for newly discovered objects (for example, cross matching to galaxy catalogs) in addition to photometric data. We now discuss the importance of contextual information for the model in learning to differentiate between objects from different classes. To investigate the impact of contextual information, we train a grid of GRU models with weighted focal loss using the hyperparameters in Table 4.1 on the same data as the other models, but without additional contextual information. The inputs to these models are just the time-series data from the light curves $\mathbf{X_T}$. We identify the best performing model by selecting the model that achieves the highest AUC score on validation set.

Performance without contextual information

The hyperparameters, AUC and F_1 scores of the best performing GRU model with weighted focal loss trained only on time-series data are shown in the bottom row of Table 4.2. The model achieves an AUC score of 0.922 on the validation set, which is higher than the validation AUC scores of the unweighted cross entropy models.

We follow the same process as in section 4.5.2 and evaluate this model on the test set. The model achieves an AUC score of 0.902 on the test set, which is the lowest AUC

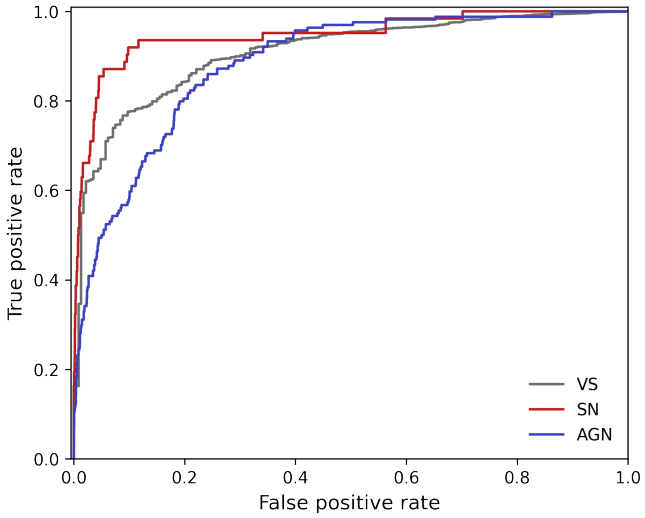


Figure 4.12: ROC curves for the VS, SN, and AGN classes (grey, red, and blue respectively) for the GRU model with weighted focal loss, trained only on time-series data.

score out of all models. In figure 4.9, we plot how the AUC score evolves as more light curve observations are included. At one observation, the model achieves an AUC score of 0.717 and then increases up until the six observations after which is starts to maintain a constant value.

Figure 4.11 shows the confusion matrices evaluated on the test set using an increasing number of maximum observations. With just one light curve observation, the model performs worse than the GRU model with weighted focal loss trained with contextual information. The model achieves an accuracy of 39% for VS and incorrectly classifying 55% of VS as SN, and 34% accuracy for AGN and incorrectly classifying 62% of AGN as SN. SN accuracy for the model is similar to the models trained with weighted focal loss, with an accuracy of 73%. When using all available light curve observations, the model trained only on time series data achieves a similar accuracy for SN as the weighted focal loss models at 87%, but lower accuracy for VS at 78% and AGN at 70%. There is a slightly higher degree of misclassification between classes compared to the models with weighted loss functions trained with contextual information.

Figure 4.12 shows the ROC curve for the model trained only on time-series data

(using all available light curve observations), by separating the three-class problem into three separate binary classification problems. Compared to Figure 4.8, the ROC curves show that the model does not perform as well when contextual information is excluded.

Overall, the model trained only with time-series data performs worse than its counterpart trained with contextual information, but is able to achieve comparable accuracy for SN. This suggests that the model is able to extract information from the light curves that allows for good separation of SN from the other classes. We expand on this in the following analysis.

t-Distributed Stochastic Neighbor Embedding representation

We use t-Distributed Stochastic Neighbor Embedding (t-SNE) to represent how the model transforms the input data at different layers of the network. t-SNE is a data visualization technique used to map a high-dimensional dataset into a low-dimensional dataset that can be visualised in a two or three dimensional scatter plot (van der Maaten & Hinton, 2008).

t-SNE is a dimensionality reduction technique, that takes a high-dimensional dataset $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ and converts it into a low-dimensional representation $\mathcal{Y} = \{y_1, y_2, ..., y_n\}$. The low-dimensional data points $\mathcal{Y}$ are mappings of the high-dimensional data points $\mathcal{X}$ in the low-dimensional space.

The high-dimensional Euclidean distances between points are converted into probabilities that represent similarities between points by centering a Gaussian distribution onto each point. The similarity between points x_i and x_j are encapsulated in the joint probability $\mathcal{P}_{ij}$; if x_i and x_j are near, then $\mathcal{P}_{ij}$ will be high and if they are far apart then $\mathcal{P}_{ij}$ will be low.

In the low-dimensional mapping, the probability $\mathcal{Q}_{ij}$ is a measure of similarity between y_i and y_j, and $\mathcal{Q}_{ij}$ will be high if the two points are near each other and low if they are far apart. Instead of centering a Gaussian distribution onto the points in

low-dimensional space, a student t-distribution is used instead. Using a heavier-tailed distribution allows moderate distances in the high-dimensional map to be modelled by larger distances in the lower-dimensional map, preventing a 'crowding' of points that are not too dissimilar (van der Maaten & Hinton, 2008).

The variance of the Gaussian distribution is set so that the probability distribution produced by the variance has a fixed perplexity, which is set by the user. The perplexity can be thought of as a measure of the effective number of neighbors in the region of the data point in question. Here, the perplexity parameter is set to 20.

Given the two joint probabilities $\mathcal{P}_{ij}$ and $\mathcal{Q}_{ij}$, t-SNE determines an optimal low-dimensional mapping $\mathcal{Y}$ of the high-dimensional dataset $\mathcal{X}$ by minimising the Kullback-Leibler divergence (Kullback & Leibler, 1951) of $\mathcal{Q}_{ij}$ and $\mathcal{P}_{ij}$ using a gradient descent method.

t-SNE has been used to visualise class separability in supernovae classification with machine learning classifiers (Lochner et al., 2016). In this analysis, we use t-SNE to visualise how class separability changes at different layers in the network. We consider two models: the GRU model with weighted focal loss trained with contextual information and the same model trained with only time-series data.

We take the output of the model at two points: the output after the final GRU layer, where the model only considers time-series information from the light curve, and the output after the final dense layer before the output layer. In the model trained with contextual information, the output of the final dense layer will encode the contextual information introduced after the final GRU layer. The model is fed the training data as input, and t-SNE is used to produce a low-dimension visualization of the high-dimensional intermediate outputs. Figure 4.13 shows the two-dimensional mapping of the model outputs at the final GRU layer and the final dense layer, for both models. The `scikit-learn` Python package (Pedregosa et al., 2011) implementation of t-SNE is used in this analysis.

Looking at the t-SNE representation of the outputs for the model trained without contextual information, there appears to be no clear clustering in the representation for the output after the GRU layer in Figure 4.13a. The majority of AGN occupy the right side of the plot, the majority of SN are sparsely clustered in the bottom-right region, and few AGN and SN occupy the left side. In Figure 4.13b, the t-SNE representation of the output after the final dense layer shows some coherent clustering of SN and AGN. AGN cluster around the top-right and bottom-left region, with some spread out in the middle. There is a compact structuring of SN on the left side, and a cluster of SN in the bottom-left near the AGN. There are a few AGN and SN in the top-left region of the plot.

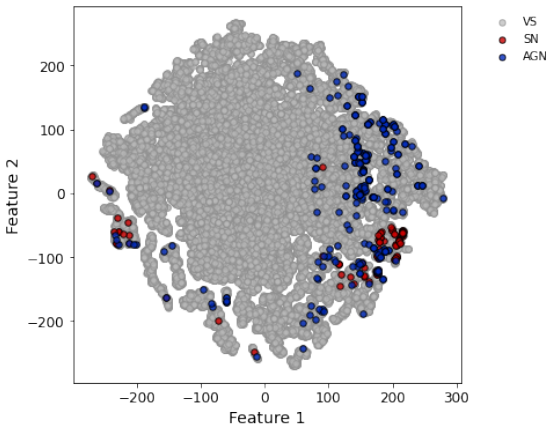
From Figure 4.13c for the output after the final GRU layer for the model trained with contextual information, there appears to be no clear clusters of objects from the same class. AGN are spread out across the plot, and there is some clustering of SN in the top-right region. Looking at Figure 4.13d after including contextual information, the clustering of objects from the same class becomes more apparent. There appears to be a compact and distinct cluster of AGN near the bottom of the plot, and a tight clustering of SN along with AGN in the bottom-right region. There is some overlap between SN and AGN, indicating where some of the misclassifications are occurring.

We note that the VS class is a very broad class, containing a multitude of different variable objects with distinct subclassifications. In Figure 4.13d, there seems to be a few coherent structures which may indicate where variable objects from the same or similar classes are located.

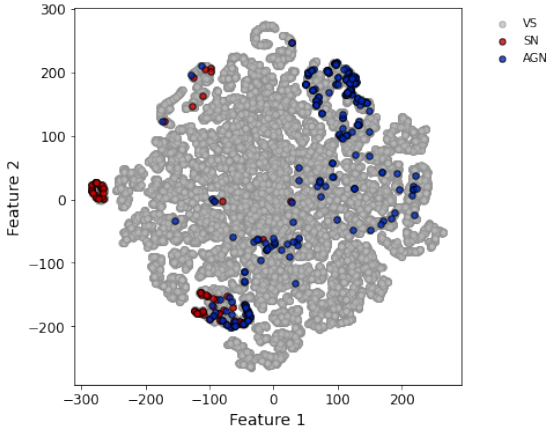
From Figure 4.13, it is clear that the incorporation of contextual information into the classifier provides useful information that allows the model to learn better class separability. Using t-SNE in this way is a method to provide an approximate measure of 'feature importance' in deep learning classifiers, which can be challenging compared to deriving feature importance in feature-based machine learning classifiers such as random

forests which utilize hand-made features.

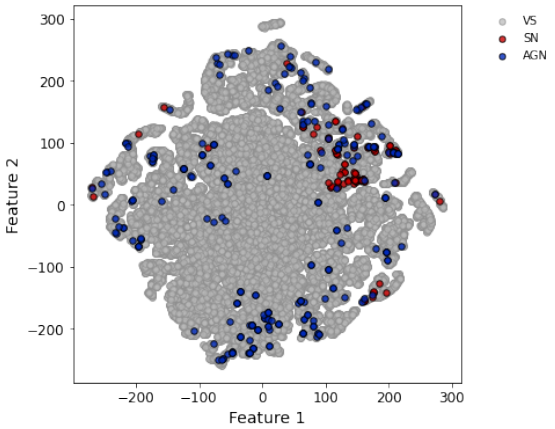
It should be noted that t-SNE is primarily a *data visualization* tool, and here it is used to visualise class separability learnt by the model at different stages. It is possible to explore the hyperparameter space when generating t-SNE plots, and use a number of diagnostics to assess the 'quality' of the dimensionality reduction (such as comparing distances between points in high and low-dimensional space) and identify any correlating features within the high-dimensional dataset (Chatzimparmpas et al., 2020). However, further interpretation of the t-SNE plots is beyond the scope of the work presented in this chapter.



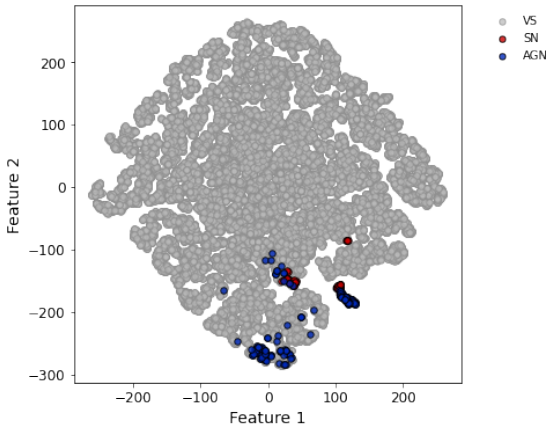
(a) t-SNE representation of model output after the final GRU layer, with no contextual information.



(b) t-SNE representation of model output after the dense layer before the output layer, with no contextual information.



(c) t-SNE representation of model output after the final GRU layer, with contextual information.



(d) t-SNE representation of model output after the dense layer before the output layer, with contextual information.

Figure 4.13: t-SNE representation for the network outputs at different stages. Each datapoint corresponds to a single object in the training set; grey points are VS, red points are SN, and blue points are AGN.

4.6 Discussion

Here, we provide a discussion on the task of object classification in the context of the GOTO survey, how we handle class imbalance in this work, and how the use of contextual information in addition to time-series data from light curves can help classifiers produce more robust classifications.

In this work, the dataset of 99,201 labelled objects are split into three broad classes: variable stars, supernovae, and AGN. There are over 98,000 variable star light curves in the dataset, and more than 350 unique class labels provided in the AAVSO catalog (Watson et al., 2006). A number of these labels include a mixture of classes, where there is no certainty to which class an object belongs to. Grouping all variable type objects into a single super-class makes the classification task simpler for the model, and allows for reliable early-time classifications. We also assume that the classifications provided in these catalogs are the ground-truth. To create an extended and more robust labelled dataset, additional catalogs can be used to provide more labelled examples, and also verify the labels provided for already known objects.

The dataset used to train the classifier was obtained from observations conducted in the GOTO all-sky survey mode. Given that the light curves in the dataset can be quite sparse, and are only in a single filter, this may have presented a challenge for classification into multiple object subtypes. A more refined classifier can be trained to classify between subtypes of objects. An approach for a more refined classification could be to take a hierarchical classification scheme as in Hosenie et al. (2020), where once a subset of objects have been classified into broad types, another classifier is used to further classify the subset into more specific classes.

Targeted sub-sky surveys that focus on observing a smaller area of sky and local galaxy clusters in search of transients present a chance to conduct observations at higher cadences and in multiple filters. Additional colour information and more uniform light

curve sampling can provide data that can be used to train models to produce better and more refined classifications into object subtypes. Models trained on light curves in multiple filters have been shown to be able to differentiate between different types of transients and supernovae subtypes (Villar et al., 2019; Muthukrishna et al., 2019).

One of the objectives of this work was to highlight the usefulness of RNNs for real-time object classification in surveys. This classifier can be implemented in the GOTO discovery pipeline, where new objects observed by GOTO are classified and the classification is returned to the GOTO Marshall (Steeghs et al., 2021), where it can be displayed in a web interface for users to see.

The classifier returns class probabilities that give a measure of how confident the classifier thinks an object belongs to certain class. This information can be used to decide to trigger additional follow-up observations. With the RNN architecture, it is possible to update the classification probability as new observations of an object are made.

4.6.1 Handling class imbalance in deep neural network architectures

The dataset used to train the RNN classifier for GOTO light curves presented a class imbalance problem. Other works dealing with class imbalance have utilised data augmentation methods for classification with feature based machine learning algorithms (Hosenie et al., 2020; Villar et al., 2019; Boone, 2019; Revsbech et al., 2018) and achieved good performance. Here, we provide an alternative to data augmentation with a RNN classifier, and use an algorithm-level approach to dealing with imbalance, where a weighted cross entropy loss function and a weighted focal loss function are used to account for imbalanced class distribution in the training set. We train two types of RNN architectures, the LSTM and GRU, with three different loss functions: an unweighted cross entropy, a

weighted cross entropy, and a weighted focal loss.

Weighting the cross entropy loss function shows an improvement over the unweighted cross entropy, going from an AUC of 0.948 to 0.968 for the best unweighted cross entropy and weighted cross entropy loss models when evaluated on the test set, respectively. The confusion matrices show that the degree of confusion between the majority and minority classes is reduced by simply weighting the cross-entropy loss function.

The focal loss models perform similar to the weighted cross entropy loss models. The AUC for the best focal loss model was 0.972, evaluated on the test set. For the focal loss models, the GRU model achieves a higher AUC score. We have shown that by using an appropriate loss function to account for imbalanced data, it is possible to achieve good real-time classification of transient and variable sources, without having to artificially augment the training data.

Krawczyk (2016) note that even if a dataset is heavily imbalanced, if the classes are well represented and come from non-overlapping class distributions, it is still possible to achieve good classifications. Future surveys may benefit from obtaining spectroscopy of a wider diversity of targets, and not just based on good signal-to-noise ratios. Having good spectroscopic coverage of sources over a range of magnitudes can help data augmentation efforts to create representative training sets for supernovae classification (Carrick et al., 2020).

4.6.2 Contextual information

We trained a GRU model with weighted focal loss to classify light curves using only time-series data without additional contextual information, to investigate the impact of contextual information on classification. The model trained without contextual information achieved an AUC score of 0.902 on the test set, which is the lowest AUC score on the test set out of all models. Without contextual information, the model has an

overall worse performance compared to the models with weighted focal loss functions that incorporate contextual information.

Using t-SNE to visualise the intermediate outputs of the GRU models trained with and without contextual information, it can be seen that using contextual information such as location in the sky and distance to the nearest galaxy allows the model to learn better class separability compared to just using information from the light curve. In principle, the model architecture used in this work where contextual information is ingested into the model separately to the time-series information from the light curve can be applied to any survey. We show that this is possible in Chapter 6.

Only three values were used as the contextual information input into all the models: the galactic latitude and longitude, and the distance to the nearest galaxy from the object. It is possible to include additional information, such as the physical properties of nearby galaxies (colour and metallicity), redshift, and galactic extinction.

In the era of large sky surveys such as ZTF and LSST, and the availability of multiple alert brokers (Möller et al., 2020; Förster et al., 2020; Smith et al., 2019; Narayan et al., 2018) that ingest the data produced by these surveys and distribute it to the astronomy community, additional information on newly discovered objects other than photometric data should be leveraged to provide accurate real-time classification of objects. Being able to classify explosive transients early in their light curve evolution allows for follow-up in the early stages of evolution that can provide constraints on explosion mechanisms and progenitor environments (Zhang et al., 2018; Khazov et al., 2016).

4.7 Conclusion

In this chapter we presented a recurrent neural network classifier to classify objects observed with the GOTO survey using their light curves, and additional contextual information such as on-sky position and distance to the nearest galaxy obtained by cross-

matching with a catalog. We create a labelled dataset from the GOTO data, and split the dataset into three classes: variable stars (VS), supernovae (SN), and active galactic nuclei (AGN). The dataset is imbalanced, with 99% of labelled objects belonging to the variable star class. We adopt weighted cross entropy and focal loss functions to account for the imbalance, and reduce the model bias towards the majority class. The weighted loss functions improves overall classification performance over the standard approach of an unweighted cross entropy loss function with deep neural network classifiers. We also train a model without contextual information and only time-series data, and find that it performs worse than the model with the same configuration trained with contextual information, but is still able to provide meaningful classification. Looking at the low-dimensional representations of model outputs shows that contextual information allows the model to make better distinctions between objects.

Chapter 5

Classification of supernova light curves from multiple surveys and transfer learning for future surveys

5.1 Introduction

With larger telescopes and an increased coverage of the night sky (both in area and time), astronomers are discovering more objects more quickly. The rate of discovery and data collection has prompted work on machine learning and deep learning approaches to automate the identification and classification of new objects. In the past decade or so, a lot of work has been done on supernova light curve classification, to classify supernovae using only photometric observations. At the time of writing, most of these studies focus on classifying supernovae from a single survey with either real or simulated data - the work presented in Chapters 3 and 4 use data from only the GOTO survey.

