THE UNIVERSITY *of* EDINBURGH

Supernova Light Curve Classification using Attention and other Techniques

Amanda Ibsen

Doctor of Philosophy
The University of Edinburgh
November 2022

Abstract

Even in the era of deep-learning the huge amount of astronomical data available has not yet allowed us to solve the problem of Supernova (SN) light curve classification.

These explosive transients are photometrically difficult to label not only because of their irregular sampling and noise, but also because of the dissimilarities that exist between different surveys and datasets. This is one of the major issues in the field, as it is often the case that models work well with simulations, but not necessarily with real data. This work explores Attention mechanisms and other Deep Learning techniques with the aim of extracting meaningful feature representations from SNe in order to be able to classify their light curves.

The main contributions of this thesis are: (1) I compare several Neural Network architectures commonly used for Time Series and light curve classification to establish some baselines, (2) I propose a model that uses simple additive Self-Attention that improves early classification of light curves and (3) I present a Transformer-based architecture adapted for light curve reconstruction and classification and show how, by framing it as a probabilistic Variational Auto-Encoder, the combination of both a regular latent space and a Multi-headed Attention mechanism can help mitigate the difficulties of dealing with different datasets.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Parts of this work have been published in Ibsen & Mann (2020).

(*Amanda Ibsen, November 2022*)

Acknowledgements

I am an incredibly lucky person. I am also incredibly prone to forget it. It is really a testament to my fortune that despite all my efforts to isolate myself, I am somehow surrounded by the most wonderful people.

To my father I owe an eternal debt, for he gifted me the most incredible thing a parent can give and my most valuable possession: my education. I would not be writing this without him and this is by no means a cliché exaggeration.

I am so very grateful for the staggering amount of support I keep receiving from my fantastic friends: Dr. Ivana Bachmann, who is the aggressive yet loving voice of reason and kept me fed; Feña Muñoz, who is both one of the funniest and most competent women I have ever met; Feña Rubio, who I could always count on to go dancing with; Jimbo, who came to pick me up at 3 a.m. under curfew, which I will never ever forget; Nesli, the kindest human I know; Janie, who will face-time with me for hours; Feño, who is as disastrously effective as I am; Raph, with whom I explored the highlands; Fiona, the most pleasant and supportive flatmate; and so many others.

I am also so glad that I found Patricia, as I got here in no small measure thanks to her ultimatums. She challenges me and genuinely helps me think, even if I often hate it.

I would also like to thank my supervisors, Bob Mann and Andy Lawrence, who I feel have been incredibly patient with me.

Although they cannot read or talk or even regulate their body temperature, I would still like to thank Boopa, Lili and Pupo, if not for them, for myself. They are responsible for at least 80% of my smiles.

And finally, I am grateful I met those who left me. I miss you, formidably.

Para Morla, que me salvó.

Contents

Abstract i

Declaration iii

Acknowledgements v

Contents vii

1 Introduction 1

1.1 Astronomical Background and Motivation 6

1.1.1 Transient Astronomy and its current state..... 6

1.1.2 Supernovae: what we know and what we do not know..... 7

1.1.2.1 Supernovae Type Ia..... 10

1.1.2.2 Core collapse supernovae..... 12

1.1.2.3 Super luminous supernovae: a special case 14

1.1.3 Usefulness of Supernovae in cosmology 17

1.1.4 Hopes for the future..... 19

1.2 Relevant machine learning background..... 21

1.2.1 Artificial Intelligence, Machine Learning and Deep Learning..... 22

1.2.2 Machine learning as an optimization problem 24

1.2.3 Artificial Neural Networks 25

1.2.3.1 Neurons, connections, weights and activation functions ... 27

1.2.3.2 Backpropagation and Gradient Descent 28

1.3 Light curve classification 32

1.3.1 Time series classification 33

1.3.1.1 Dynamic Time Warping, Shapelets, Dictionaries, Forests and Ensembles 33

1.3.1.2 Recurrent Neural Networks, Convolutional Kernels and Convolutional Neural Networks 36

1.3.2 Brief survey of light curve classification of explosive transients ... 43

1.4 Outline of the thesis 48

2 Experimental framework 51

2.1 Motivation 51

2.2 Description 51

2.3 Running an experiment 53

2.4 Experimental set-up 54

2.5 Performance metrics used 55

2.6 Tools used 57

3 PLAsTiCC Challenge 59

3.1 Introduction 59

3.2 PLAsTiCC Data Set Characteristics 60

3.3 Attempt at the PLAsTiCC challenge 68
3.3.1 Pre-processing 69
3.3.1.1 DMDT construction 70
3.3.1.2 DMDT binning strategies and resolution 70
3.3.1.3 Data augmentation 74
3.3.2 Tuning the model 77
3.3.2.1 Number of convolutional layers and kernel size 77
3.3.2.2 Number of filters 79
3.3.2.3 Input of the linear layer 79
3.3.2.4 Dropout layers 80
3.3.3 Tunning hyper-parameters 81
3.3.3.1 Learning rate and weight decay coefficient 81
3.3.3.2 Batch Size 82
3.3.4 Classification of the test set 83
3.3.4.1 Class 99 83
3.4 Follow-up experiments: Using the test set for training 86
3.5 Overview of the winning solutions 89
3.5.1 First Place 89
3.5.2 Second Place 90
3.5.3 Third Place 91
3.6 Conclusions 92

4 Paying attention to the flare 95

4.1 Introduction 95

4.2 Attention and Self-Attention 97

4.3 Baseline Architectures 98

4.3.1 Fully Convolutional Neural Network (FCN) 98

4.3.2 Residual Neural Network (ResNet) 99

4.3.3 Recurrent Neural Network with Gated Recurrent Units (RNN GRU) 100

4.4 Self-Attentive Architecture 102

4.5 Data representation 103

4.6 Self-Attention applied to the PLAsTiCC dataset 105

4.6.1 Using the original training set 105

4.6.2 Using part of the test set as training set 107

4.6.3 How does balance affect the test results? 109

4.6.4 Earliness in classification 110

4.7 Self-Attention for ZTF data 113

4.7.1 Datasets 114

4.7.1.1 The real data 114

4.7.1.2 Light curves simulated using Simsurvey 116

4.7.2 Experiments 121

4.7.3 Experiments centered on earliness of classification 123

4.8 Discussion 127

5 Looking further into Attention 131

5.1 The Transformer 132

5.1.1 Transformers in Time Series 135

5.2 Transformers for light-curve analysis 136

5.3 Experiments 139

5.3.1 Initial Experiments: Establishing a baseline 139

5.3.1.1 Data Representations 140

5.3.1.2 Embeddings 141

5.3.1.3 Positional Encoding 141

5.3.1.4 Encoded representation 142

5.3.1.5 Initial Results 143

5.3.2 Autoencoder using Attention: Adding a Transformer Decoder 147

5.3.3 Variational Auto-Encoder using Attention: a probabilistic approach to the Transformer 151

5.3.3.1 Classification using VAEs 155

5.3.3.2 Classification using VAEs and semi-supervised learning 158

5.3.4 Discussion 160

6 Conclusion 165

Bibliography 169

Chapter 1

Introduction

Although it might not always seem that way, we are indeed very lucky to be alive and bear witness to the shifts in paradigms the information age has brought upon us. From the development of transistor technology, to the popularization of the internet, we, as human kind, have come a long way. Nowadays we rely on advanced computational technology even to do basic daily tasks, and do not give much thought to this fact: it is a given.

Machine learning requires an amount of technology that our ancestors would have deemed witchcraft. Even in the 80's, when back-propagation was rediscovered and the term 'machine learning' became popular (Rumelhart, 1986), the algorithms necessary to make the field a reality were far too expensive in terms of computational resources.

This started to change when by 1987, when IBM introduced a graphic card that contained its own processor and thus did not need to draw resources from the CPU (Central Processing Unit) (WayBackMachine, 2011). In the decade of the 2000's these Graphical Processing Units (GPU) became more sophisticated (Witheiler, 2001; AMD, 2008) and ever since the 2010's GPUs have been used for large scale models in machine learning (Raina et al., 2009) because of their parallelizing capabilities. This hugely revolutionized the applicability of machine learning to the point we see today: there are GPUs specially desinged to accelerate AI (Artificial Intelligence) (NVIDIA, 2021) and even GPUs themselves use machine learning to improve their rendering (Jones, 2021). Most impressively, there exists an artificial brain capable of passing, for a significant number of humans, the Turing test (Brown et al., 2020). Fantasies from classic science fiction have

become a reality.

We are also lucky enough that today data storage is cheap and hugely abundant, compared to how it used to be a few decades back (the first personal computer introduced by Apple had only a single 400 KB floppy disk drive), which allows us to gather huge amounts of data to train our machine learning algorithms with. Leaving aside all the ethical implications and discussions that arise from this (although we should really think about them as a scientific community), the beauty of the advancements in data science lies in that we no longer need to rely solely on theory, a few data points and simulations to do science. We now have the computational resources to store and process large quantities of data, and to do science *from* it. The possibility of constructing data-driven science is a huge shift in paradigm, the implications of which we have yet to witness.