Lochner et al. (2016) and Charnock & Moss (2017) used simulated supernovae light curves from the *Supernova Photometric Classification Challenge* (SPCC) (Kessler et al., 2010) to build models to classify the light curves into three classes (Ia, Ib/c, II). Muthukrishna et al. (2019) used a recurrent neural network to classify simulated ZTF light curves of various explosive transients, including supernovae. Pasquet et al. (2019) used a convolutional neural network to classify supernovae using simulated data and also real SDSS data. Möller & de Boissière (2020) developed a bayesian neural network approach to classify a set simulated supernova light curves similar to the SPCC dataset. Dauphin et al. (2020), Hosseinzadeh et al. (2020), Villar et al. (2019) created classifiers trained on Pan-STARRS1 supernova light curves. Takahashi et al. (2020) used a neural network to classify supernova light curves from the Hyper Suprime-Cam transient survey. In all the examples listed above, the overall classification performance is good for a number of classification tasks (binary Ia/non-Ia or a multi-class problem into the different supernova subtypes), achieving accuracies of $\gtrapprox 85\%$.

As mentioned, the above studies all focus on classifying supernova light curves that have been obtained from a single survey or simulated to resemble the light curves of a particular survey. Pruzhinskaya et al. (2019) used supernova light curves from multiple

surveys, obtained through the Open Supernova Catalog (Guillochon et al., 2017) to develop an anomaly detection algorithm, capable of identifying rare supernovae classes and non-supernovae objects within the dataset. In this chapter, we present an approach to use data from the Open Supernova Catalog to develop a classifier that is capable of classifying supernova light curves from different surveys. The challenge in working with light curves from different surveys is dealing with differences in how the photometry is calibrated, and the different filters used when making observations. By having a classifier that is agnostic to differences across different surveys, it allows the use of more available real data so that model training is not limited to the size of a sample obtained from just a single survey. We expand on the literature by applying a deep learning approach to classification on a heterogeneous dataset of supernova light curves, combined from multiple surveys which also allows access to a wider wavelength coverage in broadband photometric observations.

One approach to working with a heterogeneous dataset is to find a way to standardise the data, so that they are represented in a more uniform manner. Boone (2019) used a Gaussian process to model simulated LSST light curves from the PLAsTiCC dataset, by interpolating in both time and wavelength. Gaussian processes have also been used to generate a two-dimensional representation of supernovae light curves (Qu et al., 2021; Qu & Sako, 2021), which are then used as inputs to a convolutional neural network to produce classifications. By using a Gaussian process to interpolate light curves in both time and wavelength, it is possible to create a uniform representation of light curves that consist of observations made in different filters. We use this approach to classify supernova light curves from the Open Supernova Catalog (Guillochon et al., 2017).

A challenge in creating classifiers for new surveys is the lack of a labelled training set with which to train a model with. Many machine learning and deep learning classification methods assume that the training and test data come from the same distribution and share a common feature space. When the distribution changes, the models need to

be retrained again with a new labelled training set. In most cases, creating a new labelled training set to account for the change in distribution can be extremely difficult. Transfer learning (Pan & Yang, 2010) is an approach that uses the knowledge gained in performing a task (e.g. classification) in one domain (e.g. data from one particular survey) to perform another task in a different domain (e.g. data from a different survey). We investigate the use of transfer learning to classify light curves from the PLAsTiCC dataset (The PLAsTiCC team et al., 2018), transferring domain knowledge from the task of classifying Open Supernova Catalog light curves.

In section 5.2 we introduce data from the Open Supernova Catalog, and in section 5.3 we present a two-dimensional Gaussian process to generate a two-dimensional representation of supernova light curves. Section 5.4 introduces the convolutional neural network used for classification. We present the results of classifying Open Supernova Catalog data in section 5.5. We introduce transfer learning in section 5.6, the PLAsTiCC data in section 5.7, and results on classifying PLAsTiCC data in section 5.8. We provide a discussion of the work presented in this chapter in section 5.9 and conclude with section 5.10.

5.2 Open Supernova Catalog data

Supernovae light curves and metadata (such as the supernova classification obtained via spectroscopy or through expert human inspection, any available spectroscopic data, and the R.A. and Dec) were downloaded from the Open Supernova Catalog website [1]. The downloaded data consisted of supernovae discovered from before 1989 and up to the end of 2019, totalling 80,914 objects. All objects that were labelled as 'Candidate' or other non supernovae classes were discarded. Only objects that had been labelled as type Ia, Ibc, or II (including all sub-classifications within those types) were kept.

[1] https://sne.space/download/

5.2.1 Standardising magnitudes and filters

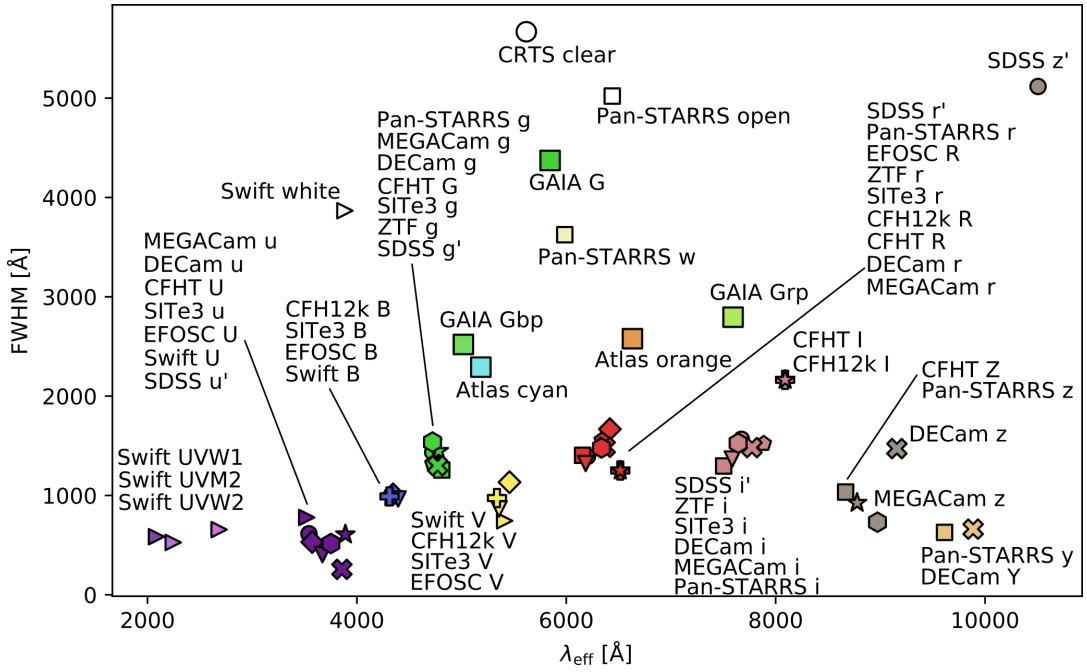


Figure 5.1: A plot of the effective wavelength λ_{eff} against the full width half-maximum (FWHM) of the filters available in the Open Supernova Catalog dataset. Both λ_{eff} and FWHM are given in Angstroms.

The light curves in the Open Supernova Catalog dataset consists of observations that have been made with a variety of instruments across a number of different surveys. Table A.1 in the appendix lists the filters used in the Open Supernova Catalog dataset, along with the associated effective wavelengths and full width half-maxima. The data tabulated in table A.1 was obtained from the Spanish Virtual Observatory (SVO) Filter Profile Service (Rodrigo & Solano, 2020)[2]. The effective wavelength against full width half-maximum for the filters listed in Table A.1 is plotted in Figure 5.1.

The dataset also contains photometry that have been generated using models or simulations, we discard these and only keep real photometric observations. Where available we use the `system` column in the photometry data to identify which magnitude system the data is calibrated to, otherwise the `instrument` column is used to identify which

[2]http://svo2.cab.inta-csic.es/theory/fps/

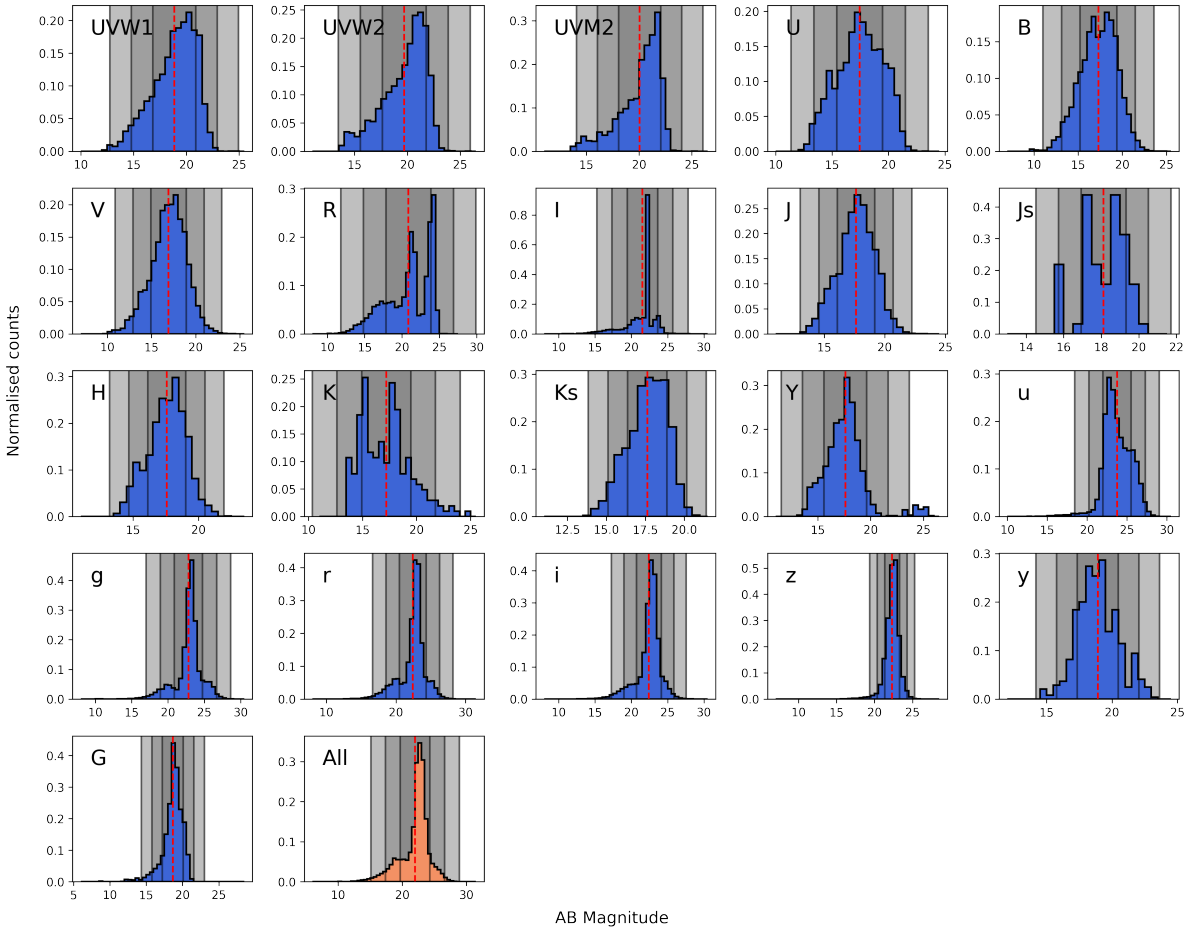


Figure 5.2: Histograms of magnitudes in the AB system in the filters included in the dataset. The red dashed line indicates the mean, and the shaded grey regions indicated the 1σ, 2σ, and 3σ values - these are included for illustration purposes only. The filters used include the Swift (UVW1, UVW2, UVM2) filters, the Johnson-Cousins filters, SDSS *ugrizy* filters, and the Gaia G filter. The histogram in orange shows the distribution over the whole dataset.

instrument was used to make the observation. The next step is then to convert all the magnitudes so that they are in the same magnitude system. The majority of magnitudes in the dataset are given in the AB magnitude system, so we convert the rest of the magnitudes into AB magnitudes. Magnitudes given in systems other than AB were converted using Tables A.2, A.3, and A.4 listed in the appendix. Figure 5.2 shows the distribution of AB magnitudes in each filter used in the dataset. After converting all magnitudes into the AB system, the magnitudes are then converted into flux (in units

of erg s^{-1} Hz^{-1} cm^{-2}) using

$$m_{AB} = -2.5 \log_{10} f_\nu - 48.60 \tag{5.1}$$

where f_ν is the monochromatic flux. We introduced the AB magnitude system in Chapter 1.

5.2.2 Light curve trimming

Some light curves in the dataset span periods of up to multiple years due to seasonal gaps, and where the only photometry available is the host galaxy without the supernova. To shorten these longer light curves so that it covers the rise, peak, and decline of the supernova, the steps listed below are taken. Figure 5.3 shows some example trimmed light curves. We find that this method is good at isolating the rise, peak, and decline of supernovae in the Open Supernova Catalog dataset.

- Split long light curves (longer than 300 days of observations) into shorter light curve chunks if there is a gap in observations longer than 60 days
- Compare the standard deviation in magnitudes of each light curve chunk σ_{chunk} to the standard deviation of the whole light curve σ_{lc}
- If $\sigma_{\text{chunk}} < \sigma_{\text{lc}}$, then that portion of the light curve is discarded.

5.2.3 Selection cuts

To create the dataset, the following selection cuts were made:

- Total number of observations in a the light curve is ≥ 6
- At least two or more filters used

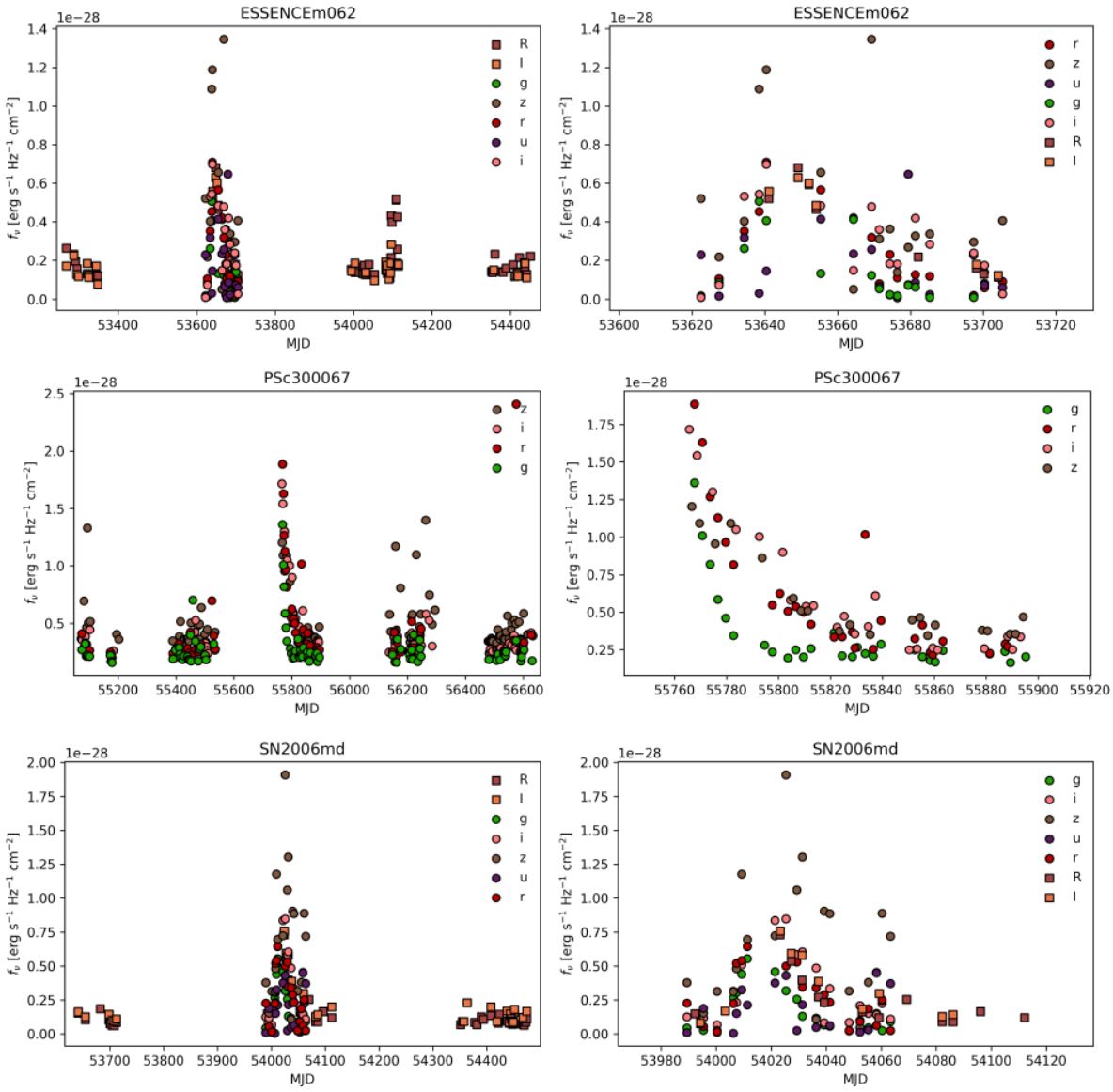


Figure 5.3: Light curves from the Open Supernova Catalog, before (left) and after trimming (right).

- The average number of observations per filter is ≥ 2
- The length of observations spans at least 20 or more days

The cuts were made to ensure that the light curves had good coverage across multiple wavelengths and in time, so that there was enough information in each light curve to allow a model to learn to differentiate between the different classes.

After the cuts, the number of remaining objects is 6330 of the total. The dataset is split into 60% for training (3796 objects), 15% for validation (951), and 25% for testing (1583). The data is split into three classes for the classification task: Ia, Ibc, and II. The proportion of each class in the training, validation, and test set is the same. This is done so that there are a good number of samples from each class in the validation and test set so we can evaluate classification performance on the class with fewer examples (Ibc), and enough examples to in the training set to learn from. Table 5.1 summarises how the dataset is partitioned. Type Ia and II supernovae make up the majority of objects in the dataset, with only a small amount of type Ibc supernovae in the data, presenting an imbalanced dataset. Figure 5.4 shows some summary statistics of all the light curves in the final dataset.

Type	**Training**	**Validation**	**Test**	**All data**
Ia	2145 (56.5%)	542 (57.0%)	883 (55.8%)	3570 (56.4%)
II	1563 (41.2%)	385 (40.5%)	657 (41.5%)	2605 (41.2%)
Ibc	88 (2.3%)	24 (2.5%)	43 (2.7%)	155 (2.4%)
Total:	3796	951	1583	6330

Table 5.1: A breakdown of how the Open Supernova Catalog dataset is divided for training, validation, and testing, along with the class distribution of the three supernova classes.

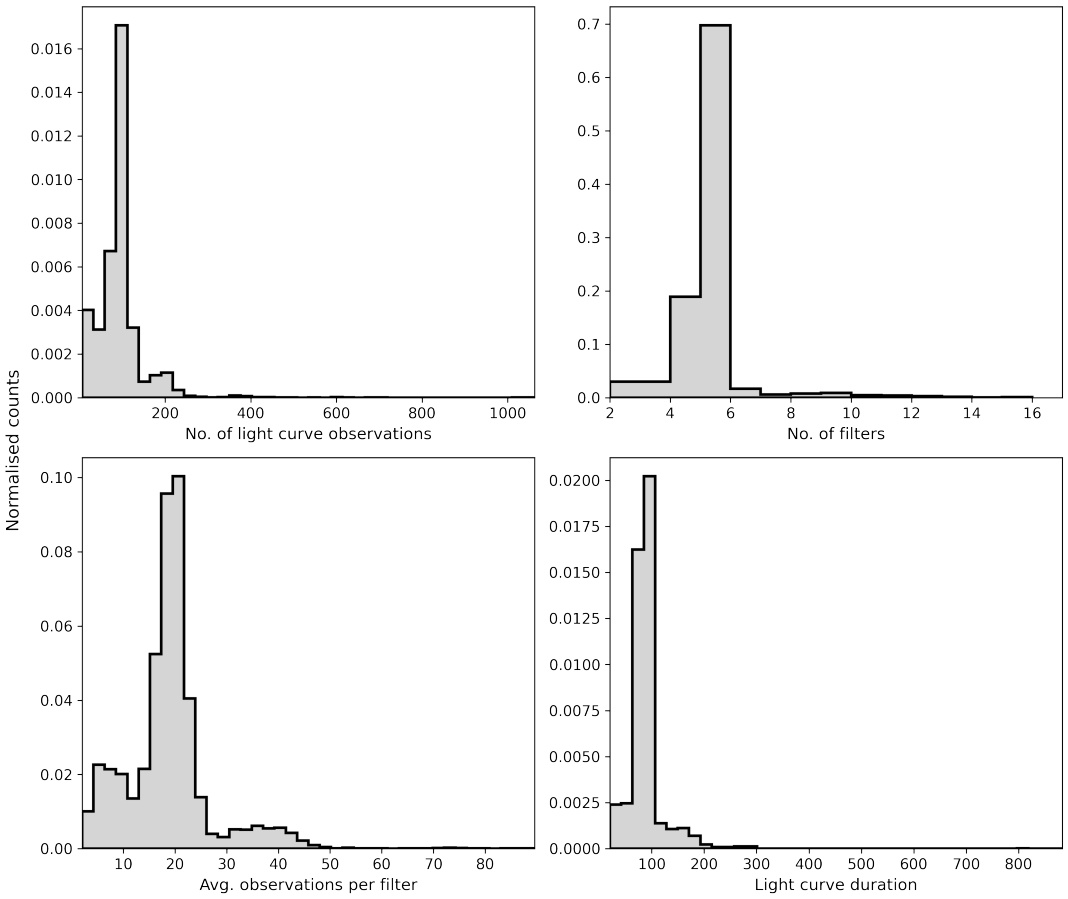


Figure 5.4: Histograms showing the summary statistics of all the light curves in the final dataset. *Top left*: the total number of observations in the light curve in all filters. *Top right*: the number of filters in which observations were made. *Bottom left*: the average observations per filter, obtained by counting the number of observations in each filter and then dividing by the total number of filters used. *Bottom left*: the duration of the light curve in days.

5.3 Gaussian processes for interpolation in time and wavelength

5.3.1 Gaussian Processes

A Gaussian process is a generalisation of the Gaussian probability distribution (which describe random variables) and can be thought of as a distribution over functions. Gaussian process regression attempts to find a function $f(x)$ given a number of observed points

$y(x)$ that determines the value $y(x')$ for unobserved independent variables x' (over a finite interval of x' values) by drawing from a distribution of functions. The distribution of functions is determined by selecting a covariance function, which specifies the covariance between pairs of random variables. Covariance functions have adjustable hyperparameters, which determine the form of the Gaussian process prediction for $f(x)$. For a detailed discussion on Gaussian processes see Rasmussen & Williams (2005).

5.3.2 Two-dimensional Gaussian process regression

In order to create a uniform representation of light curves in different filters, we follow the approach used in Qu et al. (2021) and Boone (2019), and use a two-dimensional Gaussian process regression to interpolate the light curves in wavelength and time. We model the light curves to create a two-dimensional image, referred to as a 'flux heatmap' in Qu et al. (2021) and Qu & Sako (2021), where flux is a function of time t and wavelength λ.

We label each flux measurement in all light curves in the dataset with the effective wavelength λ_{eff} of the filter with which it was observed with. The values for λ_{eff} for each filter are listed in table 5.2, and are obtained from the SVO Filter Profile Service (Rodrigo & Solano, 2020)[3]. Observations covering wavelengths from the Swift UVW2 filter (with $\lambda_{\text{eff}} = 2085.73$Å) up to the SDSS y filters (with $\lambda_{\text{eff}} = 12355.0$Å) were used. These filters were used as the vast majority of observations in the dataset were made using filters within this wavelength range. All flux values of each light curve are associated with a time measurement t (time of observations) and a wavelength value λ_{eff}, the effective wavelength of the filter used to make the observation. We scaled the time so that the time of the first observations is $t = 0$.

As in Qu et al. (2021) and Boone (2019), we use the Matérn 3/2 kernel in our two-dimensional Gaussian process, with a fixed characteristic length scale in wavelength of 2567.32Å, which is obtained by dividing the wavelength range covered by all the filters

[3]http://svo2.cab.inta-csic.es/theory/fps/

Filter	λ_{eff} (Å)
UVW2	2085.73
UVW1	2684.14
UVM2	2245.78
U	3751.0
B	4344.0
V	5456.0
R	6442.0
I	7994.0
J	12355.0
u	3546.0
g	4670.0
r	6156.0
i	7472.0
z	8917.0
y	10305.0

Table 5.2: The effective wavelengths λ_{eff} of the filters used to create flux heatmaps from light curves.

in table 5.2 by 4. We note that this value is arbitrary, and that other values of fixed characteristic length scale could be used. Boone (2019) find that their analysis on using Gaussian processes to model light curves is not sensitive to the choice of the length scale in wavelength, so we do not further investigate the choice of wavelength scale. We leave the time length scale as a trainable parameter. The Matérn 3/2 kernel has the form

$$k(r) = \sigma^2(1 + \sqrt{3r})\exp(-\sqrt{3r}) \tag{5.2}$$

where σ^2 is the variance parameter, which is left as a trainable parameter, and r is the Euclidean distance between the input points, scaled by a length scale parameter l (which can be set as a constant value or left as a trainable parameter):

$$r = \frac{x_1 - x_2}{l^2}. \tag{5.3}$$

The kernel used for the two-dimensional Gaussian process regression to model the light

curves in wavelength and time is

$$k_{\text{2D}} = \sigma^2 k_\lambda(r_\lambda) k_t(r_t) \tag{5.4}$$

where r_λ is the Euclidean distance between the wavelength input points, scaled by the fixed wavelength length scale parameter, and r_t is the Euclidean distance between the time input points, scaled by the time length scale parameter.

The two-dimensional Gaussian process is trained on each light curve in the dataset, and then used to predict flux measurements on a time-wavelength grid. The wavelength dimension in the grid runs from 2085.73Å to 10305.0Å divided into 25 bins resulting in a wavelength interval of 410.77Å, and the time dimension runs from 10 days before and 110 days after the first observation with an interval of 1 day. The resulting flux heatmap image has dimensions of 120×25 pixels, where each pixel represents a flux measurement. Figure 5.5 shows example flux heatmaps. The flux heatmaps are used as input for a convolutional neural network for classification.

5.3.3 Using two-dimensional Gaussian processes to infer spectra from light curves

Here, we present the use of two-dimensional Gaussian processes as a method to infer supernova spectra from their light curves. We select iPTF13bvn from the Open Supernova Catalog dataset as an example, a type Ib supernova that has good photometric coverage in time and across multiple filters. Figure 5.6 shows the light curve, and the corresponding flux heatmap generated with a two-dimensional Gaussian process.

We examine three spectra for iPTF13bvn, made available through the Open Supernova Catalog (Guillochon et al., 2017; Shivvers et al., 2019). To obtain the 'simulated' spectra from the flux heatmap, we take a single column at the time the spectra were taken, giving a vector that measures flux as a function of wavelength. The time of ob-

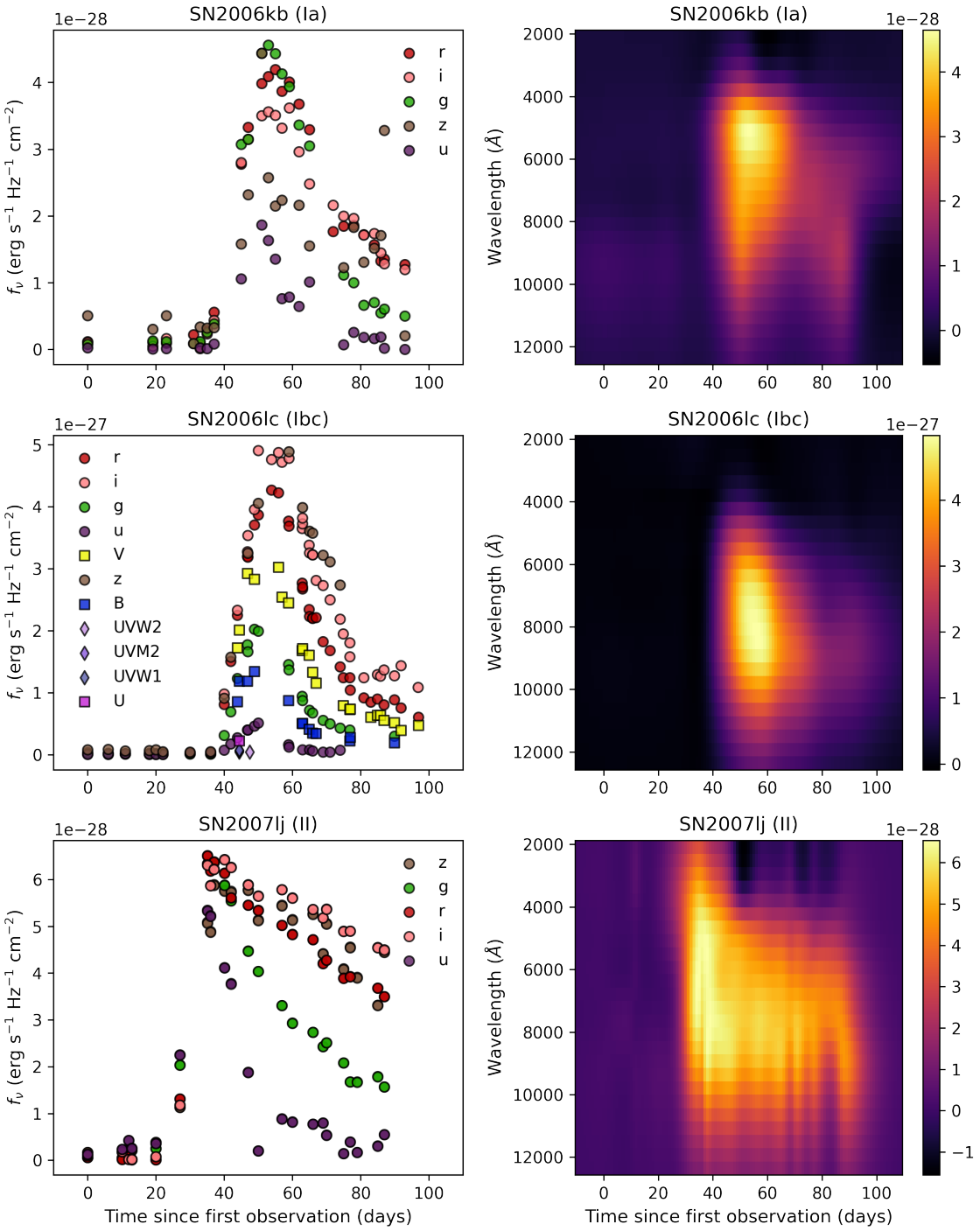


Figure 5.5: Examples of light curves (left) of SN2006kb (type Ia), SN2006lc (type Ibc), and SN2007lj (type II) and the corresponding flux heatmaps (right) generated from using a two-dimensional Gaussian process. The light curves are plotted as flux f_ν converted from AB magnitudes in each filter against time. The heatmaps show flux (brighter pixels indicating higher flux values) as a function of time (in days) and wavelength (in).

servation of the spectra is scaled to the time of first observation in the light curve, so it is given as the number of days since the first light curve observation. The real spectra for iPTF13bvn are taken at 20.6, 23.7, and 47.5 days after the first light curve observation, so the corresponding simulated spectra are obtained by taking columns from the flux heatmap at 20, 24, and 48 days after the first light curve observation. Figure 5.7 compares the real spectra of iPTF13bvn to the simulated spectra obtained from the flux heatmap.

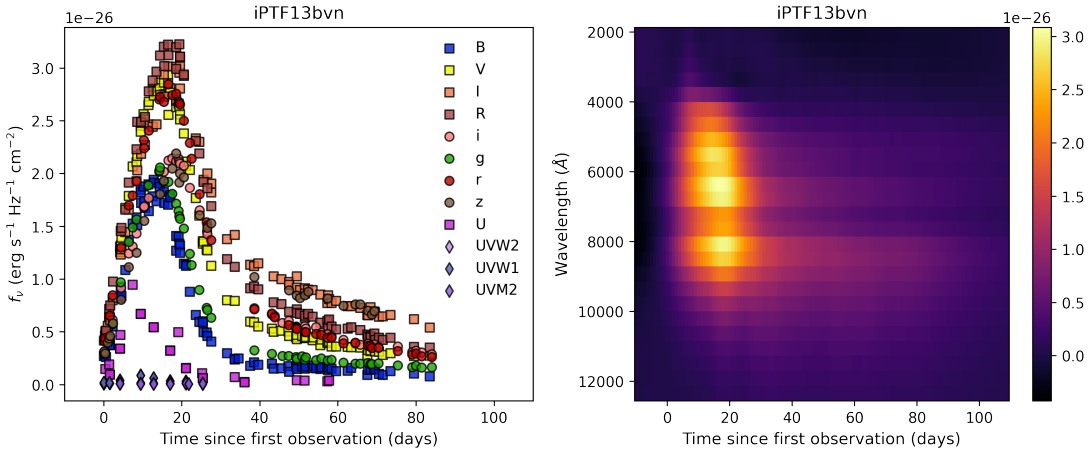


Figure 5.6: The light curve of the type Ib supernova iPTF13bvn (left) and its flux heatmap generated from the light curve (right).

From Figure 5.7, it can be seen that the spectra generated from the flux heatmap correlates quite well with the real spectra of iPTF13bvn. In all three spectra, the heatmap generated spectra appear to trace the continuum shape. For the spectra obtained at 47.5 days, the heatmap generated spectrum correlates with the Ca II IR triplet emission feature at $\sim$ 8700Å. Although there is a correlation, there is a poor match between the real spectrum and the heatmap generated spectrum which could be due to the width of the red filters (see Figure 5.1). Here, we have shown one example where the two-dimensional Gaussian process to create a flux heatmap can be used to generate low resolution spectra, provided there is good photometric coverage across multiple filters.

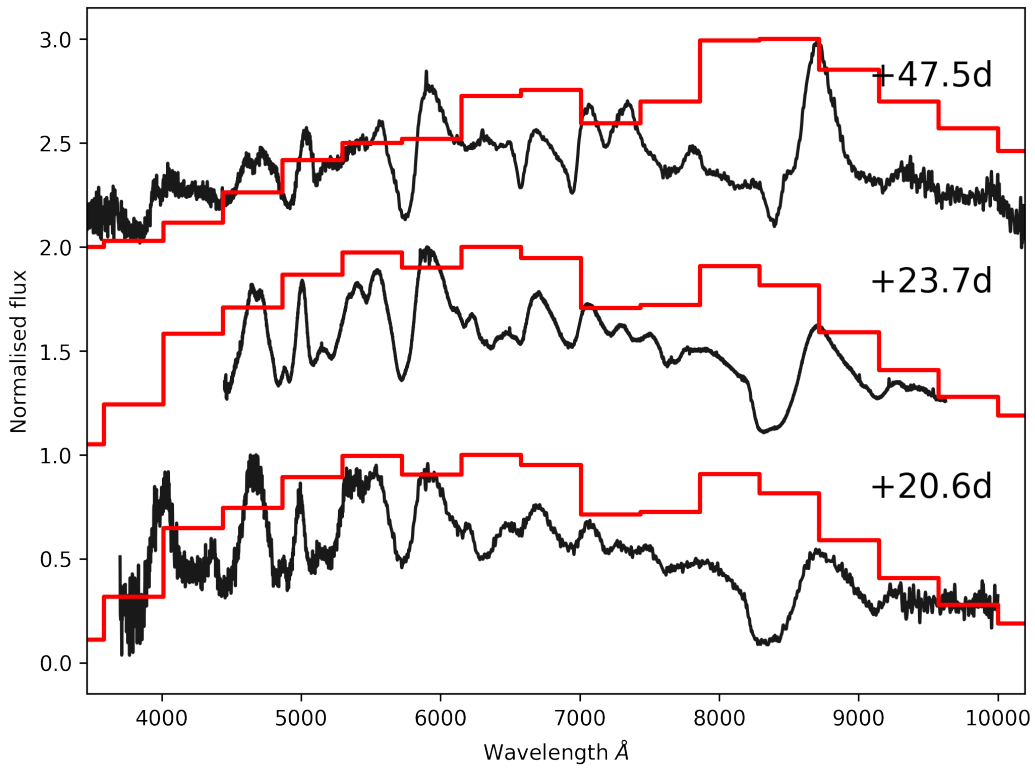


Figure 5.7: The spectra of iPTF13bvn are shown in black, and the spectra obtained from the flux heatmap are shown in red. The time of the spectra is given as days from the time of the first observation of the light curve. The spectra have been normalised (using the maximum value for each individual spectra) and shifted for clarity.

5.4 Convolutional neural networks

5.4.1 Model architecture

Convolutional neural networks (CNNs) are used to process data that have spatial features (e.g. a two-dimensional grid of pixels in an image, or a sequence of measurements in time-series data where there may be one or more measurements at each time step). CNNs were introduced in Chapter 2. We use a CNN to classify the flux heatmaps created from the Open Supernova Cataolog light curves (section 5.3.2) into three different classes: supernovae of types Ia, Ibc, and II. We build the CNN using the TensorFlow 2.0 package for Python (Abadi et al., 2016)[4] with Keras (Chollet et al., 2015) for implementation of

[4]https://www.tensorflow.org/

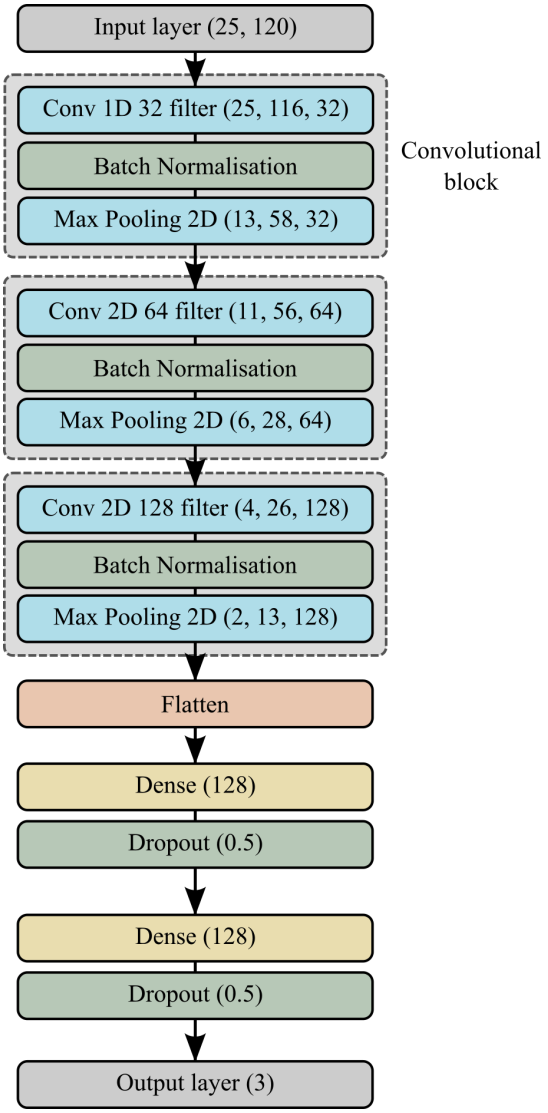


Figure 5.8: A diagram of the convolutional neural network. The grey dashed box indicates the layers that make up a convolutional block. The dimensions of the output tensors in the layers in the convolutional blocks (Conv 1D, Conv 2D, Max Pooling 2D), number of neurons in the dense layers (Dense), and the dropout fractions (Dropout) are shown in parentheses.

network layers.