We can speculate about it a little, though. When we say 'data science', what we really mean is statistics, probability and optimization. Through statistics and probability we may estimate that a thing will likely happen, but that will not necessarily shed meaning on why that is the case. For example, a study surveying over 1.6 million students, showed that girls had better grades on average than boys in every field, at every age (O'Dea, 2018). One could use that to affirm that girls are smarter, or one could instead conclude that girls feel socially pressured to do better and so put in more effort into their academic work. One could draw any number of conclusions. Thus, we need to be especially careful with the data-driven science we do nowadays, and conscious of our personal bias when doing it. It would be unwise and even irresponsible, now more than ever, to think that science is somehow an objective truth and that our point of view does not affect how we interpret the data we see. In other words, the scientific paradigm of the 'view from above' must be consciously replaced with that of a 'view from below', where the scientists insert themselves in their reality, acknowledge the point from where they are observing, and take responsibility for their research (Mies, 1993). And perhaps, it is equally important to recognize the bias a machine may have, depending on the data it has come across.

The availability of both data and hardware to process it has made it possible for artificial intelligence and machine learning specifically to be used in a range of different fields, and astronomy is no exception. The amount of astronomical data produced has greatly increased in the past few years and will continue to do so, with the arrival of a new generation of Telescopes and surveys, such as The Rubin LSST (The Legacy Survey of Space and Time or

formerly Large Synoptic Survey Telescope) (Ivezić et al., 2019a), TMT (Thirty Meter Telescope Project) (Crampton et al., 2009), Euclid, 4MOST (4-metre Multi-Object Spectroscopic Telescope) (Guiglion et al., 2019) and SKA(Square Kilometre Array) (Huynh & Lazio, 2013). This brings forth new interesting challenges related to how to deal with so much data, how to prepare for it, how to classify it, what to do with it and how to interpret it. Indeed, there has been a dramatic increase in the number of astronomy related studies that involve machine learning in the last decade (See Figure 1.1). The question at hand, and one that many astronomers are trying to answer is: How will data-driven science and our pristine technology affect our understanding of the universe? We do not fully know yet, but we will hopefully see, in time.

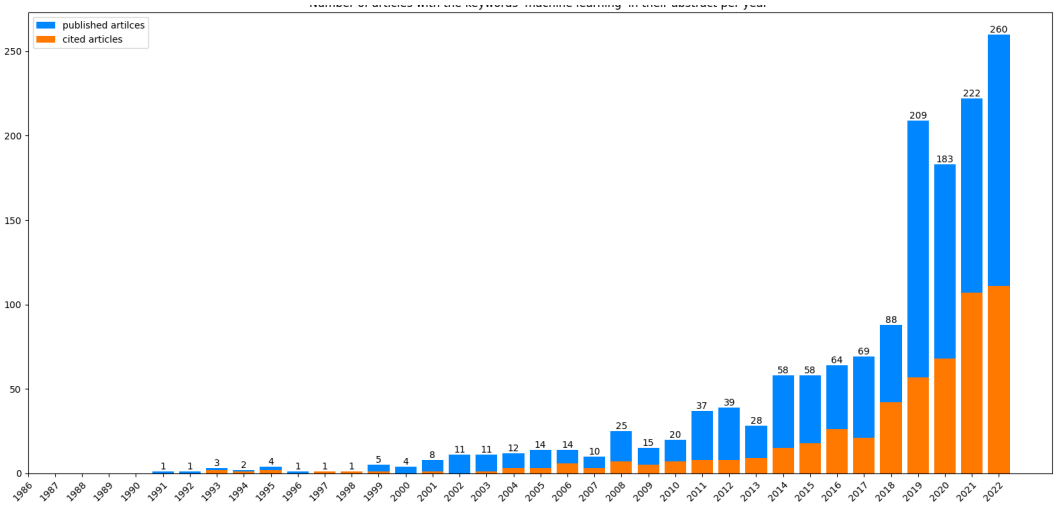


Figure 1.1 Number of astronomy related articles that have 'machine learning' as keywords in their abstract. The blue bar represents the number of published articles, while the orange bar represents the number of cited articles. The data was obtained from the Astrophysics Data System (ADS) website[1].

In this thesis though, I will narrow down the problem enormously and focus specifically on the Rubin LSST (Ivezić et al., 2019a) and ZTF (Bellm, 2014) surveys. Of these, only the later is currently ongoing.

The Zwicky Transient Facility (ZTF) is a survey designed to systematically monitor the sky to detect and study transient events and time-variable astronomical events. These include supernovae, variable stars, active galactic nuclei, asteroids, comets, etc. ZTF uses the Samuel Oschin 48-inch Schmidt Telescope at Palomar

[1]https://ui.adsabs.harvard.edu/

Observatory in California, USA, equipped with a 47-square degree field of view camera. The camera contains 16 large CCD detectors that capture images in multiple filters simultaneously. ZTF covers a significant portion of the sky in the Northern Hemisphere and it observes in three optical bands (g, r, i). It is particularly sensitive to faint and fast-changing objects. The data obtained is processed by a pipeline that performs data reduction, calibration and transient detection in real-time. Thanks to this, alerts can be issued when an interesting object is detected, which enables follow-up observations.

The Rubin Legacy Survey of Space and Time (LSST), on the other hand, is an incredibly ambitious project meant to conduct a wide and deep survey of the Southern Hemisphere sky over a period of ten years. It will use a 8.4-meter diameter mirror, a 3.2-gigapixel camera and a wide-field survey camera with a field of view of about 9.6 square degrees to observe the sky repeatedly in six optical bands (u, g, r, i, z, y). The Vera C. Rubin observatory is located in Cerro Pachón in Chile. The objective of this survey is to obtain data that will allow us to further the study of dark matter, dark energy, explosive transients and near-Earth objects. The amount of data (over 15 TB per night), length and depth of the survey will hopefully allow us to gain valuable insight about the time-variable universe and the nature of the cosmos.

However, the unprecedented data volume also poses a significant problem: of course, the more data one has, the more it costs to sort it out. The scale of the multi-PB data set will require real-time sophisticated filtering, in order to identify events that are of particular interest, such as explosive transients. In order to be able to automatically classify these objects, we could start by looking at the data we already have and use it to experiment with our machine learning algorithms. For example, in order to prepare for the Rubin LSST, one could study the ZTF case. It is also important to look at the nature of data. In general, the data products for optical transient astronomy are of two kinds: difference imaging and time series (i.e: light curves). Both of these will depend on the specifics of the telescope in question and on the atmospheric observational conditions, hence, introducing difficulties for data analysis: difference in cadence across surveys, sparsity of samples, and noise. In this thesis, I will specifically focus on the problem of classification of different types of supernova light curves.

It might seem unwise in the midst of a global pandemic, its aftermath, the imminent threat of global warming and a potential AI apocalypse to devote so much effort to a seemly unimportant question -how to classify certain types

of light curves- but it is a question small enough to tackle, though even the smallest whats and hows are actually huge. Indeed, there is a lot of different axes even to this constrained compact problem. In order to make sense of them and to properly present my research journey, this introductory chapter will be structured as follows: In Section 1.1 I will talk about why this problem is relevant for astronomy and will give an overview of the astrophysical background, as to understand the motivation. In Section 1.2 I will introduce basic machine learning concepts necessary to understand my research and will explain the challenges related to this area. In Section 1.3 I will recount how machine learning has been applied to astronomy, specifically to light curve classification to date, and what this field of study has in common with the treatment of regular time series.

1.1 Astronomical Background and Motivation

1.1.1 Transient Astronomy and its current state

Although when we think about the universe our life can seem like a mere instant when compared to the life of a star, some celestial events happen in a way that is possible for us to perceive even within our short years. Some are fleeting and transitory, and can last seconds, months, weeks or years, depending on the section of the electromagnetic spectrum that we are observing, and on the event itself. Other events are not brief, but exhibit a repetitive pattern in a span of time short enough for us to recognize it. Transient astronomy is the discipline devoted to the study of short-lived type of phenomena, called transients. This discipline is a subset of a broader topic: Time domain astronomy, that is, the study of how astronomical objects change with the passing of time.

Transient events include but are not limited to: supernovae (SNe), tidal disruption events, kilonovae, active galactic nuclei, and other violent deep-sky phenomena. In the following work, however, I will consider mostly supernovae. The reason for this, as we will see, is that their light curves are particularly difficult to tell apart.

For explosive transients, the ones that have their brightness rise rapidly and last a limited amount of time, one challenge is to know when and where to look and how to detect something interesting early enough as to be able to observe as much of the phenomenon as possible. Surveys like ZTF, the Dark Energy Survey (DES) (Collaboration: et al., 2016) and Catalina Transient Survey (CRTS) (Djorgovski et al., 2011a), spot transients by automatically rechecking the same regions in the sky regularly to detect significant changes. They do so by difference imaging, that is, subtracting a real time image with a reference one (or vise versa) that is already in an archive (Tomaney & Crotts, 1996). ZTF triggers an alert when there is a detection in a difference image that exceeds 5σ signal-to-noise threshold and, after an alert is triggered, necessary follow up observations can be performed by different telescopes.