The input to the CNN is a two-dimensional flux heatmap image of a supernova light curve, with dimensions 120×25 pixels where each pixel represents a flux value. All flux heatmaps are normalised by dividing by the highest flux value, so that the pixels in every heatmap have values between 0 and 1. One caveat of this approach is that information

about the supernova intrinsic brightness or distance is not included. We use a CNN with three convolutional blocks, followed by two fully-connected layers before the final output layer. Each convolutional block consists of a convolution layer, a batch normalisation layer, and a 2×2 max pooling layer. Figure 5.8 illustrates the model architecture used. For the convolutional and dense layers, the rectified linear unit (ReLU) activation function is used, and in the final output layer the softmax activation function is used to produce a list of probabilities that sum to unity. The probabilities returned by the model are scores that describe the level of 'belongingness' to a class.

In the first convolutional block we apply a one-dimensional convolution (also called a temporal convolution) in the time dimension instead of a standard two-dimensional convolution. This is done since the flux heatmap is generated from a light curve which measures the brightness of a supernova over time, so we attempt to extract temporal features in the first convolutional block. The output of each convolutional block has dimensions $(n_{\text{rows}}, n_{\text{columns}}, n_{\text{filters}})$, where n_{filters} is a convolutional layer parameter. Note here that n_{filters} refers to the number of kernels used to generate feature maps as explained in Chapter 2. In the second and third convolutional blocks, a two-dimensional convolution is applied to the output of the preceding convolutional block. Table 5.3 lists the series of convolutions and max pooling applied in the convolutional blocks, with the corresponding layer parameters and output dimensions at each stage.

The output of the last convolutional block is then flattened into a one-dimensional vector and then passed on to two fully-connected layers, each with dropout applied with the dropout fraction set to 0.5. We apply a L_2 regularization in the second fully-connected layer with a regularization parameter of 0.01, which is the default TensorFlow value. The final output layer is a fully-connected layer with the same number of neurons as the number of classes, which is three. In total, the CNN model has 536,003 trainable parameters.

Layer	Kernel/Pool size	Filters	Output dimension
Conv 1D	(5)	32	(25, 116, 32)
BatchNorm	-	-	(25, 116, 32)
MaxPool 2D	(2,2)	-	(13, 58, 32)
Conv 2D	(3,3)	64	(11, 56, 64)
BatchNorm	-	-	(11, 56, 64)
MaxPool 2D	(2,2)	-	(6, 28, 64)
Conv 2D	(3,3)	128	(4, 26, 128)
BatchNorm	-	-	(4, 26, 128)
MaxPool 2D	(2,2)	-	(2, 13, 128)
Flatten	-	-	3328

Table 5.3: The layer parameters and output dimension for each layer in the convolutional blocks. For the convolutional layers, the kernel size is the shape of the convolutional window and filters sets the number of convolutional filters that are learnt during training. For the max pooling layers, the pool size sets the shape of the window over which to take the maximum. The number of strides is one for the convolutional layers and two for the max pooling layers. The flattening layer takes the multidimensional output of the convolutions and shapes into a single dimensional output.

5.4.2 Model training

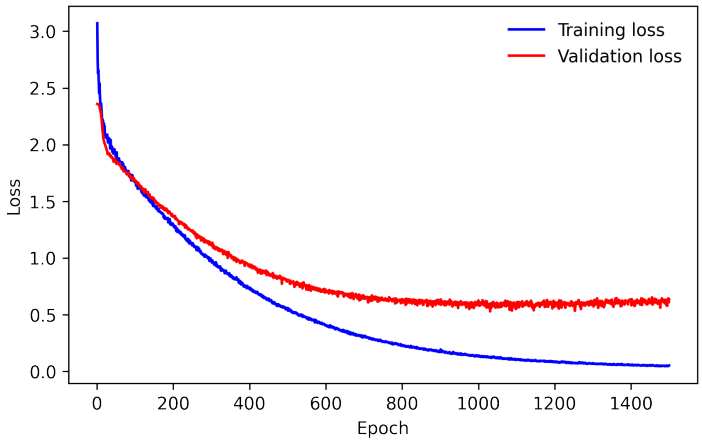


Figure 5.9: The training and validation loss for the CNN model trained on the Open Supernova Catalog data.

The CNN model is trained on the Open Supernova Catalog flux heatmaps with a learning rate of 1×10^{-5} for 1500 epochs with the Adam optimizer (Kingma & Ba, 2014), using a batch size of 128. Figure 5.9 shows how the training and validation loss

evolve with training. Within 1500 epochs of training, both the training and validation loss begin to converge (i.e. stops improving). We use the categorical cross entropy loss function, weighted to take into account the class imbalance present in the data. The class weight α_i for class i is

$$\alpha_i = \frac{1}{n} \times \frac{N}{N_i} \tag{5.5}$$

where n is the total number of classes, N is the total number of samples in the dataset, and N_i is the number of samples in class i. The class weights are obtained using samples in the training set.

The model is trained on an NVIDIA Quadro P2200 graphics processing unit with 1280 cores and 5GB of memory, which takes 4 seconds per epoch for a total time of $\sim$ 100 minutes to train the model.

5.5 Results on classifying Open Supernova Catalog data

Once the model has been trained, it is then used to make predictions on the test set. The test set consists of data that is kept apart from the training and validation sets, and used to evaluate how well the model is able to generalize on unseen data. On the testset, the model achieves an area under the receiver operating characteristic curve (AUC) score of 0.859, and an F_1 score of 0.708. Figure 5.10 shows the confusion matrix for the model evaluated with the test set.

From the confusion matrix, the model shows good classification of Type Ia and II supernovae with 92% (812) and 89% (586) accuracy for each class, respectively. The performance for Type Ibc supernovae is poor, with the model only achieving 26% (11) accuracy for that class and misclassifying 65% (28) of Type Ibc supernovae as Type Ia. This may be due to the small number of samples of type Ibc supernovae in the dataset, and the fact that it is also the class with the smallest number of samples. The

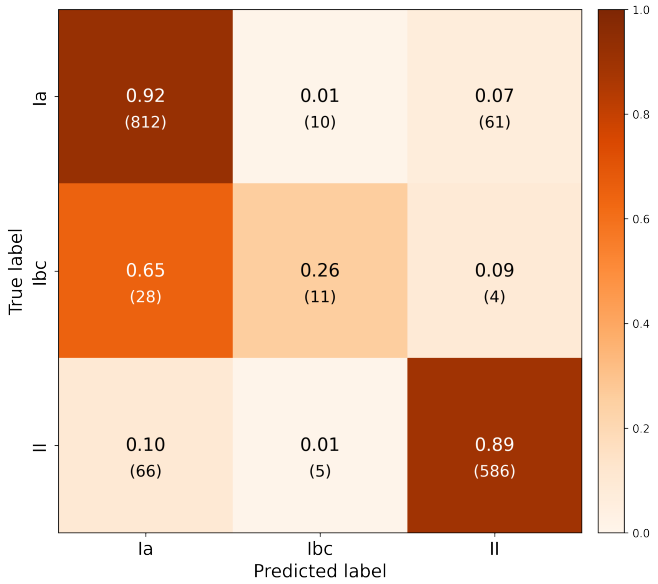


Figure 5.10: Confusion matrix for the test set of heatmaps generated from the Open Supernova Catalog light curves. The y-axis shows the true class label, and the x-axis shows the class label predicted by the model. Entries along the diagonal represent where the predicted label matches the true label, and the off-diagonal entries show where misclassifications occur. Reading along the rows, the fractional values show how samples from a class have been classified, with the absolute numbers below in parentheses.

majority of Type Ibc supernovae are misclassified as Type Ia, and it is known that it can be challenging to differentiate between Type Ia and Type Ibc with only photometry (Lochner et al., 2016). The misclassifications between Type Ia and Type II are quite low, with $\lesssim$ 10% (61 for Type Ia and 66 for Type II) of each being misclassified as the other.

5.6 Transfer learning

5.6.1 Overview

In the case of a classification task where there is a lack of labelled training data, the ability to transfer classification knowledge from one domain to the new one is useful. In astronomy, new surveys can experience the problem of a small or complete lack of a

labelled training set since it can take time to accumulate enough sources and also label them (e.g. using spectroscopy or visual inspection of the photometry). In the following sections, we present the application of transfer learning to classify supernova light curves from the Photometric LSST Astronomical Time Series Classification Challenge (PLAsTiCC) dataset (The PLAsTiCC team et al., 2018) by using classification knowledge from Open Supernova Catalog light curves presented in the previous sections.

Transfer learning is defined as improving the learning of a target predictive function (e.g classification, mapping inputs to a class) in a target domain $\mathcal{D}_T$ using knowledge from a source domain $\mathcal{D}_S$ and source task $\mathcal{T}_S$ (Pan & Yang, 2010). In this case, the target domain is the PLAsTiCC dataset, the source domain is the Open Supernova Catalog dataset, the source task is classifying Open Supernova Catalog light curves into one of three classes (Ia, Ibc, II), and the target predictive function is classifying light curves from the PLAsTiCC dataset.

5.6.2 The new classification task

Transfer learning can be used to borrow classification knowledge from one task in one domain to another task in another domain, As defined above, the domains are the two different datasets. We define a new classification task for the PLAsTiCC dataset, which is different to the classification task presented in section 5.2. We select only supernova from the PLAsTiCC dataset, and define six classifications (based on the PLAsTiCC defined classifications in The PLAsTiCC team et al. (2018)): types Ia, Iax, Ia-91bg, Ibc, II, SLSN-I. The classifications now divide type Ia supernovae into three sub-classes, and and also include a new class, type I superluminous supernova (SLSN-I).

5.7 PLAsTiCC data

The Photometric LSST Astronomical Time Series Classification Challenge (PLAsTiCC) was launched in 2018 to challenge participants from the wider science community (open to not just astronomers but experts in other fields such as computer science) to develop classification algorithms or models to classify a large dataset of simulated LSST observations (The PLAsTiCC team et al., 2018). The dataset consisted of over 3.5 million objects with a total of over 450 million observations, divided into a wide range of classes (supernovae of various types, variable objects, tidal disruption events, and more), each with light curves in six filters (LSST *ugrizy*) that include the fluxes and corresponding errors, with the time of observation. The dataset contains labelled $\sim 8,000$ objects with known types that formed a training set, which is $\sim 0.2\%$ of the total dataset, with the remainder comprising the test set. For each object, contextual information such as the R.A. and Dec, galactic latitude and longitude, and host galaxy spectroscopic and photometric redshifts were available. Each object also had a flag indicating whether it was observed under the wide-fast-deep (WFD) main survey, or was observed in one of the LSST deep drilling fields (DDF).

The goals of the challenge were to identify classification methodologies that were able to:

- Classify objects using photometric information in six filters
- Identify interesting objects for additional follow-up with spectroscopic instruments

The motivation for photometric classifiation arises from the fact that the number of expected LSST nightly discoveries (approximately on the order of 10^6) greatly exceeds the available spectroscopic resources. The light curves were generated using transient and variable source models provided by the astronomical community, coupled with an LSST operations simulator to generate realistic observing conditions (Kessler et al.,

2019). The PLAsTiCC dataset presents its own unique set of challenges, such as the presence of 'seasonal gaps' in the light curves where an object is not visible during the observation campaign, a wide distribution of class sizes (with some classes having only hundreds of examples vs. others having millions), and a training set that is not representative in redshfit of the test set (to simulate a realistic training set obtained from a spectroscopically confirmed sample).

5.7.1 Data selection

From the original PLAsTiCC dataset, we select only supernova objects of six types: Ia, Iax, Ia-91bg, Ibc, II, SLSN-I. The light curves in the PLAsTiCC dataset span the duration of the observing campaign and feature seasonal gaps when an object is not observable. We use only photometry obtained from the image-subtraction pipeline (using the flag `detected_bool` = 1), which removes the seasonal gaps and produces light curves covering the period of supernova rise and decline. We also select only observations from up to 20 days before and 100 days after the peak (which is taken as the maximum flux measurement in any filter).

After applying the selection cuts, the final dataset consists of 397,990 objects with 2,398 remaining from the original training set. For the transfer learning process, we use two training sets and compare their performance. The first is the original training set, and the second is an augmented training set which is obtained by randomly sampling 3% of the test set added to the original training set. No stratification is used when sampling the test set, so the proportion of the six classes is unchanged, still presenting a class imbalance problem. The test set with the 3% removed for creation of the augmented training set is used as the test set for both the original training set and the augmented training set, so that the models trained on the two training sets are evaluated on the same test set. In both training sets, 10% is used for validation. Table 5.4 shows the breakdown

of the PLAsTiCC dataset. The original PLAsTiCC training set contains fewer samples than the Open Supernova Catalog training set. This transfer learning approach emulates using classification knowledge from another domain after a small labelled training set has been obtained for a new survey.

Type	**Training 1**	**Training 2**	**Test**	**All data**
Ia	1,136 (47.4%)	7,168 (50.5%)	197,884 (51.9%)	206,188 (51.8%)
Iax	115 (4.8%)	341 (2.4%)	6,730 (1.8%)	7,186 (1.8%)
Ia-91bg	78 (3.3%)	197 (1.4%)	4,151 (1.1%)	4,426 (1.1%)
Ibc	259 (10.8%)	1,082 (7.6%)	26,271 (6.9%)	27,612 (6.9%)
II	670 (27.9%)	4,735 (33.4%)	128,958 (33.8%)	134,363 (33.8%)
SLSN-I	140 (5.8%)	671 (4.1%)	17,404 (4.6%)	18,215 (4.6%)
Total:	2,398	14,194	381,398	397,990

Table 5.4: Breakdown of the PLAsTiCC dataset by type. The column labelled 'Training 1' shows the original training set, and the column labelled 'Training 2' shows the augmented training set.

5.7.2 Creating heatmaps

We follow the same steps outlined in section 5.3, and use a two-dimensional Gaussian process to generate heatmaps from the PLAsTiCC supernova light curves. The time of observation was scaled so that the time of the first observation is $t = 0$. Each flux measurement in a light curve are labelled with the time it was observed, and the effective wavelength λ_{eff} of LSST filter it was observed in. Table 5.5 lists the effective wavelengths for the LSST filters.

The light curves were then used to train a two-dimensional Gaussian process to create flux heatmaps. A fixed characteristic wavelength scale of 2980.09Å was used, obtained by dividing the wavelength range coverage of the filters by two. The time length scale and variance parameter were left as trainable parameters. The flux heatmaps were generated onto a grid, where $-5 < t < 115$ with a one-day interval and wavelength runs from 3751.36Å to 9711.53Å , divided into 25 bins giving an interval of 238.81Å. The flux

Filter	λ_{eff} **(Å)**
u	3751.36
g	4741.64
r	6173.23
i	7501.62
z	8679.19
y	9711.53

Table 5.5: The effective wavelength λ_{eff} of the LSST filters used to simulate observations in the PLAsTiCC dataset. The values were obtained from the SVO Filter Profile service Rodrigo & Solano (2020).

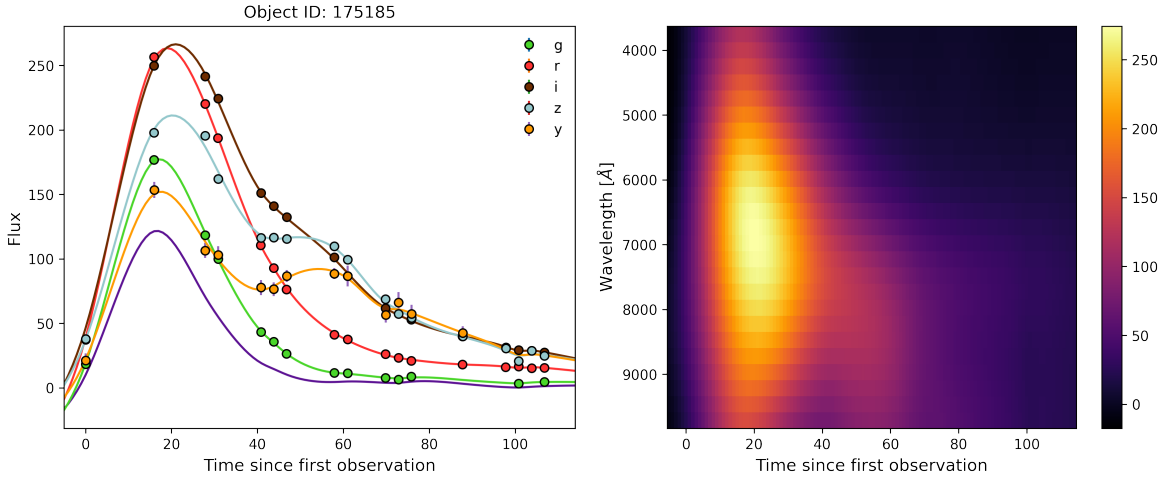


Figure 5.11: An example of a type Ia supernova light curve (left) and the flux heatmap generated from the light curve (right). The interpolated flux from the two-dimensional Gaussian process at the wavelength corresponding to the filter effective wavelength is also plotted.

heatmaps have dimensions of 120×25 pixels, where each pixel represents a flux value. Figure 5.11 shows an example light curve and the flux heatmap generated using the two-dimensional Gaussian process.

5.7.3 Applying transfer learning to PLAsTiCC light curves

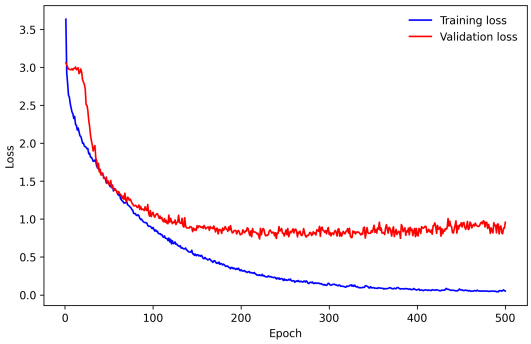
We compare two models on their classification performance on the PLAsTiCC dataset, one with transfer learning and without. In both cases we examine how including redshift information and using the augmented training set affects performance. We use the

estimated host galaxy photometric redshift value in the PLAsTiCC data listed in the `hostgal_photoz` column. We use the same CNN model architecture presented in section 5.4, but change the output layer to have six neurons (for the six classes in the PLAsTiCC classification task). For the models that include redshift information, we append the redshift value to the flattened output of the last convolutional block.

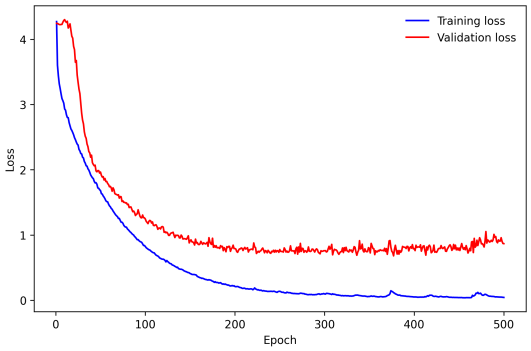
Transfer learning is implemented by setting the parameters of the convolutional block as non-trainable parameters, a method known as 'freezing' layers in a neural network. The parameters in the convolutional blocks are fixed, and the model only changes the parameters in the dense layers during training. The idea is that the 'knowledge' of extracting salient features from the heatmaps developed in the convolutional blocks of the model trained on the Open Supernova Catalog is used to extract features from heatmaps in the PLAsTiCC dataset. Since the only trainable parameters are those in the dense layers, the model is then just tasked with learning the feature-class relationship to group the data into different classes using the features extracted from the heatmaps.

For the models without transfer learning, all parameters are left as trainable parameters. In this case, the model has to learn to extract features from heatmaps in the convolutional blocks as well as the feature-class relationship in the dense layers to classify the heatmaps into the six classes. Since the models used in transfer learning have fewer trainable parameters, the time needed to train them is less than the time needed to train the models without transfer learning. The models were trained on a NVIDIA Quadro P2200 graphics processing unit with 1280 cores and 5GB of memory, and the models with transfer learning required 0.29s per epoch of training, while the model without transfer learning required 0.51s per epoch. With transfer learning, the models could be trained $\sim 56\%$ faster. All models are trained for 500 epochs with a learning rate of 1×10^{-4}. Figure 5.12 shows the training and validation loss for models trained without transfer learning, and Figure 5.13 shows the training and validation loss for models trained with transfer learning. From these figures, it can be seen that 500

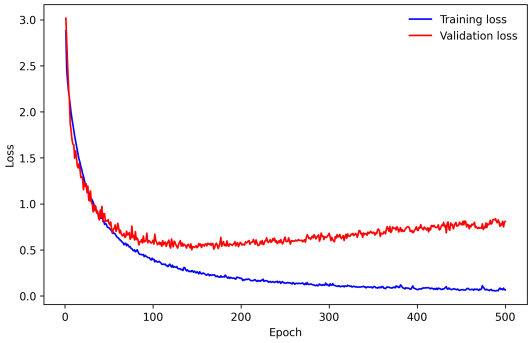
epochs is sufficient for the models to converge. The models trained on the augmented training set without transfer learning suffer from overfitting, where the validation loss begins to increase as the training loss continues to decrease.



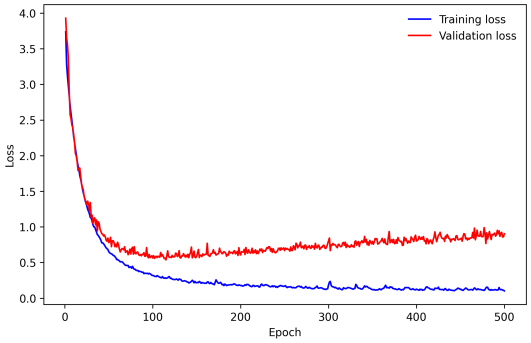
(a) Original training set, no redshift



(b) Original training set, with redshift

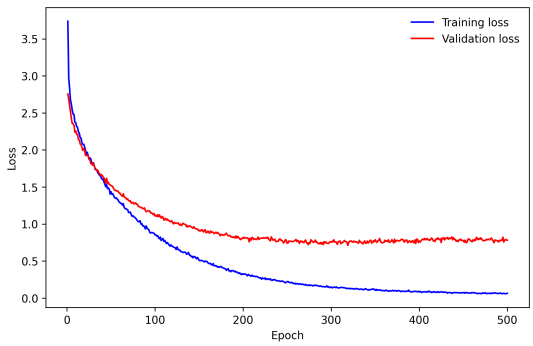


(c) Augmented training set, no redshift

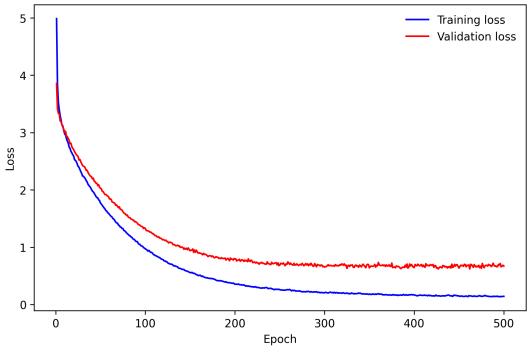


(d) Augmented training set, with redshift

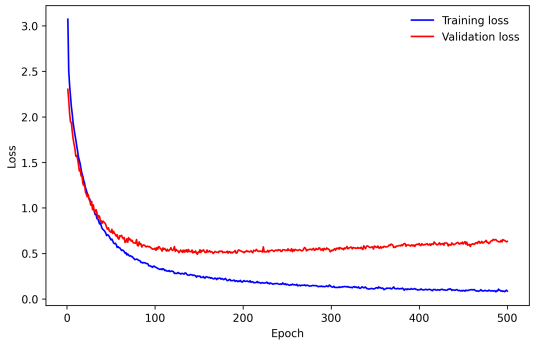
Figure 5.12: Training and validation loss during training for models without transfer learning.



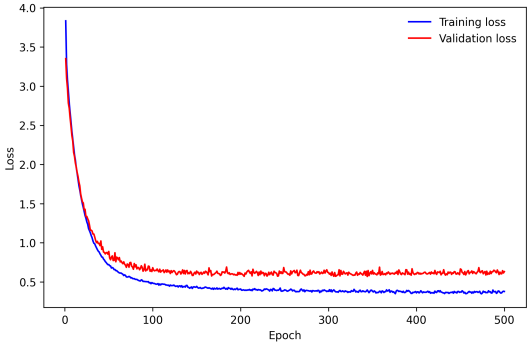
(a) Original training set, no redshift



(b) Original training set, with redshift



(c) Augmented training set, no redshift



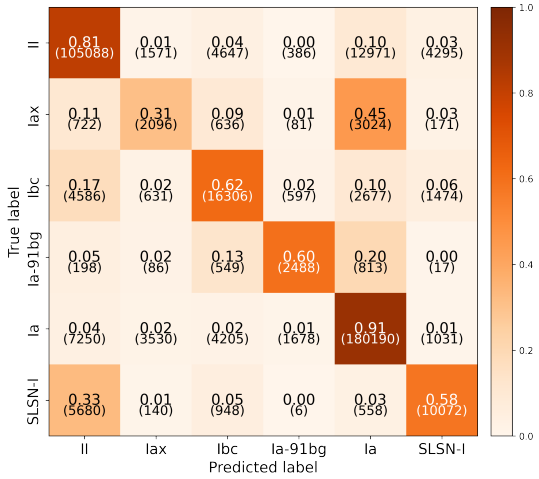
(d) Augmented training set, with redshift

Figure 5.13: Training and validation loss during training for models with transfer learning.

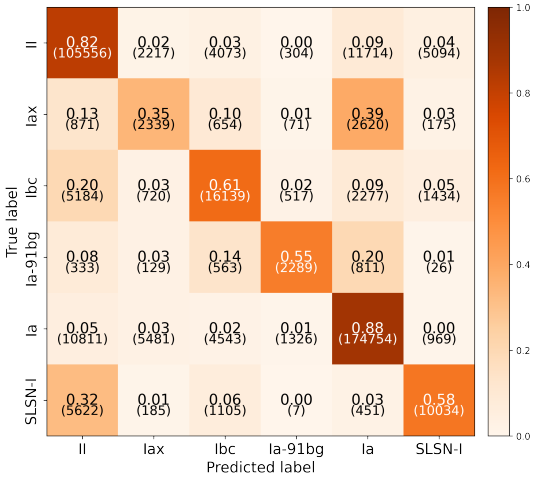
5.8 Results on classifying PLAsTiCC light curves with transfer learning

5.8.1 Models without transfer learning

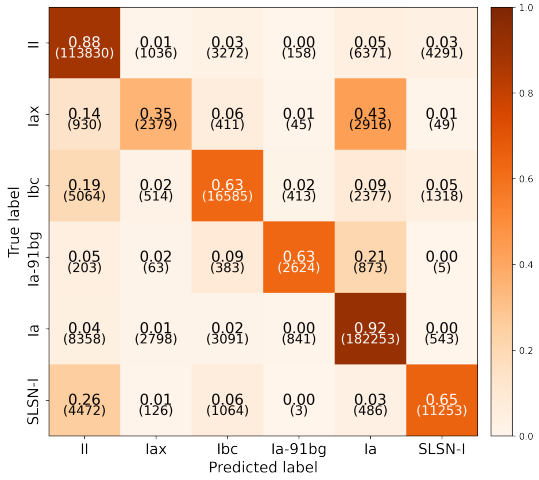
After training, all models were evaluated on the test set. Figure 5.14 shows the confusion matrices for the models trained without transfer learning, using the original and augmented training set, with and without redshift information. Looking at the confusion matrices for the original training set, the model achieves good accuracy ($> 80\%$) for type Ia and II supernovae, a medium level of accuracy for type Ibc, Ia-91bg, and SLSN-I ($> 60\%$), and poor accuracy for type Iax supernovae. The biggest sources of confusion are type Iax and Ia-91bg being classified as Ia, and type Iax, Ibc, and SLSN-I being



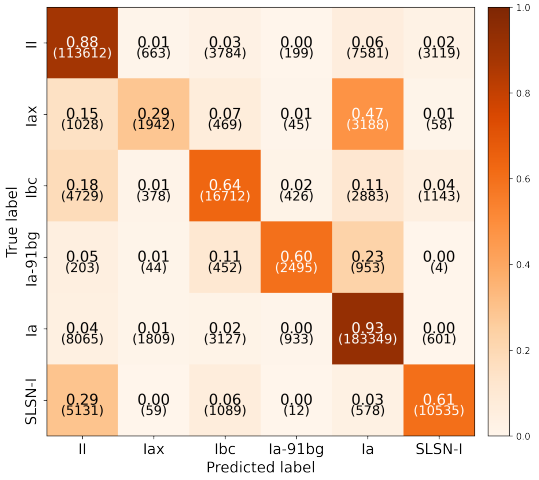
(a) Original training set, no redshift



(b) Original training set, with redshift



(c) Augmented training set, no redshift



(d) Augmented training set, with redshift

Figure 5.14: Confusion matrices for models without transfer learning, evaluated on the test set.

classified as type II. When redshift information is included, there is no major sign of improvement in performance.

When the augmented training set is used, there is a slight improvement in accuracy for type II supernovae (and increase of $\sim$ 7% in accuracy), but no major change in performance in the other classes. Including redshift information does not have any significant improvement over the model without redshift information. There is an increase in the number of type Iax being classified as type Ia, and fewer SLSN-I being classified

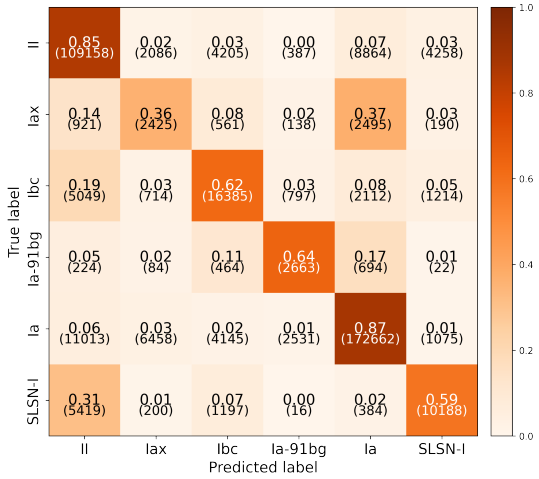
as type II.

5.8.2 Models with transfer learning

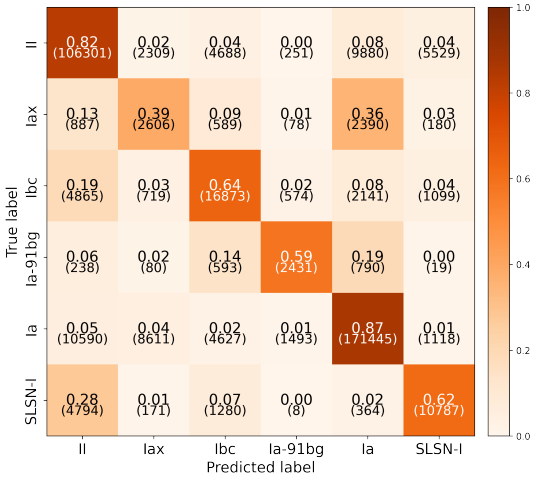
Figure 5.15 shows the confusion matrices trained with transfer learning, using the original and augmented training set, with and without redshift information. For models trained on the original training set, both have slightly overall better performance over the same models without transfer learning, with an increase in accuracy by a few percent across most classes, and fewer misclassifications. When including redshift, there is no significant improvement in performance.

Looking at the models trained with the augmented training set, the performance for the model without redshift information is similar to the performance for the same model without transfer learning. When redshift information is included, the performance of the model with transfer learning is improved over the same model without transfer learning. There is good accuracy for type Ia and II supernovae ($> 80\%$), and improved accuracy for type Ibc, Ia-91-bg, and SLSN-I ($> 70\%$). There are fewer misclassifications overall ($< 15\%$), and the model trained with transfer learning and redshift information achieves the best accuracy out of all models on type Iax (45%). We plot the difference between the confusion matrices for the models trained with transfer learning and without, for the augmented training set with redshift in Figure 5.16, to illustrate the change in performance between the two models.

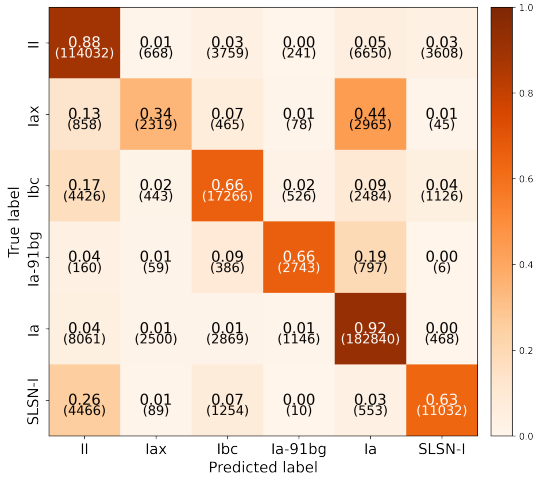
Table 5.6 shows the area under the receiver operating characteristic curve (AUC) score and the F_1 score for all trained models. The Hand & Till (2001) formulation is used to obtain the multi-class AUC scores presented in Table 5.6. In both cases with models trained with and without transfer learning, including redshift shows an improvement in the AUC score, but not always an improvement in the F_1 score. A higher AUC score indicates that the model is able to produce fewer false positives, so when redshift is



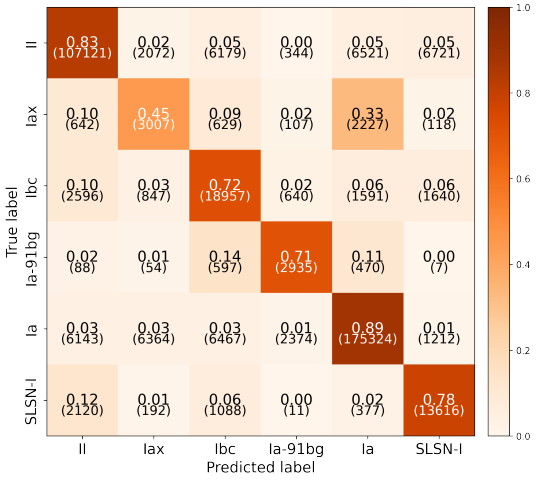
(a) Original training set, no redshift



(b) Original training set, with redshift



(c) Augmented training set, no redshift



(d) Augmented training set, with redshift

Figure 5.15: Confusion matrices for models with transfer learning, evaluated on the test set.

included the models are able to make predictions that have slightly less contamination at the small cost of not correctly classifying all true positive samples in each class.

We also examine how selecting a threshold for class membership reduces the number of false positives in each class. Since the model makes predictions by producing a list of scores that represent how likely an object belongs to a specific class, we can define a threshold score so that if the score is above the threshold then the object belongs to that class, and if it is below then it is considered to not belong to that class. Three

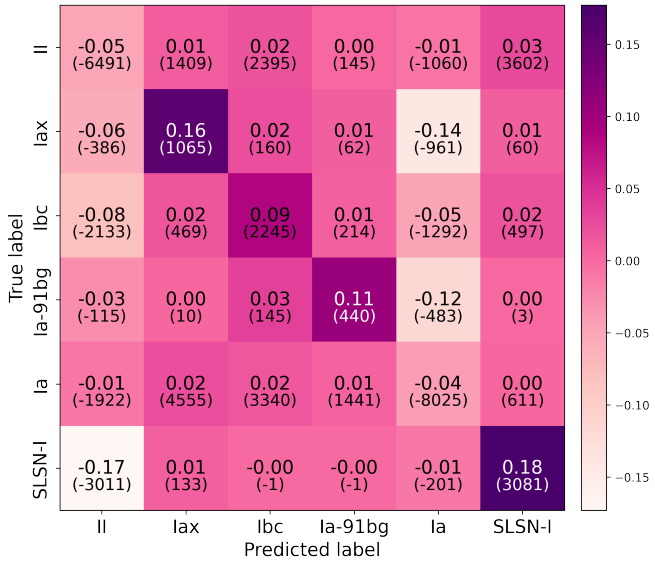
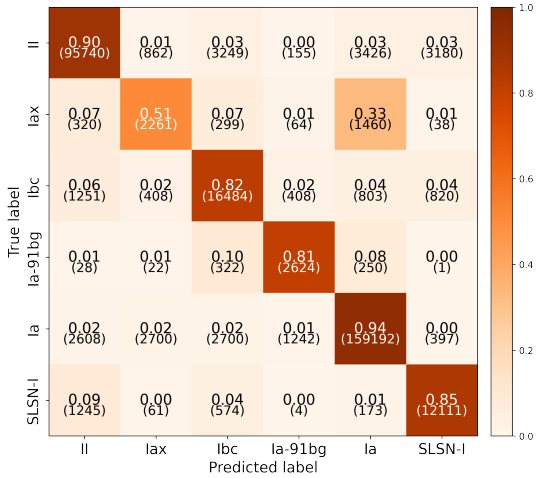


Figure 5.16: The difference between the confusion matrix for models trained on the augmented training set with redshift for with and without transfer learning. Positive values along the diagonal indicate an improvement when transfer learning is used. Negative values in the off diagonals indicate fewer misclassifications.