Since there are still many mysteries surrounding explosive transients, it is important to observe them soon after the explosion. To ensure that the follow-up observations are organized, we need to be able to classify them early enough, that is, hopefully within a few days of the trigger, which is where early classification

of light curves becomes an interesting problem. Some transient brokers, such as ALeRCE[2](Förster et al., 2021), ANTARES[3](Matheson et al., 2021) and Lasair[4](Smith et al., 2019) are dedicated to doing this, although they differ in their focus and their techniques. To really understand the motivation behind the automatic classification initiatives, let us review first what we currently know about SNe and what we do not know but hope to learn.

1.1.2 Supernovae: what we know and what we do not know

Everything that is born must die, and stars are not the exception to this inescapable rule. The fate of stars and the manner of their death is largely determined by their mass.

The thermonuclear life of a star starts drawing to a close once fusion in its core stops. Some stars can resist gravitational collapse thanks to electron degeneracy pressure. The dense object that remains is about a hundred times smaller than the sun, though its mass is comparable (their maximum stable mass is $\approx 1.4 M_{\odot}$, the Chandrasekhar limit). These objects can spend billions of years slowly dying by cooling off. Some of them, however, may absorb enough mass for fusion to be re-ignited in their cores, and so they come back to life like a phoenix rising from the ashes. They then undergo a runway reaction and go off in a much more dramatic and spectacular fashion: a thermonuclear explosion that is so bright it can be seen across the Universe and be used to measure it. These are Type Ia supernovae (SNe-Ia).

More massive stars ($> 8 M_{\odot}$) suffer a different fate. After nuclear fusion has stopped in their cores, they cannot, ultimately, resist their own gravity and collapse into themselves. This collapse produces a shock wave response that makes the star explode into a supernova, shedding most of its mass, and so the interstellar medium is enriched by the matter ejected and the new elements synthesised during the explosive event. These core collapse supernovae (CCSNe) leave their cores as remnants behind, either as a neutron star, if neutron degeneracy pressure is enough to halt the collapse, or a black hole otherwise.

A picture of a galaxy with and without this type of SN can be seen in Figure 1.2.

[2]http://alerce.sience
[3]http://antares.noao.edu
[4]http://lasair.roe.ac.uk
[5]https://www.instagram.com/leftysastrophotography/, https://www.flickr.com/people/leftysastrophotograp

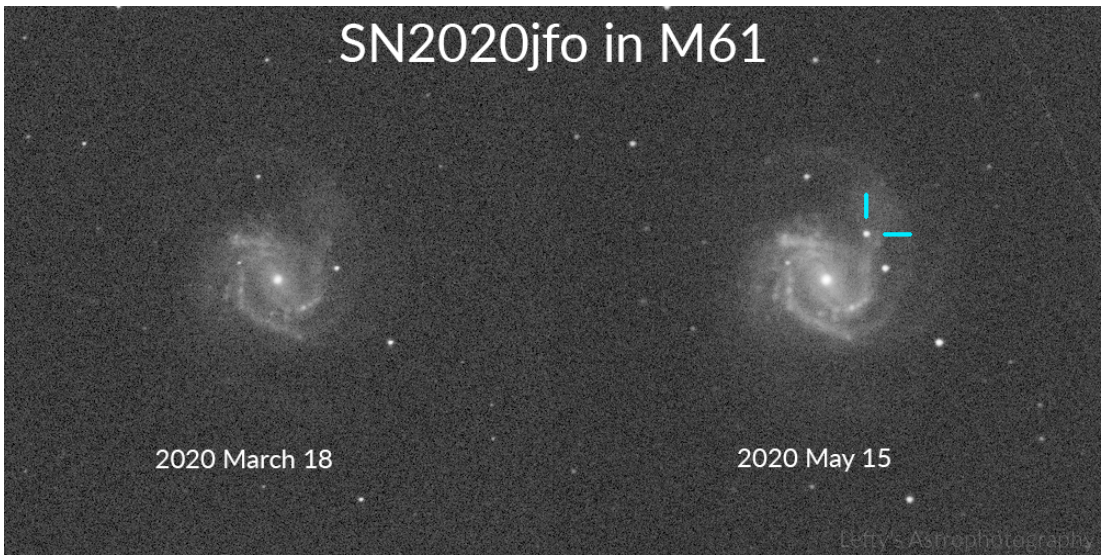


Figure 1.2 SN2020jfo before and after explosion in galaxy M61. This Type II supernova was discovered by ZTF. The picture was taken and edited by Lefty's Astrophotography[5], who allowed me to use it but made it clear they prefer to maintain their online presence anonymous.

Humanity has been studying SNe for centuries, across different cultures. The earliest SN evidence recorded goes back to 185 A.D, when Chinese astronomers saw a 'guest star' appear for 8 months in the sky. There is some evidence of Roman astronomers also recording the event, which is now known as RCW86 (Helder et al., 2009). Chinese astronomers recorded another supernova event now cataloged as G347.3-0.5 in 393 A.D. In 1006, a supernova so bright that is supposed to have been visible during daytime (its peak magnitude is estimated to have been -7.5) was recorded by Chinese and Japanese astronomers, as well as in chronicles from Iraq, Iran, Egypt, Europe (Neuhäuser et al., 2017; Rada & Neuhäuser, 2015) and even rock art in North America (Hamacher, 2014), although this last example is under debate. Personally I have a hard time believing people in America would not have noticed and made art out of it, but it is difficult to prove or retrieve any records for obvious reasons. There are also historical records of SN1054, the supernova that produced the Crab Nebula (See Figure 1.3) in 1054, coming from several different cultures.

Despite these early studies, there are still a lot of uncertainties surrounding SNe. Since upcoming surveys, such as the Rubin LSST, will operate for longer periods of time than previous surveys, they could have a huge impact on our

[5] https://www.astrobin.com/itrx92/

understanding of these events.

SNe taxonomy, unfortunately for this thesis, has little to do with photometry (with the exception being the division between SNe-IIL and SNe-IIP) and all to do with their spectral lines. Supernovae Type I (SNe-I) show no hydrogen in their spectrum, while supernovae Type II (SNe-II) do. SNe-Ia present an ionised silicon line and are the only thermonuclear type. All other common supernova types (Ib, Ic, IIb, IIP, IIn, IIL) are thought to be core collapse. The taxonomy is this confusing for historical reasons, but we can see a summary of it in Table 1.1.

<table>
<tr><td rowspan="3">Type I
No hydrogen</td><td colspan="3">Ia
Si II line at 6.15 nm</td></tr>
<tr><td rowspan="2">Ib/c
weak or no silicon absortion</td><td colspan="2">Ib
shows He I at 587.6 nm</td></tr>
<tr><td colspan="2">Ic
weak or no helium</td></tr>
<tr><td rowspan="4">Type II
Shows hydrogen</td><td rowspan="3">II P/L/n
Type II spectrum throughout</td><td rowspan="2">II P/L
no narrow lines</td><td>IIP
light curve flattens after explosion</td></tr>
<tr><td>IIL
linear decrease in light curve</td></tr>
<tr><td colspan="2">IIn
Some narrow lines</td></tr>
<tr><td colspan="3">IIb
Spectrum becomes like Type Ib</td></tr>
</table>

Table 1.1 Supernova taxonomy (Turatto, 2003). Supernova Type Ia (cell in light blue) is the only one thought to be a thermal runaway. All the other supernovae are core collapse.

The visual light curves of supernovae in general show a sharp increase in luminosity followed by a slower decline that lasts for months. The glow of these light curves is sustained by the radioactive decay of the elements produced during the explosion. If it were not for the radioactive heating of the ejected material, the SNe would quickly deem out. The variations in their amplitude and general shape depends on the way this radiation is produced, that is to say, the underlying mechanism, how transparent the ejected material is and the epoch of its observation all play a role.

Potentially, supernovae could be strong sources of gravitational waves, though none have been detected as of yet (Morozova et al., 2018; Solheim & Yungelson, 2005).

1.1.2.1 Supernovae Type Ia

SNe-Ia are the most common and fairly sought after because of their cosmological use as standard candles. Although initially they were thought to be relatively homogeneous, the more studies their usefulness inspired, the more questions regarding their sub-classes and variability appeared (Hillebrandt et al., 2013).

Although there are many uncertainties regarding how this type of SN comes to be, there is consensus in that they happen when a white dwarf, usually made of carbon and oxygen, undergoes a thermonuclear explosion. Since white dwarfs are very stable, they need to accrete enough matter for that stability to be disturbed. The threshold that determines when this happens is called the *Chandrasekhar limit*, which is $\approx 1.4 M_{\odot}$. Once a white dwarf gathers enough mass to exceed this value, it can no longer support its own weight Chandrasekhar (1931).

How close the white dwarf gets to the Chandrasekhar limit when gaining mass, what the progenitor system looks like exactly and the details of the explosion are a matter of debate (Gamezo et al., 2003).

There are several possible scenarios that might result in a SN-Ia. One possibility would be that there is a collision with another compact object within a globular cluster, but this would not account for the number of these events that we observe. The most likely scenario is that the white dwarf gets extra matter from a binary companion (either a star or another white dwarf) (Langer et al., 2002).

If the companion of the white dwarf is either a star in the main sequence or a red giant, we have what is called a Single-Degenerate scenario (SD) (Whelan & Iben, 1973; Hillebrandt et al., 2013). In this scenario, the companion overfills its Roche-lobe (the region around an object in a binary system where orbiting material is bound to the object by gravity) and so transfers matter (usually hydrogen, but also helium in rarer cases (Solheim & Yungelson, 2005; Postnov & Yungelson, 2006)) to the white dwarf. Once the white dwarf accretes enough mass to approach the chandchandrasekhar limit, carbon fusion restarts. This increases the temperature of its core, but since white dwarfs are sustained through electron degeneracy, pressure does not increase and the white dwarf cannot expand and cool itself. This triggers a thermal runaway reaction: the rise in temperature causes a rise in the fusion reaction rate, which causes a rise in temperature, in a feed-back loop. The runaway reaction releases enough energy to cause a thermonuclear explosion that completely destroys the star (Mazzali et al., 2007;

Hillebrandt & Niemeyer, 2000).

The Double-Degenerate mergers scenario (DD) (Webbink, 1984), on the other hand, is thought to occur when two carbon-oxygen white dwarfs merge together and their combined mass exceeds the Chandrasekhar limit. Although this scenario initially received a lot of criticism because it was thought it would result in the collapse to a neutron star (Saio & Nomoto, 1985; Shen et al., 2012), more recently it has been suggested that an ignition might occur in the case of 'violent' mergers (Liu et al., 2017; Piro et al., 2014).

But there are also rarer cases of SNe-Ia. Supernovae Type Iax are sufficiently different and common to perhaps need their own separate classification. This type of SN are less luminous and present lower ejection velocity. They are theorized to happen when a carbon-oxgen white dwarf accretes helium stably from a companion (either another degenerate or a star (Iben & Tutukov, 1991)) and never approaches the Chandrasekhar limit, but still explodes. The white dwarf is thought to accumulate a layer of helium which might detonate under the right conditions, and this detonation in the shell would then trigger a second detonation in the carbon oxygen core. This is known as the Double-Detonation scenario (Wang et al., 2013). SNe-Iax and SNe-Ia are similar in spectra, and though at late times they may diverge, their composition still suggests a physical connection. SNe-Iax occur at a rate between 5 and 30% of the normal SNe-Ia occurance. It is not clear whether SNe-Iax destroy their progenitor or whether they may leave a zombie star (Foley et al., 2013; McCully et al., 2014).

In all these scenarios there are uncertainties regarding the evolutionary process during accretion, the initiation of burning and the formation of detonations, but we know that the result of them is the ejection of the entire mass at a high kinetic energy.

As for the light curves of SNe-Ia, which is the part that concerns this thesis the most, they achieve their peak luminosity in approximately 18-20 days, followed by a rapid decline of about three magnitudes in one month. In the near infrared, they show a second maximum around 20 days after the first one (Hillebrandt & Niemeyer, 2000). Weaker explosions are less luminous, redder, and have a faster declining light curve, than the more energetic ones, that is to say, there is a correlation between the peak brightness and the width of the light curve (Hamuy et al., 1996).

1.1.2.2 Core collapse supernovae

Core collapse supernovae, on the other hand, happen when massive stars stop fusion in their core and, as the name suggests, collapse against their own gravity. For most CCSNe this happens because after a stable iron core is formed, electron degeneracy pressure is not enough to sustain them. Their implosion elevates the density in their inner core so much that a rebound shock wave is produced and propagated to the outer imploding core. This turns the implosion into an explosion, where a huge number of neutrinos are emitted, heavier elements are created and the mantle of the star is ejected, leaving behind a neutron star or, if the progenitor was massive enough and had the right metallicity, a black hole (Heger et al., 2003). CCSNe are on average visually fainter than SNe-Ia, but the total energy released is far higher, and the emitted electromagnetic radiation is only a tiny fraction of it.

Although this mechanism has served as suitable explanation for decades, computer simulations disagreed. In most simulations, the process failed to cause an explosion and the shock wave would be stalled into accretion. In the earlier stages of the collapse, the high densities in the core would produce neutrinos through electron capture on protons and these neutrinos would become trapped in the opaque inner core. As the rebound shock wave travelled outwards, the decrease in density would cause a burst of neutrinos which would take away energy from the shock wave and cause it to stall (Woosley & Janka, 2005). What then, caused this shock wave to be re-energized? It was not clear, for a while, and even now some researchers maintain that the matter is far from resolved because there are little observational signatures (Pejcha, 2020).

The main constraint in the study of the core of CCSNe in the last half century used to be that running multiple simulations was far too computationally expensive and did not allow for multiple dimensions. Spherical models rarely collapsed, and this is why it is theorized that asymmetry plays a huge role in the explosion. In recent years, however, simulations have become increasingly more sophisticated, thanks again to the advancement in the processing power of GPUs. We now know that turbulence, the structure of the progenitor, details about how neutrinos interact with high density matter and the rate at which the inner core accretes matter during the implosion are all key to determine the outcome of the collapse. In particular, neutrino heating in the gain region behind a stalled shock wave and neutrino driven turbulence seem to be agents of the explosion (Burrows &

Vartanyan, 2021; Mabanta & Murphy, 2018). This explains why previous models failed to explode and, recently, different codes and simulations have started to agree (Burrows & Vartanyan, 2021; O'Connor et al., 2018).

It is agreed that once the collapse has started and the shock wave is stalled, the inner core maintains a quasi-equilibrium because it increases in radius by accretion of matter falling inwards but decreases by neutrino loss. Just inside the stalled shock wave, in what is called the *gain region*, where neutrinos that escape the inner core deposit energy (Burrows & Vartanyan, 2021). This process is known as neutrino heating and was controversial in the early 2000's, as different research groups obtained a variety of different results (Janka, 2001). Current simulations show that the neutrino heating mechanism is much more efficient than previously thought, and also that it powers turbulence (Mabanta & Murphy, 2018).

The matter falling to the inner core has perturbations that enable turbulence flow to increase, which in turn makes the matter accreted take longer paths towards the inner core. Since matter takes longer to reach the inner core, there is more time for neutrinos to get to the gain region, making the conditions for neutrino heating more favourable (Müller et al., 2017). As the scatter rate of neutrinos increases (and the proto neutron star shrinks) the mean energy of emitted neutrinos increases as well. As the neutrino heating rate increases and more energy is deposited in the gain region, turbulence becomes more violent. The turbulence dissipation then contributes to the thermal pressure, and all of this makes the supernova explosion more likely (Foglizzo et al., 2005). Although we know that the energy transfer to the gain region happens through the absorption of neutrinos into nucleons and electrons, figuring out details of the process (particularly figuring out how exactly the spectral energy distribution is changed) is still computationally difficult Bruenn et al. (2020).

Simulations also show that the actual explosion happens along an axis, with a narrow region in the middle of it that allows the core to keep accreting mass while the explosion is happening. Lower mass progenitors explode faster, but have less explosion energies and the opposite is true for more massive progenitors, because more massive stars have shallower density structures, while less massive stars have steeper ones. If the core collapse of a star happens when it still has a hydrogen envelope, then a SNe-II occurs. If a star has lost most of its hydrogen envelope before the collapse, then the result is a SNe-Ib or SNe-Ic. Whether a star sheds its hydrogen envelope or not has to do with the metallicity of its environment and its luminosity. The accretion during explosion can generate relativistic jets

that result in gamma-ray bursts or supernovae that are super luminous (Kasen et al., 2011a) (SLSNe). It is also believed that SNe-Ib/c could potentially produce gamma ray bursts, if their geometry permitted it (Ryder et al., 2004; Bhandari et al., 2020).

Although clearly the understanding of CCSNe has increased immensely and the field is advancing rapidly, there are still many details and uncertainties left to solve.

1.1.2.3 Super luminous supernovae: a special case

While both thermo-nuclear and core-collapse supernovae reach a peak luminosity of no more than $10^9 L_\odot$, and release $\approx 10^{51}$ erg in kinetic energy, super luminous supernovae (SLSNe), are a type of explosive transient that is at least an order of magnitude more luminous, hence the name (Nicholl, 2021). While for regular SNe research centers around the details of their explosion mechanisms, for SLSNe the main question is: how are they so very luminous?

We know that for them to have such high luminosity and sustain it after the explosion, there must be something heating up the matter that they eject, but there is no consensus yet as to the heating source. Still, there is a lot of information regarding their characteristics.

SLSNe are classified as traditional SNe: Type I (hydrogen poor) and Type II (hydrogen rich). There are many more studies concerning SLSNe-I than SLSNe-II due to the lack of samples, so sometimes researchers refer to SLSNe-I simply as SLSNe (Nicholl, 2021).

Although SLSNe-I usually show a peak magnitude above Mg = -19.8, it is difficult to define a clear threshold (Gal-Yam, 2019), since it would depend on the survey properties and since outliers exist. SLSNe-I, however, show a very distinctive spectrum at the time of peak luminosity, so it makes more sense to classify any event with this type of spectrum as SLSNe, regardless of whether they make the cut (Nicholl, 2021).