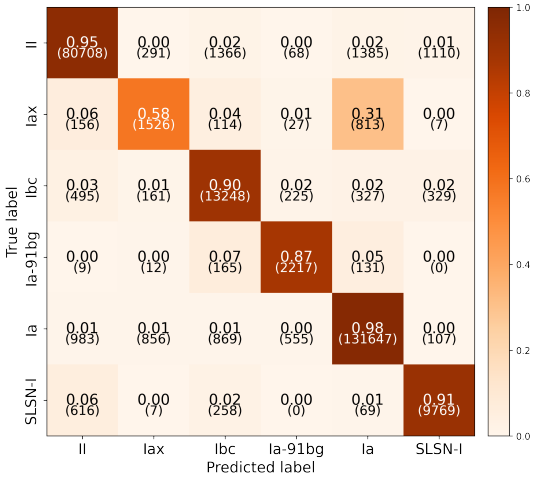
threshold values are selected: 0.5, 0.7, and 0.9. We consider the model trained with transfer learning using the augmented training set and redshift information. For each threshold value, any predictions that are below the threshold are excluded (looking at the highest score out of the six classes). Table 5.7 lists the AUC score, F_1 score, and the fraction of samples retained at the different threshold values. Figure 5.17 shows the confusion matrices for thresholds at 0.7 and 0.9. As the threshold value increases, the AUC score and F_1 score improves but the fraction of samples retained decreases.

	Model	AUC	F_1
No transfer learning	Original, no redshift	0.893	0.624
	Original, with redshift	0.895	0.614
	Augmented, no redshift	0.924	0.680
	Augmented, with redshift	0.921	0.669
With transfer learning	Original, no redshift	0.901	0.617
	Original, with redshift	0.915	0.620
	Augmented, no redshift	0.925	0.683
	Augmented, with redshift	0.945	0.657

Table 5.6: AUC and F_1 scores, trained on the original and augmented training sets with and without redshift, for both with and without transfer learning.



(a) Augmented training set, with redshift, at a threshold of 0.7



(b) Augmented training set, with redshift, at a threshold of 0.9

Figure 5.17: Confusion matrices for the model with pre-training, trained on the augmented training set with redshift at different thresholds.

Threshold	AUC	F_1	Fraction retained
0.5	0.950	0.684	95.6%
0.7	0.960	0.752	83.2%
0.9	0.971	0.838	65.7%

Table 5.7: AUC and F_1 scores for the transfer learning model trained on the augmented training set with redshift, evaluated at different probability thresholds. The column on the right shows the fraction of the test set retained when discarding predictions that are below the threshold.

5.9 Discussion

Here, we provide a discussion on the work presented in this chapter on how to classify supernova light curves from multiple surveys, using a Gaussian process to create an image representation of light curves, and how transfer learning can be useful when developing classifiers for new surveys.

5.9.1 Classifying supernovae from multiple surveys

In order to classify the heterogeneous supernova light curve dataset, we use a two-dimensional Gaussian process to model the light curves, and create a flux heatmap image for each light curve where each pixel in the image represents flux, as a function of time and wavelength. We also show that in the case where a supernova has good photometric coverage in multiple filters (measuring the flux at different wavelengths), the Gaussian process can be used to generate a low-resolution spectra of the supernova. Comparing the real spectra and Gaussian process generated spectra of supernova iPTF13bvn, we find that the two are comparable and the generated spectra does resemble the real spectra. It is not guaranteed, however, that all supernovae will have the same quality of photometric observations. Out of the original $\sim 80,000$ supernova light curves from the Open Supernova Catalog, only 6,330 were used to generate flux heatmaps with a a two-dimensional Gaussian process after selection cuts. A larger sample could be used, but at the cost of lower quality light curves (i.e. poor sampling in time and lack of multi-colour observations) which may result in poor fitting with Gaussian processes.

We used a convolutional neural network to classify the supernova flux heatmaps, since the data is in a grid format which is well suited for the convolution operations carried out in the neural network. The model is able to classify type Ia and II supernovae with good accuracy, but the class imbalance in the dataset presents a challenge for classifying type Ibc supernovae, since it is the class with the smallest number of samples and is

not well represented in the training set. Deep learning approaches benefit from having a large dataset to learn from, and we note that the Open Supernova Catalog dataset is rather small for a deep learning application with less than 4,000 samples in the training set.

5.9.2 Transfer learning for future surveys

We used a subset of 397,990 supernova light curves from the PLAsTiCC dataset (The PLAsTiCC team et al., 2018), and use the two-dimensional Gaussian process to generate flux heatmaps from the light curves. For the PLAsTiCC dataset we split the data into six classes, presenting a different classification task than the one for the Open Supernova Catalog dataset. The original training set (containing 2,398 SNe) and an augmented training set (containing 14,914) are used. Typically in most machine learning and deep learning methods, the training set is larger than the test set. Here, we use training sets that are much smaller than the test set to emulate the case where there is a scarcity of labelled data (the training set) and a large amount of unlabelled data (the test set).

We demonstrate that it is possible to transfer knowledge between two different domains (Open Supernova Catalog data and PLAsTiCC data) and two different classification tasks (three classes to six classes). The use of transfer learning shows a small improvement over when no transfer learning is used (and the model is only trained on the PLAsTiCC training set). We find the best increase in performance comes when redshift information is included and the augmented training set is used. It is possible to obtain better classifications with fewer misclassifications between classes when a threshold is used to remove 'unconfident' classifications provided by the classifier.

A limitation of the two-dimensional Gaussian process approach used in this work is the requirement for sufficiently good coverage across multiple photometric bands and also in time. For future surveys such as LSST, this is dependent on the choice of observing

strategy to provide a good enough cadence and wavelength coverage. The use of a two-dimensional Gaussian process also relies on the full supernova light curve to create a good flux heatmap representation.

5.10 Conclusion

In this chapter, we present an approach to classify Open Supernova Catalog light curves from multiple surveys with a convolutional neural network by using a two-dimensional Gaussian process to generate an image representation of supernova light curves. We find that using this method achieves good classification when there is good representation of the data in the training set. In the case of type Ibc classification, the performance is poor since there is a lack of representation of type Ibc supernovae in the training set. For classification tasks, it is important to have a good representative training set with good coverage in feature space for all classes so that a model is able to learn the feature-class relationship to make robust classifications.

We then investigate the usefulness of transfer learning in the context of future surveys where theyre may be a lack of labelled data to form a training set with which to train classifiers. The use of transfer learning shows a small improvement in classifiers compared to when no transfer learning is used when classifying PLAsTiCC supernova light curves. The addition of contextual information such as redshift and an augmented training set provided the best improvement in classification performance, highlighting the importance of a representative training set and the benefits of incorporating contextual information when classifying light curves.

The methods presented in this chapter could also be extended to classifying light curves of other non-supernova objects (such as variable stars, flare events, and AGN). The flux heatmaps generated with the two-dimensional Gaussian process could be used with a different neural network architecture such as a recurrent neural network, where

each the input at each time step is a single column of the heatmap representing the flux interpolated along wavelength. This would allow classifications to be obtained with time, and also be used to classify partial light curves (unlike the full light curves used in this chapter), where the Gaussian process is used to interpolate the light curves up to the most recent observation as in Qu & Sako (2021).

A classification model that is agnostic to the different filters used across different surveys would be useful in the near future of time-domain astronomy. New objects observed by surveys such as LSST with the Vera Rubin Observatory could trigger follow-up observations by various instruments world wide, which could be ingested by such a model to provide fast early-time classifications to identify good candidates for time-sensitive observations.

Chapter 6

Conclusions

In this final chapter we summarise the work presented in this thesis and provide some concluding remarks of the results and findings of our work. We also present how the recurrent neural network classifier presented in Chapter 4 can be adapted for two other use cases (one in astronomy and one in the medical field), highlighting the flexibility and utility of the classifier. We end with a discussion on some possible limitations of the work in this thesis, and possible directions for future work.

6.1 Time-series classification in time-domain astronomy

We presented the use of a machine learning algorithm, the random forest, to classify light curves from the Gravitational-wave Optical Transient Observer (GOTO) survey in Chapter 3, and the use of a recurrent neural network to classify light curves from the same survey but on a larger dataset in Chapter 4. In both instances, the datasets used presented a class imbalance problem where some classes had much fewer examples compared to other classes, and a combination of time-series information as well contextual information were used to perform the classification. Here, we summarise the results of classification on GOTO light curves, and key findings for the work presented in Chapters 3 and 4.

For the random forest classifier, the dataset of GOTO light curves consisted of 10,200 labelled light curves, divided into six classes. A set of handcrafted features were extracted from the light curves to create a set of features that were used as inputs for the random forest, including time-series features from the light curves, as well as additional contextual information such as on-sky position and association with any nearby galaxies. To deal with the imbalanced dataset, data augmentation was performed by drawing synthetic samples in the feature space to increase the number of examples from minority

classes. Two random forest classifiers were trained on the original training set and on the augmented training set. In both cases, the random forest provided adequate classification for four out of seven classes (achieving between 56% to 79% accuracy for eclipsing binaries, long-period variables, RR Lyrae stars and supernovae) but performed poorly on the other two classes (which were classes with relatively few examples). Training on the augmented training set showed an improvement for supernova classification. Exploring the feature importance showed that contextual features were useful in discriminating between Galactic sources and extra-galactic transients. Although useful, the random forest classifier requires a feature extraction step on complete light curves and can only provide classifications after an object has been observed for an extended duration.

A larger dataset of GOTO light curves was used to train a recurrent neural network classifier, comprising light curves of 99,201 labelled objects, almost an order of magnitude larger than the dataset used for the random forest classifier. The dataset was divided into three broad but distinct classes: variable stars, active galactic nuclei, and supernovae. Two separate inputs were provided to the recurrent neural network: the light curves as a time-series input, and contextual information about the object. An alternative method was used to deal with class imbalance to the data-level approach used for the random forest classifier; the recurrent neural network was trained with loss functions weighted to the number of samples in each class and the focal loss function (Lin et al., 2017) to heavily penalise misclassified samples from minority classes. The recurrent neural network classifier provided good classification for all three classes, and is able to provide good classification on incomplete light curves with only a few observations. Recurrent neural networks were trained with and without contextual information, and our findings showed that classification performance improves when contextual information was included.

When developing classifiers for time-domain optical surveys, it is important to have a representative training set with which to train the classifier. In Chapter 3, some classes had an inadequate number of examples for the random forest classifier to learn from,

resulting in confusion between those minority classes with majority classes that had many more examples. In Chapter 4, although there was still a class imbalance problem present, each class in the training set contained enough examples so that the recurrent neural network classifier could learn to differentiate between the different classes. It also helped to reformulate the classification problem by grouping together similar classes (e.g. different types of variable stars/eclipsing binaries into a single class) so that the classification task is not too complex.

For time-series photometric classification, it is useful to include additional contextual information relating to the objects to be classified where possible. Non time-dependent information, such as the position of the object, benefits classifiers by giving additional information to differentiate between different objects.

To be able to rapidly characterise interesting transients, a recurrent neural network works better than a random forest to provide early classification when only a few observations of a new object are available. Early classification is useful, for example, to identify supernovae before peak brightness to allow for follow-up observations for the study the early-time behaviour of such transients.

6.2 Applications of a recurrent neural network to other problems

6.2.1 Classifying Zwicky Transient Facility alerts

During an average night of observation with clear conditions, ZTF carries out approximately 700 science exposures, yielding about 1 TB of uncompressed data. The number of 5σ alerts (where the signal-to-noise ratio of a candidate object is $\mathrm{S/N} > 5$) that ZTF generates every night varies from 10^5 to 3×10^6 (Mahabal et al., 2019). The ZTF alerts are made available to the public as 'alert packets', that contain image data of the object,

metadata specific to science, nearest cross-matches to objects in other catalogs, and the history of the object for the past 30 days.

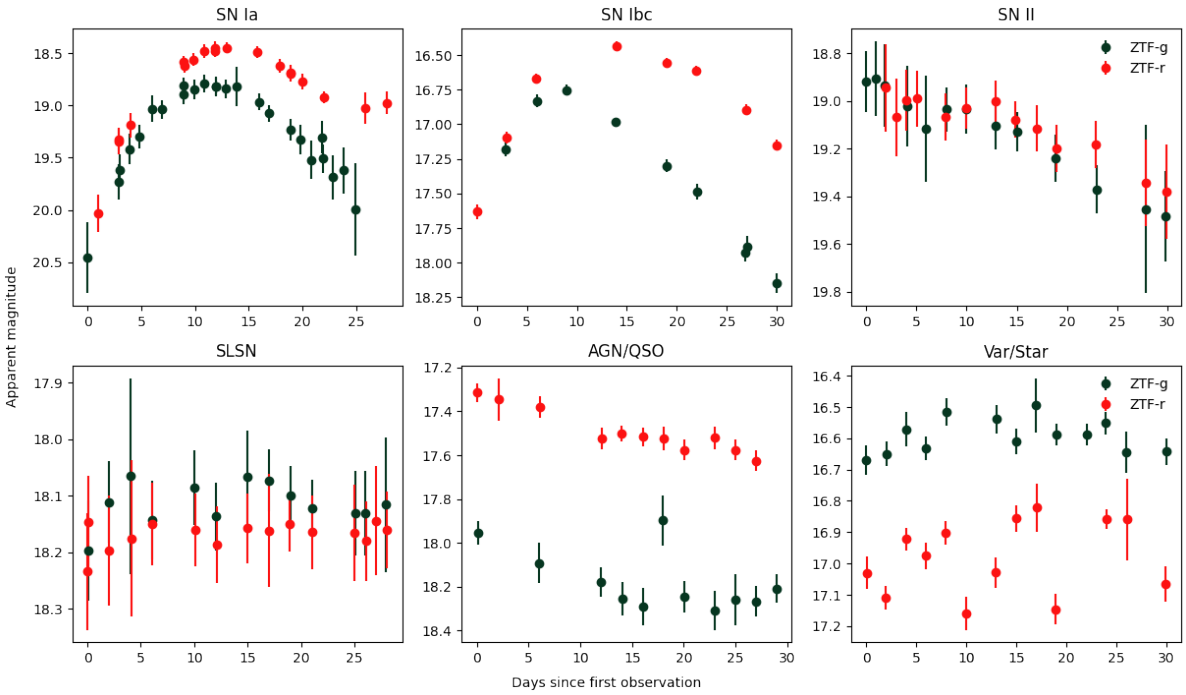


Figure 6.1: Example ZTF light curves. The light curves are scaled in time so that the time of the first observation is zero.

We use a dataset of ZTF public alerts, with labels obtained by cross-matching with the SIMBAD catalog and identifying alerts with available classifications on the Transient Name Server (TNS). This dataset was used by Leoni et al. (2021) to train a classifier for early Type Ia supernova identification with active learning, and made available through a Zenodo online data repository[1]. Photometric information for the alerts in ZTF g and r for this dataset was obtained through the FINK broker (Möller et al., 2020) interface. Each alert contained a unique alert ID, a ZTF object ID, observation history of up to 30 days (given in g and r magnitudes, with corresponding errors), the R.A. and Dec, and classifications obtained via cross-matching. We divide the alerts into six classes: SN Ia, SN Ibc, SN II, SLSN, AGN/QSO, and Var/Star. The AGN/QSO and Var/Star

[1]https://zenodo.org/record/5645609#.YhipM1SnxUQ

Class	**Training**	**Validation**	**Testing**
Var/Star	68,130	11,991	19,968
AGN/QSO	15,926	2,791	4,785
SN Ia	2,849	517	814
SN II	1,240	250	355
SN Ibc	245	38	75
SLSN	149	38	44

Table 6.1: The number of samples in each class for the ZTF alerts dataset.

classes were made following the SIMBAD classification schema, and contain objects that are not supernovae. The supernovae classes were made following classifications provided from TNS. Example light curves for each class are shown in Figure 6.1. Since each alert contains the observation history of an object for up to 30 days, for objects that vary on longer timescales than transients such as Var/Star and AGN/QSO there may be multiple alerts for a single object. There were over 4 million alerts that were cross-matched to SIMBAD, so we randomly remove duplicate alerts for each object so that each Var/Star and AGN/QSO object had one alert. We only use alerts that had at least 3 observations in both g and r bands. The final dataset consisted of 132,205 alerts. We divide the alerts into 80% for training and 20% for testing, and of the training set we use 15% for validation. Table 6.1 shows the class distribution of the ZTF alerts dataset, and it shows that the dataset is imbalanced, with more Var/Star alerts compared to some supernova classes (e.g. Type Ibc and superluminous supernovae).

We perform minimal preprocessing to the alerts, binning the light curve in both bands into nightly bins, so observations made in one night are grouped together. Then a matrix is constructed from the photometry information contained in each alert in the following format:

$$X_T = \begin{bmatrix} t_0 & ... & ... & ... & t_N \\ g_0 & ... & ... & ... & g_N \\ r_0 & ... & ... & ... & r_N \end{bmatrix} \tag{6.1}$$

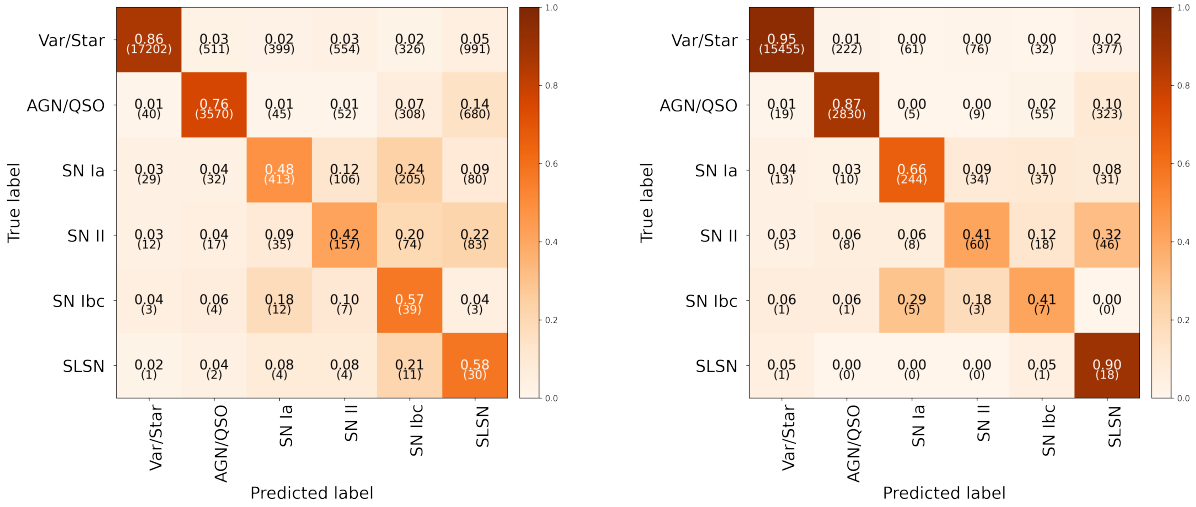
where $t_0, ..., t_N$ are the times of observations up to N days after the first observation scaled so that $t_0 = 0$, g_t is the g-band magnitude at t days and r_t is the r-band magnitudes at t days. If there is only an observation in one band at the same time-step, the missing magnitude value is filled in using the mean magnitude of all other available magnitudes in that band. We also include additional contextual information: the Galactic longitude and latitude of the object which have been scaled so that they range from 0 to 1 for latitude, and -1 to 1 for longitude. We also cross-match each alert to the GLADE catalogue (Dálya et al., 2018) to find the on-sky separation to the nearest galaxy in arcminutes. With this preprocessing method, approximately 50 alerts can be preprocessed per second with an 8-core CPU. The combination of the light curve and contextual information are used as inputs for the classifier.

We train a GRU recurrent neural network with focal loss, using the hyperparameters listed in Table 6.2 for 500 epochs. Training was carried out on a NVIDIA Quadro P2200 graphics processing unit with 1280 cores and 5GB of memory, and took approximately 130 seconds per epoch of training. After training, the classifier was evaluated on the test set, and we show the confusion matrix in Figure 6.2a.