SLSNe-I initially go through a hot photospheric phase, where their spectrum shows a blue continuum with broad OII absorption complex, unique to SLSNe-I. Several strong absorption lines can be seen in the UV, and the Hubble Space Telescope has shown that a significant part of their luminosity is emitted in the

UV. Weaker OI and CII lines in the red part of visible light are also displayed during this phase (Quimby et al., 2018). These characteristics last through peak magnitudes and even a few weeks later, when a cool photospheric phase ($\approx$ 12000 K) starts and their spectra becomes very similar to those of SNe-Ic. This is maintained during the nebular phase, where they show Fe emission lines in the blue part of their spectra (Jerkstrand, 2017). Since in the later stages of their spectral evolution they resemble SNe-Ic, that they have been considered to be super hot SNe-Ic (Nicholl, 2021).

Surprisingly, some hydrogen poor SLSNe show hydrogen emission lines in their spectra at later times (100-200 days after the explosion), which suggests that maybe they have caught up with their shed hydrogen envelopes (Gal-Yam, 2019). For this to be possible, they would have had to lose their envelope mere decades before they exploded (Lunnan et al., 2018). If their helium layer was also expelled, the interaction with their ejected material could contribute to their luminosity at their hot photospheric phase.

Samples of SLSNe-I appear most often in dwarf galaxies with low (sub-solar) metallicities, which is counter intuitive, as one would expect that a super luminous explosion would come from a massive progenitor found in galaxies with higher mass. It seems then that low metallicity has to play an important role (Nicholl, 2021). Unlike the case with SLSNe-I, galaxies with higher metallicities do not exhibit a lower fraction of regular SNe-Ic (Gal-Yam, 2019).

SLSNe-I light curves are characterized not only by their unusually high luminosity, but by the longer time scales for both rise (100-200 rest frame days) and decay (not well defined because they would need to be followed up for a long time), when compared to regular SNe-I. They also appear to be bumpy (Nicholl et al., 2016). Although initially there were events that showed bumps previous to peak luminosity times, it was later shown that they were only displayed in few of the samples, so they are not considered to be a defining characteristic (Gal-Yam, 2019). At later times after peak luminosity however, they do show undulations in events that decline slowly.

Three types of heating mechanisms have been proposed that could explain how SLSNe maintain their remarkable luminosity: The radioactive decay of elements produced during the explosion, the existence of a central engine, and the interactions of the ejecta with circumstellar material (CSM) (Gal-Yam, 2019; Nicholl, 2021; Chen, 2021).

Radioactive decay (specifically of ^{56}Ni) is what sustains the light curve for regular SNe-I, but for SLSNe-I the amount of nickel produced through nucleosynthesis would have to be much higher to account for their observed magnitudes. In theory, Pair-Instability Supernovae (PISN) (Kasen et al., 2011b) could produced the amounts (several $M_{\odot}$) required, but so far the spectroscopic models do not agree with observations (Nicholl, 2021).

Central engine models are perhaps a more likely explanation. Possible candidate sources for an engine are remnants such as magnetars, rapidly spinning neutron stars with a magnetic field strength greater than $10^3 G$. Magnetar models fit many samples of SLSNe-I, as they seem to be consistent with observations of both spectra (Nicholl et al., 2017) and light curves (De Cia et al., 2018). For a magnetar to be a valid candidate though, the core of the collapsing progenitor would have to spin fast enough to create a neutron star with a rotational period of less than ≈ 10 milliseconds. They would have to be less likely to lose angular momentum, which is consistent with the low metallicity environments (Nicholl, 2021; Gal-Yam, 2019).

A magnetar central engine, however, would not account for bumps in the light curves before or after peak luminosity. Additionally, masses of the ejecta predicted by models do not match those derived from observations (Gal-Yam, 2019). Another criticism is that the light curves this model predicts are perhaps too diverse when compared to other models (Nicholl, 2021). Still, observed light curves *are* diverse.

Black holes have also been considered to be a possible central engine, since fallback accretion could liberate enough gravitational energy to power SLSNe-I (Dexter & Kasen, 2013). However, although the unstable mass accretion could explain undulations in the light curves, when contrasting models to observed samples it seems that this case would require a fall back mass of $\approx 10 - 100 M_{\odot}$, which is not realistic (Nicholl, 2021).

It could also be that the heating source that powers SLSNe is not within its core, but outside. Interactions with circumstellar material (CSM) offer scenarios that would make this possible for SLSNe-I, specially since they lost their outer layers, which could be the source of CSM. Hydrogen lines in the spectra of some SLSNe-I samples could be evidence of this. It is possible that Pulsational Pair Instability Supernovae (PPISN) (Woosley, 2017) would cause the progenitor to shed layers in a series of events, which could possibly explain late bumps in the

light curves (Chen, 2021). Some CSM interaction models fit the observed light curves as good as magnetars do, but no CSM model fits observed SLSNI-e spectra as of yet (Gal-Yam, 2019). CSM interaction models are perhaps a better fit for SLSNe-II (Nicholl, 2021), since they appear to be similar to SNe-IIn, where this type of interaction is observed. We do not know for certain, since there are not many samples of SLSNe-II to compare.

Given that there are many observed features that theory needs to account for, perhaps it is more likely that the mechanism responsible for the heating of ejecta is actually a hybrid of two or more of the explanations presented (Gal-Yam, 2019). Hopefully, upcoming surveys will provide enough samples to enable further research based on statistics.

1.1.3 Usefulness of Supernovae in cosmology

Of all transients, SNe-Ia are particularly important because of their usefulness in cosmology as standard candles. A sub type of astronomical objects or events are considered to be standard candles if their intrinsic luminosity is known or can be estimated with a degree of certainty. Given than the observed brightness of a source measured on Earth is proportional to its luminosity and inversely proportional to its distance squared, standard candles make it possible for us to estimate distances in the universe.

As discussed in 1.1.2.1, SNe-Ia are thermonuclear explosions that occur when carbon-oxygen white dwarfs accreate enough mass to approach the Chandrasekhar limit and ignite fusion that in turn produces a thermal runaway reaction. Because the masses of white dwarfs before explosion are similar, SNe-Ia present similar luminosities, which was confirmed through observations of nearby events (Churazov et al., 2014). SNe-Ia, however, do present differences in their luminosities and so are *standardizable* candles, rather than standard (Coelho et al., 2014). The standardization process became possible when it was discovered that intrinsically brighter SNe-Ia are also the ones with the wider light curves. Accounting for the correlation between peak luminosity and decay time, allows us to determine whether a given SN looks brighter because it is nearer than another such event or because it is intrinsically brighter (Phillips, 1993).

Since SNe-Ia are incredibly bright and can be observed in very distant galaxies, they are ideal for estimating really long distances. The intrinsic luminosities of

SNe-Ia are fairly consistent, so we can determine their distance from earth based on their apparent brightness. They still need to be calibrated though, which can be done by comparing their apparent brightness with those of nearby SNe-Ia that had their distances measured by other means. We can also measure the redshift of SNe-Ia and both these values allow us to build a luminosity-redshift relationship, which can be plotted to construct a Hubble diagram.

It was thanks to this type of supernova that in 1998 it was discovered that the universe is not only expanding, but also accelerating while doing so (Riess et al., 1998; Schmidt et al., 1998). Currently, the most accepted explanation for this accelerated expansion is the existence of an unknown form of energy, called dark energy, which would generate gravitational repulsion. The properties of dark energy can be constrained by analyzing the Hubble diagram and comparing it to the existent dark energy models. However, to measure properties such as the dark energy equation of state and its density, we need to get hold of a large enough sample of this type of event (Dai et al., 2018) and it is important that this sample be as immaculate as possible.

There are, however, uncertainties concerning how standarizable SNe-Ia actually are. While some exceptionally bright SNe-Ia are thought to have emerged from white dwarfs that exceed the Chandrasekhar limit by a margin ($\approx 1.8 M_{\odot}$) (Hachisu et al., 2011), there is also evidence of really dim explosions (SNe-Iax).

SNe-Ia might not be the only type of supernova useful for cosmology (Gal-Yam, 2019). It has been suggested that since SLSNe-I have similar luminosities, they could be standarized and used as cosmological probes, although this is not yet clear, since the light curves of SLSNe-I are diverse. There has been debate about considering a subdivision of SLSNe-I to separate rapidly declining events from slow declining one, which could potentially help with the standarization process (Gal-Yam, 2019). This subdivision has not been accepted yet though, because sample studies have failed to find a clear division so far (De Cia et al., 2018; Lunnan et al., 2018). One study considering unsupervised clustering techniques did find that a set of events clustered into two groups, but the one of the groups (the slow category) contained too few events for this work to be conclusive (Inserra et al., 2018).

1.1.4 Hopes for the future

We have seen that there are still many puzzles left to be solved when it comes to the study of SNe.

In the case of SNe-Ia, we do not know the details of the initiation of burning and the formation of detonations, and there is questions regarding the masses of their progenitors and how that affects how standarizable they are.

In the context of CCSNe, researchers are reaching a general consensus as to their explosion mechanisms, but details are not agreed upon and these details are important, as they include mapping characteristics of the progenitor stars to those of the explosions. As it stands, even for the same star, simulations vary widely in terms of times, directions, energies and residual mass of the explosions. While CCSNe are thought to be responsible for huge emissions of neutrinos, the only supernova neutrino event detected so far is SN1987A (Agnes et al., 2021), so having a greater sample of these events to study could improve our understanding of particle phenomena (Seadrow et al., 2018).

While we know many characteristics of SLSNe-I, we have little data of SLSNe-II. What mechanism (or combination of mechanisms) powers their light curves is still not known and it is necessary to keep looking for clues that could enable us to map models to observations. Having more SLSNe-I samples might also help us figure out whether they can be standarized.

Considering all these reasons, it is clear that having more data and being able to classify it correctly is crucial, since it will enable researchers to perform statistical sample analysis that might shed light upon some of these enigmas. The Legacy Survey of Space and Time (LSST) that will be conducted at the Vera Rubin Observatory presents exciting possibilities for gaining insight, provided we know what to do with the data we will get.

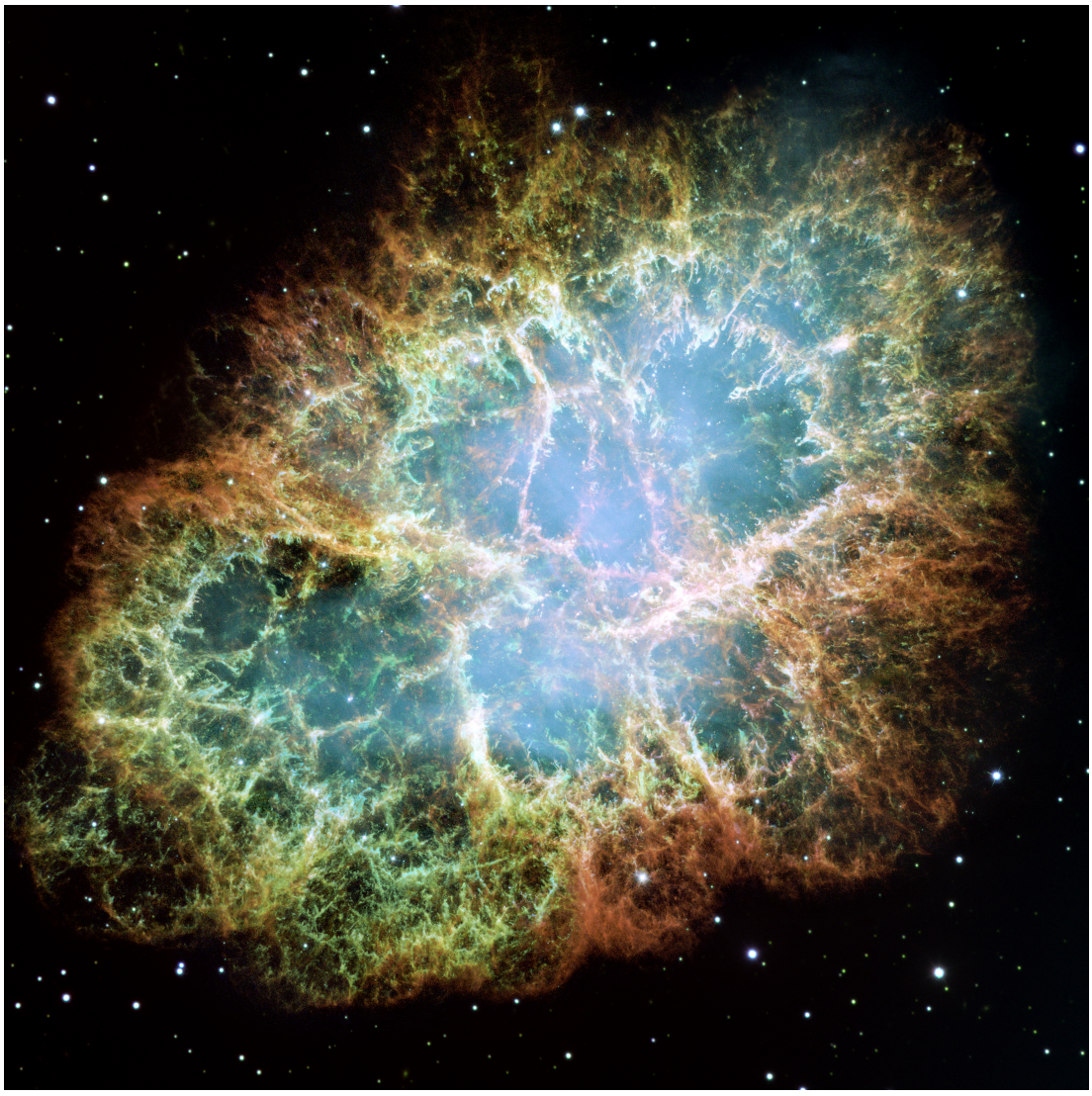


Figure 1.3 The Crab Nebula, a renmant of SN1054. This is a mosaic image taken by the Hubble Space Telescope. The bluish glow at the center comes from a rapidly spinning neutron star, while the orange filaments are the remains of the deceased star.

1.2 Relevant machine learning background

Computer science and its continuous improvement of algorithms has helped us greatly in the automation of tasks that are time consuming for humans to perform, which has of course aided a wide range of sciences. While today it is clear that Artificial Intelligence (AI) is a thriving research field which has the potential to transform a wide range of industries (Poole et al., 1998; Russell & Norvig, 2009), there is still no consensus on its formal definition (Collins et al., 2021; Bhatnagar et al., 2018; Legg & Hutter, 2007). One commonly cited definition, however, is the following:

> *'The theory and development of computer systems able to perform tasks that normally require human intelligence.'*
>
> (Mitchell, 1997)

In other words, AI involves creating machines, systems and/or agents that can display intelligence performing tasks such as: recognizing patterns in data, decision making, visual perception, speech recognition, decision-making, translation between languages and more (Joiner, 2018; Goodfellow et al., 2016).

Although it is widely accepted that an AI is a machine that can 'think humanly', major researchers prefer to the term 'act rationally' and would argue that an artificial agent need not be intelligent the way a human is (Russell & Norvig, 2009), as that view is unnecessarily anthropocentric. Just as some animals are incredibly intelligent in ways that are vastly unknown to us (octopuses think with their tentacles and can morph their skin into the colours presented by their environment, which is a very intelligent thing to do, one could argue) an artificial agent could interact with its environment in an effective way and yet think in ways that are not necessarily similar to human thought.

AI is usually divided into Weak AI and Strong AI (Collins et al., 2021). While Weak AI refers to systems designed to perform a specific task equally or better than humans; Strong AI, also referred to as Artificial General Intelligence (AGI), refers to systems capable of performing any intellectual task a human would be able to do, such as making sense of its surroundings, understanding and learning. As of today, an AGI is not yet a reality, although given the speed at which the field is developing, I would wager it will be soon. Various nations are indeed

aware of the fact and are devoting efforts to legislating accordingly[6789].

For the moment, however, we can determine what tasks seem important to us humans and create agents that are proficient at tackling those specific tasks.

1.2.1 Artificial Intelligence, Machine Learning and Deep Learning

One example of a cognitive function an AI could perform, perhaps the most important of all, is learning. There are two widely cited definitions of what Machine Learning (ML) is:

> *"The field of study that gives computers the ability to learn without being explicitly programmed."*
>
> (Samuel, 1959)

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*
>
> (Mitchell, 1997)

Informally then, ML studies how an agent can improve the way they carry out a task without a human directly telling them. It is important to note, for the sake of clarity, the distinction between algorithm and model: an algorithm is a recipe, a series of steps that define a method, while a model is a representation of some part of reality. A *machine learning algorithm* is a procedure that creates a *machine learning model*. Often the ML model and the ML algorithm share the same name ('neural network', for example), but ML algorithms usually group many other algorithms ('gradient descent' and 'back-propagation' in the case of a neural network), that work to create and output a structure, the model, which is where the information that is learnt is stored.

[6] https://www.gov.uk/government/publications/ai-regulation-a-pro-innovation-approach/white-paper
[7] https://artificialintelligenceact.eu/
[8] https://www.parl.ca/DocumentViewer/en/44-1/bill/C-27/first-reading
[9] https://www.whitehouse.gov/ostp/ai-bill-of-rights

There are several different approaches to ML. One of the most popular ones is *Supervised learning*, which needs a data set previously labeled in order to feed it to a model and prepare it to perform a certain task over new unlabeled data. The task can be either a regression (prediction) or a classification, both of which are essentially a function that maps an input to an output (Russell & Norvig, 2009). *Unsupervised learning*, on the other hand, aims to find patterns in unlabelled data through self-organization (Hinton & Sejnowski, 1999).

Both these approaches can be combined in *Semi-supervised learning*, an approach which uses a small portion of labeled data and and a bigger set of unlabeled data from which a model can learn and make predictions[10][11]. This approach is in my opinion of special interest for astronomy given the nature of astronomical data. It will also be relevant in Chapter 5 of this thesis.

Some other popular examples are *Reinforcement learning*, which is a process where an agent maximizes the amount of rewards it receives while interacting with a complex environment (Sutton & Barto, 2018), and *Transfer learning*, which refers to using the knowledge learnt from one problem and applying it to a different one (West et al., 2007).

All ML models have their benefits and shortcomings. What model is the most useful for a specific problem we need to solve is not a straightforward decision, although some were specifically designed with a task in mind and perform well for it, or have been empirically proven to work well for certain situations.