Hyperparameter	**Value**
Batch size	128
Learning rate	1×10^{-4}
LSTM/GRU output dimension	100
Dropout	0.4
Number of neurons in dense layers	128
Regularization factor λ	0.01
Focal loss γ	1.0

Table 6.2: Adjustable hyperparameters in the GRU RNN.

From the confusion matrix, we can see that there is good classification for Var/Star and AGN/QSO but the performance for the supernova classes is sub-par, with confusion occurring between supernova types. Some SLSNe are misclassified as AGN/QSO. However, the classifier is able to distinguish most supernovae from the non-supernovae



(a) Confusion matrix on the ZTF alerts test set.

(b) Confusion matrix on the ZTF alerts test set, at a threshold of 0.6

Figure 6.2: Confusion matrices for the trained classifier evaluated on the test set.

Class	No threshold cut		Threshold = 0.6	
	Precision	Recall	Precision	Recall
Var/Star	99.5%	86.1%	99.7%	95.3%
AGN/QSO	86.3%	76.0%	92.2%	87.3%
SN Ia	45.5%	47.7%	75.5%	66.1%
SN II	17.8%	41.5%	33%	41.4%
SN Ibc	4%	57.4%	4.7%	41.2%
SLSN	1.6%	57.7%	2.3%	90%

Table 6.3: Precision and recall for each class.

objects. These results could benefit from additional exploration of hyperparameter space and attempting alternative methods of light curve preprocessing for a recurrent neural network (e.g. using interpolation methods to fill in missing obervations). We can reduce the number of false positives in each class by employing a classification probability threshold as in Chapter 4. We show the confusion matrix when a threshold of 0.6 is used in Figure 6.2b. In Table 6.3, we show the precision and recall for each class when no threshold is used and when a threshold of 0.6 is used. The trained classifier is able to classify the entire test set (26,401 alerts) in 5 seconds, and the preprocessing step can

process ~ 50 light curves per second.

6.2.2 Categorizing heartbeats

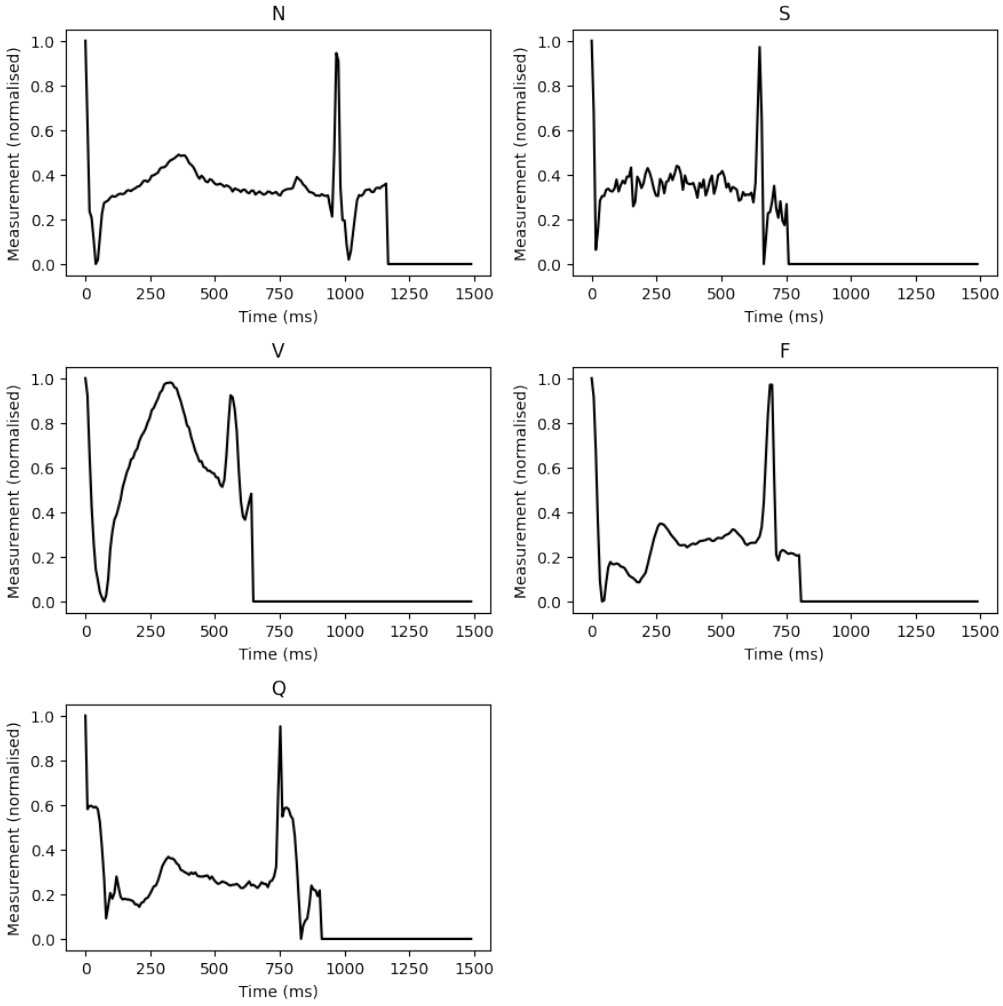


Figure 6.3: Example ECGs for the five categories outlined in Kachuee et al. (2018)

Electrocardiograms (ECG) can be used to monitor the functionality of human cardiovascular systems, and are widely used in medicine to monitor cardiac health. The challenge in manually analysing ECG measurements is identifying and categorizing the different features in the signals. Here, we apply the recurrent neural network presented in Chapter 4 in an attempt to categorise a dataset of ECG heartbeat measurements. We

Label	Description	Number
N	• Normal • Left/right bundle branch block • Atrial escape • Nodal escape	90,589
S	• Atrial premature • Aberrant atrial premature • Nodal premature • Supra-ventricular premature	2,779
V	• Premature ventricular contraction • Ventricular escape	7,236
F	• Fusion of ventricular and normal	803
Q	• Paced • Fusion of paced and normal • Unclassifiable	8,039

Table 6.4: A summary of the five categories of heartbeats with the number of measurements for each category, from Kachuee et al. (2018)

use the MIT-BIH Arrhythmia Database (Moody & Mark, 2001), which has been preprocessed and categorised into five different types of heartbeats (Kachuee et al., 2018).

The dataset contains 109,446 ECG measurements of single heartbeats in a 1.4s time windows, sampled with a frequency of 125Hz, and normalised so the signal values range from zero to one. Each sequence had a length of 187 elements. The different categories of heartbeats are tabulated in Table 6.4, along with the number of measurements for each type. For a more detailed discussion on the dataset, preprocessing, and categorisation, see Kachuee et al. (2018). Examples of ECG measurements for each category are shown in Figure 6.3.

The dataset was divided into 87,554 samples for training, and 21892 for testing, with similar class proportions for both training and test. Out of the training set, 10% was used for validation. From Table 6.4, it can be seen that the dataset is imbalanced, making it well suited for a test case with the focal loss approach to dealing with imbalance in recurrent neural networks. We trained a single GRU recurrent neural network with focal loss, with a GRU output dimension of 100, 128 neurons in the dense layers, a focal loss

gamma value of 1.0 for 100 epochs at a learning rate of 1×10^{-4} using a batch size of 256. We show the confusion matrix of the classifier evaluated on the test set in Figure 6.4.

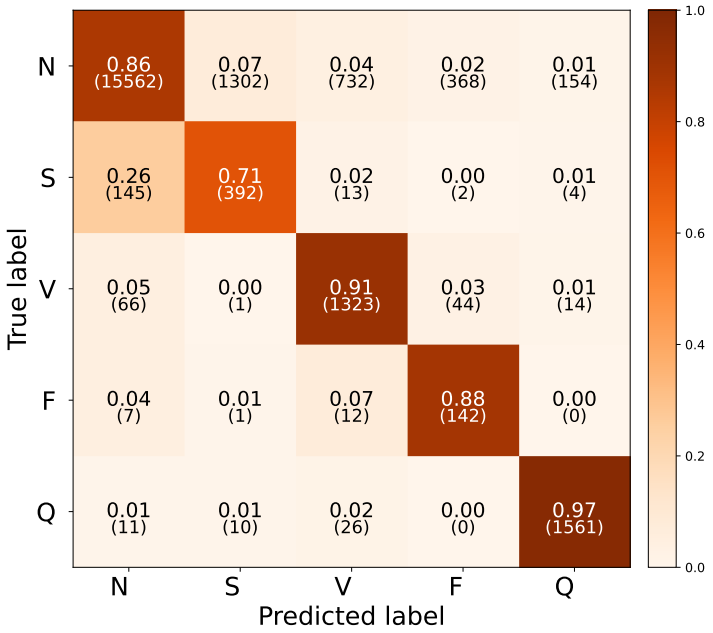


Figure 6.4: ECG classification confusion matrix, evaluated on the test set.

From the confusion matrix, we see that the classifier is able to achieve good classification across all classes. There is some degree of confusion between the 'N' and 'S' type heartbeats. The classifier is able to achieve good classifications for the classes where there were relatively fewer samples ('Q' and 'F'), highlighting the usefulness of the focal loss when dealing with imbalanced data. The results shown here were obtained with minimal hyperparameter exploration. Further work could include optimising over a grid of hyperparameters, or incorporating additional information into the classifier (e.g. additional patient information) as 'contextual information' as done in Chapter 4.

6.3 Classifying data from multiple surveys

In Chapter 5, we presented a method to classify supernova light curves from multiple surveys. Supernova light curves from different surveys were obtained from the Open Supernova Catalog (Guillochon et al., 2017) to create a heterogeneous dataset (i.e. one where the samples in the dataset may not necessarily come from the same distribution). We used a two-dimensional Gaussian process to generate standardised flux heatmaps (using a similar approach used by Qu et al. (2021)) from the supernova light curves. Data with a grid-like topology are well suited for convolutional neural networks, which we used to classify the supernova flux heatmaps. Due to the limited sample size, the classifier struggles to classify Type Ib/c supernova, but is able to classify Type Ia and Type II supernovae with good accurcacy.

We also apply transfer learning, by taking a classifier trained on Open Supernova Catalog supernova light curves and applying it to a subset of supernova light curves from the PLAsTiCC dataset (The PLAsTiCC team et al., 2018). The motivation behind this is to 'borrow' expertise from one domain and apply it to a different domain. We generated flux heatmaps from PLAsTiCC supernova light curves with the same two-dimensional Gaussian process used for the Open Supernova Catalog supernova light curves, and compare convolutional neural networks trained with and without transfer learning. The use of transfer learning showed improvement in classification for a number of classes in the PLAsTiCC dataset, and including photometric redshift information provided a small improvement to overall classification.

Large ground-based time-domain surveys such as the Vera Rubin Observatory Legacy Survey of Space and Time will be able to discover millions of transients during its campaign, but with an expected cadence of ~ 3 days in its main survey strategy (Ivezić et al., 2019), it will be useful to utilise additional facilities to provide auxiliary observations (for increased time and also wavelength coverage). A classifier that can aggregate data

from different facilities to provide classifications will be useful in this regard. Transfer learning can be useful in developing classifiers for new surveys, where there is a lack of labelled data to create a training set. A classifier trained on data from other surveys can be used to develop an initial classifier for a new survey, which can be refined as the new survey progresses.

6.4 Limitations

The main work presented in this thesis is focused on the task of *supervised classification* for optical time-domain ground-based surveys. In this context, machine learning and deep learning is used to learn a mapping from a set of inputs (e.g. light curves and contextual information) to a set of outputs (pre-defined classes of objects). The expected outputs are known beforehand, which are the different classes of time-varying astronomical objects. These are usually well-understood, with theoretical understanding and models built on decades of observational data to provide a physical description of such objects, and how they differ from one another. This highlights the first limitation of the work presented in this thesis: we provide a means of classifying new objects into already established classes, but not how to identify new kinds of transients. The second limitation is that we provide a 'static' description of machine learning and deep learning in time-domain surveys. In reality, time-domain surveys will continually make new discoveries, and classifiers will need to be re-trained to 'keep up' with the incoming data stream.

The task of identifying new and unknown objects that do not fall into any classes of known objects falls under anomaly detection. Given a set of known objects with a known distribution, a new object may be discovered that lies outside the known distribution, which can be considered anomalous. Anomaly detection has been applied to identify anomalous objects in the Open Supernova Catalog using an isolation forest (Pruzhin-

skaya et al., 2019), identifying non-supernovae events or instances of rare supernova classes. On average, an anomalous object will traverse fewer splits to reach a node in a tree compared to common objects, and this distance can be used as an anomaly score. A real-time approach to anomaly detection has been developed by Muthukrishna et al. (2021) applied to ZTF survey data. A model is used to predict the future photometric evolution of a transient given its past observations, and anomaly score is calculated using the discrepancy from the prediction to the observation. This approach was able to identify rare classes of transients such as kilonovae and superluminous supernovae with respect to common supernovae classes.

An outstanding challenge of developing classifiers for new surveys is the scarcity of labelled objects to create a training set. Active learning is a field of machine learning that tackles the problem of maximising the performance of a classifier by constructing a training set with a minimum number of labels. This is done by iteratively identifying samples in the test set that would give the best improvement in a classifier if it were included as part of a labelled training set. Ishida et al. (2018) used an active learning approach to select the best supernova candidates for spectroscopic follow-up, to create an optimised labelled training set for supernova photometric classification.

6.5 Future work

We have shown how the recurrent neural network classifier can be used to classify ZTF alerts. This can be incorporated into a transient broker such as FINK (Möller et al., 2020). Transient brokers such as FINK, and others (LASAIR; Smith et al. (2019), ALeRCE; Förster et al. (2020)) ingest data from large surveys such as ZTF, and distribute them to the scientific community. This is done in preparation for LSST, which will generate an order of magnitude more data per night than ZTF, and to trial the data processing and distribution pipelines. Our classifier can be used to provide initial clas-

sifications for ZTF alerts using their light curves, and be used to provide classification reports as part of the broker 'eco-system'.

In Chapter 4, we generated a high-dimensional output of the recurrent neural network that carries some feature representation of the data. This data representation could be fed into an anomaly detection algorithm, to identify anomalous objects. Both supervised and unsupervised classifications are useful in astronomical surveys: supervised classification provides utility and automation by classifying new objects into known classes, and unsupervised classification acts as a facilitator for specific science goals that utilise observations of rare and novel objects or even discovery of new transients.

In this thesis, we have used a random forest, a recurrent neural network, and a convolutional neural network, which have been widely used in astronomy with much success. Although they achieve good results, however, there may be algorithms or network architectures that are better suited for time-series photometric classification. Temporal convolution networks (TCNs) have been shown as a viable alternative to recurrent neural networks for processing transient light curves, and are faster to train (Muthukrishna et al., 2021).

Observing astronomical objects is not only limited to optical wavelengths; space-based observatories such as the James Webb Space Telescope (Gardner et al., 2006), eROSITA (Predehl et al., 2021) and radio array observatories such as Square Kilometer Array (Dewdney et al., 2009) are able to observe at wavelengths beyond the optical, from X-ray to radio - the techniques presented in this thesis could be applied for classification of transients visible in other wavelengths. Machine learning has been applied to classify radio transient and variable sources using features extracted from radio light curves with a random forest (Sooknunan et al., 2018). We have shown how machine learning and deep learning can be used to provide classifications in the discovery process of time-domain surveys operating in optical wavelengths. This is just a means to an end; the task of automated classification provides utility and enables further scientific study of

individual objects in a new 'Big Data' era of astronomy.

Bibliography

Abadi, M., Barham, P., Chen, J. et al 2016, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) Tensorflow: A system for large-scale machine learning. pp 265–283

Abbott, B.P., Abbott, R., Abbott, T.D. et al 2016, Physical Review Letters, 116, 061102

Abbott, B.P., Abbott, R., Abbott, T.D. et al 2017, ApJL, 848, L12

Abbott, B.P., Abbott, R., Abbott, T.D. et al 2017, , 119, 161101

Abbott, B.P., Abbott, R., Adhikari, R. et al 2009, Reports on Progress in Physics, 72, 076901

Acernese, F., Agathos, M., Agatsuma, K. et al 2015, Classical and Quantum Gravity, 32, 024001

Aihara, H., Arimoto, N., Armstrong, R. et al 2018, Publications of the Astronomical Society of Japan, 70, S4

Aizerman, M.A., Braverman, E.A. and Rozonoer, L. 1964, in Automation and Remote Control, No. 25 in Automation and Remote Control,, Theoretical foundations of the potential function method in pattern recognition learning.. pp 821–837

Baldwin, J.A., Phillips, M.M. and Terlevich, R. 1981, PASP, 93, 5

Ball, N.M. and Brunner, R.J. 2010, International Journal of Modern Physics D, 19, 1049

Baltay, C., Rabinowitz, D., Hadjiyska, E. et al 2013, PASP, 125, 683

Baron, D. 2019, arXiv e-prints, p. arXiv:1904.07248

Bellm, E.C. 2018, arXiv e-prints, p. arXiv:1802.10218

Bellm, E.C., Kulkarni, S.R., Graham, M.J. et al 2019, PASP, 131, 018002

Betoule, M., Kessler, R., Guy, J. et al 2014, A&A, 568, A22

Bishop, C.M. 2006, Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg

Blanton, M.R. and Roweis, S. 2007, AJ, 133, 734

Bloom, J.S. and Richards, J.W. 2012, Data Mining and Machine Learning in Time-Domain Discovery and Classification. Chapman and Hall/CRC, pp 89–112

Bloom, J.S., Richards, J.W., Nugent, P.E. et al 2012, PASP, 124, 1175

Boone, K. 2019, The Astronomical Journal, 158, 257

Breeveld, A.A., Landsman, W., Holland, S.T. et al 2011, in McEnery J. E., Racusin J. L., Gehrels N., eds, Gamma Ray Bursts 2010 Vol. 1358 of American Institute of Physics Conference Series, An Updated Ultraviolet Calibration for the Swift/UVOT. pp 373–376

Breiman, L. 2001, Machine Learning, 45, 5

Breiman, L., Friedman, J., Stone, C. et al 1984, Classification and Regression Trees. Taylor & Francis

Brink, H., Richards, J.W., Poznanski, D. et al 2013, MNRAS, 435, 1047

Burhanudin, U.F., Maund, J.R., Killestein, T. et al 2021, MNRAS, 505, 4345

Cabral, J.B., Sánchez, B., Ramos, F. et al 2018, Astronomy and Computing, 25, 213

Carrasco-Davis, R., Cabrera-Vives, G., Förster, F. et al 2019, PASP, 131, 108006

Carrick, J.E., Hook, I.M., Swann, E. et al 2020, arXiv e-prints, p. arXiv:2012.12122

Charnock, T. and Moss, A. 2017, ApJL, 837, L28

Chatzimparmpas, A., Martins, R.M. and Kerren, A. 2020, arXiv e-prints, p. arXiv:2002.06910

Chawla, N.V., Bowyer, K.W., Hall, L.O. et al 2002, J. Artif. Int. Res., 16, 321–357

Cho, K., van Merrienboer, B., Bahdanau, D. et al 2014, arXiv e-prints, p. arXiv:1409.1259