Another important point to ML is to decide how to represent the data a model needs to learn. Traditionally, ML tackles data analysis problems (such as the classification of data) in two stages: in the first stage, features are handcrafted and extracted from the data, while in the second stage, said features are given to a trainable model so it can learn. These engineered features aim to encapsulate the relevant aspects of a class[12]. For example, if I wanted a model to learn to decide whether it is wise for me to go on a hike today, it would be relevant for it to know details about the weather (wind velocity, temperature, humidity), about the route itself (length, altitude) and about my current health (am I ill?, am I hydrated?, did I sleep enough?). Deciding what kind of information a model needs to be given is called feature engineering.

[10] https://en.wikipedia.org/wiki/Semi-supervised_learning
[11] https://scikit-learn.org/stable/modules/semi_supervised.html
[12] https://ai.stanford.edu/ ang/slides/DeepLearning-Mar2013.pptx

Unfortunately, unlike the hiking example, it is often not straightforward to decide what kind of features should be picked to get a model to learn. Indeed, feature engineering is a research field itself. There is, however, another way to go about it: instead of crafting the features we want, we can teach a model to extract whatever features work best for what it needs to do. This approach is called Deep Learning (DL)(Goodfellow et al., 2016) and it is based on involves Neural Networks (NNs).

For the specific problem I am trying to solve in this thesis (light curve classification of supernova) several different methods have been tried (we will go over them in Section 1.3), some involve feature engineering, some do not. I am going to focus on DL methods, for several reasons: first, because it has been done less often than the feature engineering approach; second, because DL has been outperforming traditional approaches in other disciplines for a while now; and last, because I simply find it more interesting to see what the model thinks is the most relevant information to extract from a light curve.

However, before we can move on to the rest of the thesis, it is necessary to do a quick review of some key ML and DL concepts that are relevant for the problem at hand. This review goes into the basic mechanisms of NNs (Neural Networks) and how they learn, so a reader familiar with the field may skip it, as it is intended for a someone not familiar with the topic.

1.2.2 Machine learning as an optimization problem

One of the core components of most ML algorithms is optimization (Sun et al., 2019). The aim of optimization is to find a set of inputs to an objective function that results in a maximum or minimum function evaluation. The goal of ML is to construct a model and tweak its parameters so the output of that model is as close as possible to an expected target. The measure used to define the distance between a given output and a desired output is called *loss function* or *cost function*.

ML models essentially learn by minimizing their cost function through an iterative process in which they update their parameters. This process is called a *learning algorithm*.

There are several different cost functions. Which one is used will depend on the

problem that we are trying to solve. A common loss function used for multi-class classification problems (which is the kind of task with which this thesis is concerned) is the *cross-entropy error function*(or *log loss*) (Janocha & Czarnecki, 2017), which builds upon the idea of *entropy* from information theory (Shannon, 1948).

The entropy $H(X)$ of a random variable X with distribution p, also denoted $H(p)$, is a measure of its uncertainty. For a discrete variable with K states, it is defined as:

$$H(X) = -\sum_{k=1}^{K} p(X = k) \log p(X = k). \tag{1.1}$$

Cross-entropy, on the other hand, can be interpreted as a measure of distance between two distributions p and q, or the average number of bits needed to encode data coming from a source with distribution p when we use a model q (Murphy, 2012) (Cover & Thomas, 2006). It is defined as:

$$H(p, q) = -\sum_{k=1}^{K} p_k \log q_k. \tag{1.2}$$

In the case of ML and classification, K represents the number of classes, p represents the probability distribution the model is trying to learn (the labels, in supervised learning) and q represents the predictions of our model.

There are also many types of learning algorithms. To choose one, we need to look not only at the task at hand, but also at the type of model that we are trying to build. In the next Section 1.2.3, we will briefly review Neural Networks and the way they learn.

1.2.3 Artificial Neural Networks

Artificial Neural Networks (NNs) were first inspired by how the animal brain works: neurons connected to each other to propagate impulses (Rosenblatt, 1958). Although the first computational model of a NN was presented in 1943 (Fitch,

1944), it was not until 1960 that researchers figured out an algorithm to train them (Kelley, 1960), and even then it all remained theoretical, since machines still lacked the the computational power (and storage) to make the implementation possible.

In the 1980s however, the development of metal-oxide-semiconductors allowed for a higher transistor count in digital electronics, making the computation of more complex problem possible and allowing the construction of the first dedicated Graphic Processing Unit (GPU) in 1987, by IBM (WayBackMachine, 2011). GPUs became a lot more sophisticated in the early 2000s, when Nvidia and AMD both launched GPUs capable of computing pixel shading (Witheiler, 2001) (AMD, 2008). Many times in history technology has advanced with warfare as a motivation, but funnily enough the motivation in this case was to provide a better experience for the gaming community[13].

In 2007 nVidia introduced CUDA (Pu, 2008), a programming model specifically designed for GPUs, and in 2009, finally, NNs had become advanced enough as to begin win prizes in ML competitions, outperforming other models and getting near to human performance in several tasks (Graves & Schmidhuber, 2009).

Without GPUs this simply could not have happened. While algorithms like *Backpropagation*, used to train NNs, have existed on paper since the 70s (Werbos, 1975), they would be too slow to be used in practice if it were not for the fact that they can be parallelized. GPUs are designed to perform thousands of simple mathematical operations simultaneously, which makes them ideal for matrix multiplications and element-wise operations. These are used intensively in backpropagation. Additionally, the larger the NN, the more parameters and computations are required. Also, the training process itself can be parallelized by dividing the training data into mini-batches and processing it on different GPU cores. GPUs also usually have a higher memory bandwidth than CPUs, which makes it easier to handle data movement. All these characteristics of GPUs contribute to the acceleration of computationally intensive algorithms and thus the rapid growth seen in the field of ML.

There are many NN architectures and ways of customizing them and many more appear constantly. They all, however, have the same building blocks: neurons, connections, weights and activation functions.

[13] https://en.wikipedia.org/wiki/Graphics_processing_unit_GPU_companies

1.2.3.1 Neurons, connections, weights and activation functions

The simplest NN model is known as the *Multi-layered perceptron* (Murtagh, 1991) (MLP), which consists of one input layer, one output layer and at least one hidden layer between them, shown in Figure 1.4. Each neuron takes in an input and produces an output. Between neurons there are connections that have weights assigned to them. These weights represent the relative importance a given connection has within the context of the entire network. The neurons in NNs are organized in layers, which makes the process of learning manageable. A neuron computes its output by taking the weighted sum of all its inputs (either the input data or the output of other neurons) plus a bias and running this term through an *activation function*.

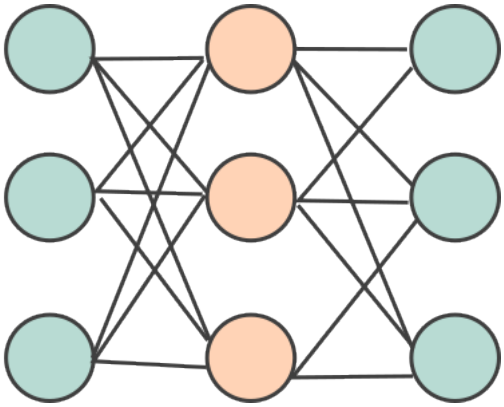


Figure 1.4 Basic Multi Layer Perceptron

An activation function (Lecun et al., 1998) determines the output of a neuron in an NN. In a chip circuit, an activation function would be a perceptron where the output of the activation is either 0 or 1, that is, an impulse is either propagated or it is not. In a modern NN, however, activation functions are essential for complex problem solving because they introduce non-linearity into the network. If it were not for these non-linear functions, an NN would be limited to representing linear relationships only, which would dramatically reduce their usefulness. There are several non-linear functions that can serve as activation functions.

Initially, it was common to use sigmoid or hyperbolic tangent since they are differentiable and thus can be used in gradient based methods such as backpropagation. Currently, the most used activation function is the *rectified linear unit* (ReLU)(Nair & Hinton, 2010), which is defined by the positive part

of its argument:

$$f(x) = max(0, x) \tag{1.3}$$

Although ReLU looks qualitatively different to its earlier counterparts, which quite clearly approximated a choice between two states, it has a number of advantages. It is computationally cheaper and mitigates the problem of vanishing gradients (Glorot et al., 2011), which makes it empirically more effective and thus popular in the machine learning community.

Soft-argmax (Bridle, 1989), commonly referred to as *softmax* in the machine learning community (and in the rest of this thesis) is another popular activation function. It takes a K dimensional vector z as input and maps it to a vector where each value is within the range $(0, 1)$. In the context of a classification problem, it is usually used in the final layer of a network, where K can be interpreted as the number of classes. The K^{th} element of the softmax output vector is defined as:

$$y_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \tag{1.4}$$

In other words, softmax is the multi-class version of the two-class sigmoid. As sigmoid models a Bernoulli distribution, so softmax models a categorical distribution, giving as an output a set of probabilities, one for each different class.