Chollet, F. et al, 2015, Keras, https://keras.io

Chornock, R., Berger, E., Kasen, D. et al 2017, ApJL, 848, L19

Cortes, C. and Vapnik, V. 1995, Mach. Learn., 20, 273–297

Coulter, D.A., Foley, R.J., Kilpatrick, C.D. et al 2017, Science, 358, 1556

Dálya, G., Galgóczi, G., Dobos, L. et al 2018, MNRAS, 479, 2374

Dark Energy Survey Collaboration 2016, MNRAS, 460, 1270

Dauphin, F., Hosseinzadeh, G., Villar, V. et al 2020, in American Astronomical Society Meeting Abstracts American Astronomical Society Meeting Abstracts, Photometric Classification of Transients from the Pan-STARRS1 Medium-Deep Survey. p. 276.18

De, K., Kasliwal, M.M., Ofek, E.O. et al 2018, Science, 362, 201

Dewdney, P.E., Hall, P.J., Schilizzi, R.T. et al 2009, IEEE Proceedings, 97, 1482

Djorgovski, S.G., Drake, A.J., Mahabal, A.A. et al 2011, arXiv e-prints, p. arXiv:1102.5004

Drake, A.J., Djorgovski, S.G., Mahabal, A. et al 2009, ApJ, 696, 870

Drout, M.R., Piro, A.L., Shappee, B.J. et al 2017, Science, 358, 1570

Duev, D.A., Mahabal, A., Masci, F.J. et al 2019, MNRAS, 489, 3582

Dyer, M.J. 2020, PhD thesis, University of Sheffield, UK

Dyer, M.J., Steeghs, D., Galloway, D.K. et al 2020, arXiv e-prints, p. arXiv:2012.02685

Fawcett, T. 2006, Pattern Recognition Letters, 27, 861

Filippenko, A.V. 1997, ARAA, 35, 309

Foley, R.J., Challis, P.J., Chornock, R. et al 2013, ApJ, 767, 57

Foley, R.J., Chornock, R., Filippenko, A.V. et al 2009, The Astronomical Journal, 138, 376

Förster, F., Cabrera-Vives, G., Castillo-Navarrete, E. et al 2020, arXiv e-prints, p. arXiv:2008.03303

Friedman, J.H. 2001, The Annals of Statistics, 29, 1189

Gaia Collaboration 2018, A&A, 616, A10

Gal-Yam, A. 2012, Science, 337, 927

Gal-Yam, A. 2019, Annual Review of Astronomy and Astrophysics, 57, 305

Galama, T.J., Vreeswijk, P.M., van Paradijs, J. et al 1998, Nature, 395, 670

Gardner, J.P., Mather, J.C., Clampin, M. et al 2006, Space Science Reviews, 123, 485

Gehrz, R.D., Truran, J.W., Williams, R.E. et al 1998, PASP, 110, 3

Gieseke, F., Bloemen, S., van den Bogaard, C. et al 2017, MNRAS, 472, 3101

Goodfellow, I., Bengio, Y. and Courville, A. 2016, Deep Learning. The MIT Press

Graves, A. 2013, CoRR, abs/1308.0850

Guillochon, J., Parrent, J., Kelley, L.Z. et al 2017, ApJ, 835, 64

Hamuy, M., Phillips, M.M., Suntzeff, N.B. et al 1996, AJ, 112, 2391

Hand, D. and Till, R. 2001, Hand, The, 45, 171

Hernanz, M. 2005, in Hameury J.-M., Lasota J.-P., eds, The Astrophysics of Cataclysmic Variables and Related Objects Vol. 330 of , Classical nova explosions. p. 265

Hochreiter, S. and Schmidhuber, J. 1997, Neural Comput., 9, 1735–1780

Hosenie, Z., Lyon, R., Stappers, B. et al 2020, MNRAS, 493, 6050

Hosseinzadeh, G., Dauphin, F., Villar, V.A. et al 2020, arXiv e-prints, p. arXiv:2008.04912

Ioffe, S. and Szegedy, C., 2015, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Ishida, E.E.O., Beck, R., González-Gaitán, S. et al 2018, Monthly Notices of the Royal Astronomical Society, 483, 2

Ivezić, Ž., Kahn, S.M., Tyson, J.A. et al 2008, ArXiv e-prints, p. arXiv:0805.2366

Ivezić, Ž., Kahn, S.M., Tyson, J.A. et al 2019, ApJ, 873, 111

Johnson, J. and Khoshgoftaar, T. 2019, Journal of Big Data, 6, 27

Justesen, N., Bontrager, P., Togelius, J. et al 2017, arXiv e-prints, p. arXiv:1708.07902

Kachuee, M., Fazeli, S. and Sarrafzadeh, M. 2018, CoRR, abs/1805.00794

Kaiser, N., Burgett, W., Chambers, K. et al 2010, , 7733, 7733

Kasliwal, M.M. 2011, PhD thesis, California Institute of Technology

Kasliwal, M.M., Kulkarni, S.R., Arcavi, I. et al 2011, ApJ, 730, 134

Kasliwal, M.M., Kulkarni, S.R., Gal-Yam, A. et al 2012, The Astrophysical Journal, 755, 161

Kasliwal, M.M., Kulkarni, S.R., Gal-Yam, A. et al 2010, The Astrophysical Journal Letters, 723, L98

Kessler, R., Conley, A., Jha, S. et al 2010, arXiv e-prints, p. arXiv:1001.5210

Kessler, R., Narayan, G., Avelino, A. et al 2019, PASP, 131, 094501

Khazov, D., Yaron, O., Gal-Yam, A. et al 2016, ApJ, 818, 3

Killestein, T.L., Lyman, J., Steeghs, D. et al 2021, MNRAS, 503, 4838

Kingma, D.P. and Ba, J. 2014, arXiv e-prints, p. arXiv:1412.6980

Kochanek, C.S., Shappee, B.J., Stanek, K.Z. et al 2017, PASP, 129, 104502

Krawczyk, B. 2016, Progress in Artificial Intelligence, 5, 221

Krisciunas, K., Contreras, C., Burns, C.R. et al 2017, AJ, 154, 211

Kulkarni, S.R. 2013, The Astronomer's Telegram, 4807

Kulkarni, S.R., Ofek, E.O., Rau, A. et al 2007, Nature, 447, 458

Kullback, S. and Leibler, R.A. 1951, Ann. Math. Statist., 22, 79

Law, N.M., Kulkarni, S.R., Dekany, R.G. et al 2009, PASP, 121, 1395

Leavitt, H.S. and Pickering, E.C. 1912, Harvard College Observatory Circular, 173, 1

Lecun, Y. 1989, Generalization and network design strategies. Elsevier

LeCun, Y., Boser, B., Denker, J.S. et al 1989, Neural Comput., 1, 541–551

Lemaître, G., Nogueira, F. and Aridas, C.K. 2017, Journal of Machine Learning Research, 18, 1

Leoni, M., Ishida, E.E.O., Peloton, J. et al 2021, arXiv e-prints, p. arXiv:2111.11438

Li, W., Filippenko, A.V., Chornock, R. et al 2003, PASP, 115, 453

Lin, H.W., Chen, Y.T., Wang, J.H. et al 2018, Publications of the Astronomical Society of Japan, 70, S39

Lin, T.Y., Goyal, P., Girshick, R. et al 2017, arXiv e-prints, p. arXiv:1708.02002

Liu, J., Soria, R., Wu, X.F. et al 2021, An. Acad. Bras. Ciênc. vol.93 supl.1, 93, 20200628

Lochner, M., McEwen, J.D., Peiris, H.V. et al 2016, The Astrophysical Journal Supplement Series, 225, 31

LSST Science Collaboration, Abell, P.A., Allison, J. et al 2009, arXiv e-prints, p. arXiv:0912.0201

Mahabal, A., Rebbapragada, U., Walters, R. et al 2019, Publications of the Astronomical Society of the Pacific, 131, 038002

Maoz, D., Mannucci, F. and Nelemans, G. 2014, Annual Review of Astronomy and Astrophysics, 52, 107

Margutti, R., Metzger, B.D. and Chornock, R.e.a. 2019, ApJ, 872, 18

Maund, J.R., Crowther, P.A., Janka, H.T. et al 2017, Philosophical Transactions of the Royal Society of London Series A, 375, 20170025

Maund, J.R. and Smartt, S.J. 2005, MNRAS, 360, 288

Metzger, B.D., Martínez-Pinedo, G., Darbha, S. et al 2010, Monthly Notices of the Royal Astronomical Society, 406, 2650

Mitchell, T.M. 1997, Machine Learning. McGraw-Hill, New York

Modjaz, M., Gutiérrez, C.P. and Arcavi, I. 2019, Nature Astronomy, 3, 717

Möller, A. and de Boissière, T. 2020, MNRAS, 491, 4277

Möller, A., Peloton, J., Ishida, E.E.O. et al 2020, MNRAS

Mong, Y.L., Ackley, K., Galloway, D. et al 2020, arXiv e-prints, p. arXiv:2008.10178

Moody, G. and Mark, R. 2001, IEEE Engineering in Medicine and Biology Magazine, 20, 45–50

Muthukrishna, D., Mandel, K.S., Lochner, M. et al 2021, arXiv e-prints, p. arXiv:2111.00036

Muthukrishna, D., Narayan, G., Mandel, K.S. et al 2019, PASP, 131, 118002

Narayan, G., Zaidi, T., Soraisam, M.D. et al 2018, ApJS, 236, 9

Nun, I., Protopapas, P., Sim, B. et al 2015, arXiv e-prints, p. arXiv:1506.00010

Oke, J.B. and Gunn, J.E. 1983, ApJ, 266, 713

Pan, S.J. and Yang, Q. 2010, IEEE Transactions on Knowledge and Data Engineering, 22, 1345

Pasquet, J., Pasquet, J., Chaumont, M. et al 2019, A&A, 627, A21

Pastorello, A., Mason, E., Taubenberger, S. et al 2019, Astronomy Astrophysics, 630, A75

Pedregosa, F., Varoquaux, G., Gramfort, A. et al 2011, Journal of Machine Learning Research, 12, 2825

Perley, D.A., Fremling, C., Sollerman, J. et al 2020, The Astrophysical Journal, 904, 35

Perley, D.A., Mazzali, P.A. and Yan, L.e.a. 2019, MNRAS, 484, 1031

Perlmutter, S., Aldering, G., Goldhaber, G. et al 1999, ApJ, 517, 565

Phillips, M.M. 1993, ApJL, 413, L105

Poznanski, D., Chornock, R., Nugent, P.E. et al 2010, Science, 327, 58

Predehl, P., Andritschke, R., Arefiev, V. et al 2021, A&A, 647, A1

Pruzhinskaya, M.V., Malanchev, K.L., Kornilov, M.V. et al 2019, MNRAS, 489, 3591

Qu, H. and Sako, M. 2021, arXiv e-prints, p. arXiv:2111.05539

Qu, H., Sako, M., Möller, A. et al 2021, AJ, 162, 67

Quimby, R.M., Aldering, G., Wheeler, J.C. et al 2007, ApJ, 668, L99

Rasmussen, C.E. and Williams, C.K.I. 2005, Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press

Rau, A., Kulkarni, S.R., Law, N.M. et al 2009, PASP, 121, 1334

Revsbech, E.A., Trotta, R. and van Dyk, D.A. 2018, MNRAS, 473, 3969

Richards, J.W., Starr, D.L., Butler, N.R. et al 2011, ApJ, 733, 10

Riess, A.G., Filippenko, A.V., Challis, P. et al 1998, The Astronomical Journal, 116, 1009

Rodrigo, C. and Solano, E. 2020, in XIV.0 Scientific Meeting (virtual) of the Spanish Astronomical Society The SVO Filter Profile Service. p. 182

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. 1986, Nature, 323, 533

Scalzo, R.A., Yuan, F., Childress, M.J. et al 2017, PASA, 34, e030

Shappee, B.J., Prieto, J.L., Grupe, D. et al 2014, ApJ, 788, 48

Shappee, B.J., Simon, J.D., Drout, M.R. et al 2017, Science, 358, 1574

Shivvers, I., Filippenko, A.V., Silverman, J.M. et al 2019, MNRAS, 482, 1545

Smartt, S.J. 2009, Annual Review of Astronomy and Astrophysics, 47, 63

Smartt, S.J., Chen, T.W., Jerkstrand, A. et al 2017, Nature, 551, 75

Smith, K.W., Williams, R.D., Young, D.R. et al 2019, Research Notes of the AAS, 3, 26

Smith, M., D'Andrea, C.B., Sullivan, M. et al 2020, The Astronomical Journal, 160, 267

Sooknunan, K., Lochner, M., Bassett, B.A. et al 2018, arXiv e-prints, p. arXiv:1811.08446

Srivastava, N., Hinton, G., Krizhevsky, A. et al 2014, Journal of Machine Learning Research, 15, 1929

Starrfield, S., Iliadis, C. and Hix, W.R. 2016, PASP, 128, 051001

Steeghs, D., Galloway, D., Ackley, K. et al 2021, Monthly Notices of the Royal Astronomical Society

Sullivan, M. 2013, Astronomy & Geophysics, 54, 6.17

Takahashi, I., Suzuki, N., Yasuda, N. et al 2020, Publications of the Astronomical Society of Japan

The PLAsTiCC team, Allam, Tarek, J., Bahmanyar, A. et al 2018, arXiv e-prints, p. arXiv:1810.00001

Tonry, J.L. 2011, Publications of the Astronomical Society of the Pacific, 123, 58

Tonry, J.L., Denneau, L., Heinze, A.N. et al 2018, Publications of the Astronomical Society of the Pacific, 130, 064505

Trouille, L., Barger, A.J. and Tremonti, C. 2011, arXiv e-prints, p. arXiv:1110.0008

Valenti, S., Benetti, S., Cappellaro, E. et al 2008, Monthly Notices of the Royal Astronomical Society, 383, 1485

Valenti, S., David, J.S., Yang, S. et al 2017, The Astrophysical Journal Letters, 848, L24

van der Maaten, L. and Hinton, G. 2008, Journal of Machine Learning Research, 9, 2579

Véron-Cetty, M.P. and Véron, P. 2010, A&A, 518, A10

Villar, V.A., Berger, E., Miller, G. et al 2019, ApJ, 884, 83

Villar, V.A., Guillochon, J., Berger, E. et al 2017, ApJL, 851, L21

Watson, C.L., Henden, A.A. and Price, A. 2006, Society for Astronomical Sciences Annual Symposium, 25, 47

Wenger, M., Ochsenbein, F., Egret, D. et al 2000, A&A Suppl., 143, 9

Werbos, P.J. 1990, Proceedings of the IEEE, 78, 1550

Wright, D.E., Smartt, S.J., Smith, K.W. et al 2015, MNRAS, 449, 451

Zhang, J., Wang, X., Vinkó, J. et al 2018, ApJ, 863, 109

Appendix A

Filters and magnitude conversions Open Supernova Catalog light curves

Filter	λ_{eff}	**FWHM**
SDSS u′	3542.19	614.25
SDSS g′	4724.6	1434.93
SDSS r′	6202.58	1392.09
SDSS i′	7673.03	1564.83
SDSS z′	10506.54	5117.17
Pan-STARRS g	4810.88	1256.27
Pan-STARRS r	6156.36	1404.4
Pan-STARRS w	5985.87	3625.0
Pan-STARRS open	6439.35	5020.68
Pan-STARRS i	7503.68	1296.71
Pan-STARRS z	8668.56	1034.32
Pan-STARRS y	9613.45	628.19
Swift UVW2	2085.73	584.89
Swift UVM2	2245.78	527.13
Swift UVW1	2684.14	656.6
Swift U	3520.95	778.49
Swift white	3885.68	3866.59
Swift B	4346.25	978.33
Swift V	5411.43	744.99
EFOSC U	3572.7	530.98
EFOSC B	4345.36	1011.22
EFOSC V	5457.99	1133.39
EFOSC R	6416.14	1667.47
ZTF g	4722.74	1320.55
ZTF r	6339.61	1557.05
ZTF i	7886.13	1522.14

SITe3 u	3670.92	404.4
SITe3 B	4394.17	964.87
SITe3 g	4734.32	1487.5
SITe3 V	5358.16	864.15
SITe3 r	6185.57	1321.94
SITe3 i	7585.98	1366.52
CFH12K B	4311.67	990.37
CFH12K V	5338.0	973.98
CFH12K R	6516.05	1250.9
CFH12K I	8090.53	2163.85
CFHT U	3890.07	607.39
CFHT G	4781.92	1447.11
CFHT R	6515.87	1250.9
CFHT I	8090.45	2163.85
CFHT Z	8777.99	925.28
DECam u	3857.11	256.06
DECam g	4770.83	1299.43
DECam r	6371.33	1484.35
DECam i	7774.19	1481.73
DECam z	9157.9	1471.42
DECam Y	9886.35	660.79
ATLAS cyan	5183.87	2291.8
ATLAS orange	6632.15	2577.94
CRTS clear	5618.41	5668.12
Gaia G_{bp}	5017.28	2520.03
Gaia G	5845.67	4372.44
Gaia G_{rp}	7593.4	2794.54
MEGACam u	3751.88	516.26
MEGACam g	4722.38	1537.58
MEGACam r	6336.47	1478.52
MEGACam i	7641.47	1526.57
MEGACam z	8972.59	730.84

Table A.1: The effective wavelength λ_{eff} and full width half-maximum (FWHM) of filters used in the Open Supernova Catalog dataset, given in angstroms.

Filter	**AB - Vega**
V	-0.01
B	-0.13
U	+1.02
UVW1	+1.51
UVM2	+1.69
UVW2	+1.73

Table A.2: Conversion table for Swift magnitudes given in the Vega system into AB magnitudes, obtained from Breeveld et al. (2011)

Filter	**AB - Vega**
U	0.79
B	-0.09
V	0.02
R	0.21
I	0.45
u	0.91
g	-0.08
r	0.16
i	0.37
z	0.54
J	0.91
H	1.39
K_s	1.85

Table A.3: Conversion table for Vega magnitudes into AB magnitudes, obtained from Blanton & Roweis (2007)

Filter	**AB - CSP**
u	-0.06
g	-0.02
r	-0.01
i	0.00
B	-0.013
V	-0.02
Y_{RC}	0.63
J	0.90
H_{RC}	1.34
Y_{WIRC}	0.64
J_{WIRC}	0.90
H_{WIRC}	1.34

Table A.4: Conversion table for magnitudes given in the Carnegie Supernova Project (CSP) system into AB magnitudes, obtained from Krisciunas et al. (2017)

Appendix B

List of Publications

This is a list of publications from the work I have done during my PhD.

- **Burhanudin, U. F.**; Maund, J. R.;Killestein, T.; Ackley, K.; Dyer, M. J.; Lyman, J.; Ulaczyk, K.; Cutter, R.; Mong, Y. -L.; Steeghs, D.; Galloway, D. K.; Dhillon, V.; O'Brien, P.; Ramsay, G.; Noysena, K.; Kotak, R.; Breton, R. P.; Nuttall, L.; Pallé, E.; Pollacco, D.; Thrane, E.; Awiphan, S.; Chote, P.; Chrimes, A.; Daw, E.; Duffy, C.; Eyles-Ferris, R.; Gompertz, B.; Heikkilä, T.; Irawati, P.; Kennedy, M. R.; Levan, A.; Littlefair, S.; Makrygianni, L.; Mata-Sánchez, D.; Mattila, S.; McCormac, J.; Mkrtichian, D.; Mullaney, J.;Sawangwit, U.; Stanway, E.; Starling, R.; Strøm, P.; Tooke, S.; Wiersema, K., 2021, MNRAS, 5, 4345, "Light-curve classification with recurrent neural networks for GOTO: dealing with imbalanced data"[1].

[1]Presented in Chapter 4

y e b o i i i i