1.2.3.2 Backpropagation and Gradient Descent

For an ML model to learn, it needs to minimize a cost function C using a learning algorithm, by updating its parameters. In the case of a NN, the parameters that need to be updated are all the weights $w^l_{i,j}$, which are connections between the j^{th} neuron in layer l and the i^{th} neuron in layer $l-1$. To update them, we need to know how much they contributed to the final error, or in other words, we need to calculate all the derivatives $\frac{\partial C}{\partial w^l_{i,j}}$.

If we define z^l as the weighted input of layer l, a^l as the output, and f as the

activation function, we have:

$$z^l = W^l a^{l-1} + b^l \tag{1.5}$$

$$a^l = f(z^l) \tag{1.6}$$

Where W^l is a matrix of the weights connecting layer l with layer $l-1$ Using the chain rule, we know that the derivative of the cost C with respect to an input x of the network is:

$$\frac{dC}{dx} = \frac{dC}{da^L} \circ \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \circ \frac{da^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{da^{L-2}} \cdots \frac{da^1}{dz^1} \cdot \frac{\partial z^1}{\partial x} \tag{1.7}$$

Where each term is a total derivative and $\circ$ is the Hadamard product (or element-wise product). Because of the definitions of a^l and z^l, we see that the terms $\frac{da^l}{dz^l}$ represent the derivatives $(f^l)'$ of the activation functions and the terms $\frac{dz^l}{da^{l-1}}$ are simply the matrices of weights W^l. We can then rewrite 1.7 as:

$$\frac{dC}{dx} = \frac{dC}{da^L} \circ (f^L)' \cdot W^L \circ (f^{L-1})' \cdot W^{L-1} \ldots (f^1)' \cdot W^1 \tag{1.8}$$

Or equivalently:

$$\nabla_x C = (W^1)^T \cdot (f^1)' \ldots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} C \tag{1.9}$$

The core idea of *backpropagation* (Lecun, 1992) is to calculate all these terms efficiently from right to left. In other words, we want to make use of the structure of the network: we start by calculating the gradients with respect to the weights in the last layer, and recursively propagate this calculations backwards through the layers of the network.

If we define the error at level l as the auxiliary term δ^l:

$$\delta^l := (f^l)' \circ (W^{l+1})^T \ldots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} C \tag{1.10}$$

We have that the gradient of the cost with respect to the weights in layer l is:

$$\nabla_{W^l} C = \delta^l (a^{l-1})^T \tag{1.11}$$

And from the last layer of the network to the first, we can compute the gradients by recursively calculating δ^l. That way, we reuse calculations from previous gradients.

$$\delta^{l-1} = (f^{l-1})' \circ (W^l)^T \cdot \delta^l \tag{1.12}$$

Once we have efficiently calculated the necessary gradients using backpropagation, we need to use them to update the weights. The way we change these parameters to minimize the cost, is through a learning algorithm called *Stochastic Gradient Descent*(SGD)(Robbins & Monro, 1951).

Stochastic Gradient Descent (in Algorithm 1) is an optimization algorithm that considers that a multi-variable function $C(W)$ (our cost) decreases fastest if one goes from W in the direction of the negative gradient $-\nabla C(W)$, provided that C is differentiable and defined in a neighbourhood near W. The idea is that for a predefined number of *epochs*, we will update the weights in our network iteratively by calculating the necessary gradients (using backpropagation) for every training example (x_i, y_i) in our training dataset D. An epoch is one complete pass of D through the network H.

Algorithm 1 *Stochastic Gradient Descent*

Input: η, D
1: Initialize W
2: **for** epoch in epochs **do**
3: **while** there are examples x_i, y_i in dataset D **do**
4: Draw random (x_i, y_i)
5: $\hat{y}_i \leftarrow H(x_i)$
6: $C_i \leftarrow C(y_i, \hat{y}_i)$
7: $W \leftarrow W - \eta \nabla_W C_i(W)$
8: **end while**
9: **end for**
10: **return** W

The *learning rate* η represents the velocity at which we update the weights

(Ruder, 2016). Since the weights of the model get updated after each sample passes through it, the gradient estimate can contain a significant amount of noise. While it is easier for the algorithm to escape local minima, the noise makes the convergence process less stable. Also, since we deal with only one example at a time, we cannot use a vectorized implementation and that will impact the computation time negatively. However, there is a more efficient version of SGD: *Mini-Batch gradient descent*.

In this case (See Algorithm 2)the weights get updated after a subset of m samples in the dataset D has gone through the network. The number of batches is $M = D/m$:

Algorithm 2 *Mini-batch Gradient Descent*

Input: η, D

initialize W
for epoch in epochs **do**
    while there are mini batches $x_{[i:i+m]}, y_{[i:i+m]}$ in dataset D **do**
        Draw random $x_{[i:i+m]}, y_{[i:i+m]}$
        $\hat{y}_{[i:i+m]} \leftarrow H(x_{[i:i+m]})$
        $C_M \leftarrow C(y_{[i:i+m]}, \hat{y}_{[i:i+m]})$
        $W \leftarrow W - \eta \bigtriangledown_W C_M(W)$
    end while
end for
return W

Updating the weights after each mini batch is the most practical solution and the one most used. Using mini-batches of data instead of single samples reduces the noise in the gradients, since each mini-batch average cost C smooths out some of the randomness. This makes it more robust compared to SGD. However, there are still challenges to the approach:choosing η can be difficult, sometimes it might not make much sense to apply the same η to all parameter updates, and sometimes the optimization process might get stuck in local (instead of global) minima. These challenges inspired the development of a number of improvements over SGD, such as the addition of *momentum*, the idea of performing bigger updates when consecutive gradients point in the same direction and reducing updates if the gradients change direction (Ruder, 2016).

In the following chapters of this thesis we will be using *Adam*(Kingma & Ba, 2014), a learning algorithm that is fairly popular due to its empirical good performance. Adam (explained in Algorithm 3) considers a variable decreasing

learning rate, which means that as the learning progresses, the amount by which the weights are changed decreases. It also considers two momentum terms, m_t and v_t, that represent the mean and the variance of the weights.

Algorithm 3 *Adam*

Input: η
Input: $\beta_1, \beta_2 \in [0, 1)$
Input: ϵ
for epoch in epochs **do**
  while there are mini batches $x_{[i:j]}$ in dataset X **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_W C(W_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1)$
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2)$
    $W_t \leftarrow W_{t-1} - \eta \cdot \hat{m} / (\sqrt{\hat{v}} + \epsilon)$
  end while
end for

The terms β_1, β_2 and ϵ are hyperparameters, the first two represent exponential weight decay rates and the last one is a small number to avoid dividing by zero. Even though Adam seems to perform really well, there is still substantial research on further improving this type of learning algorithm. Some ideas on this are changing η according to a predefined schedule and 'restarting' η when it has decreased too much (Loshchilov & Hutter, 2017).

1.3 Light curve classification

A spectrograph decomposes light into different wavelengths to obtain measures of specific fluxes at a given time. In an ideal world we would like to obtain a detailed spectrum at every point in time for every interesting astronomical event, but this both technically difficult and very expensive. Instead, we can get a cheap approximation by measuring in multiple broad passbands centered on different wavelengths and thus getting 'colours' for different points in time. This approximation is a multi-band light curve.

There are several different aspects that are relevant to the task of light curve

classification, but perhaps it would be wise to start by looking at the problem in more general terms. After all, light curve classification can be regarded as a special case for the broader case of time series classification.

1.3.1 Time series classification

Formally, a time series is a set of data points ordered across the time dimension. A uni-variate time series consist of $X = [x_0, x_1, ..., x_t]$ real values, where t is the number of data points. A multi-variate time series is a set $M = [X^0, X^1, ..., X^m]$ of m different uni-variate time series, where $X^i \in \mathbb{R}^t$. Given a data set $D = (X_0, Y_0), ..., (X_n, Y_n)$ where X_i is either a uni-variate or multi-variate time series and Y_i corresponds to a label representing a class, the task of time series classification (TSC) consists of getting a classifier to correctly map from the space of possible inputs to a probability distribution over the labels. Informally, TSC aims to find patterns in data and classify accordingly either into classes that are predefined or that are defined on-the-fly.

There are several approaches to tackle TSC. We will first go briefly over the more traditional approaches and then review the differences they have compared to deep learning approaches, which is what concerns this thesis the most.

1.3.1.1 Dynamic Time Warping, Shapelets, Dictionaries, Forests and Ensembles

Aside from deep learning, there are several types of TSC strategies (Bagnall et al., 2016) (Ruiz et al., 2021). One of the most common techniques is Dynamic Time Warping (DTW) combined with K-Nearest Neighbour (K-NN), which is usually used as a benchmark algorithm.

K-NN is a simple clustering algorithm where, in this case, given a set of labeled time series, a new one is classified by finding the K labeled samples closest to it and assigning it the class most common amongst these neighbors. The obvious issue is that in order to assert that a pair of time series are similar to each other, we need first a way to measure that similarity. A naive way to do this would be to simply compare the time series point by point, add these distances and then chose the lowest value. However, this approach will most likely fail, since we have no assurance that the events encoded in the time series begin at a similar time

frame. In other words, the curves we are comparing might look very much alike, but if one of them is slightly shifted in the time dimension, we might end up with a high value of distance that is not representative of the similarity between them.

DTW is a way to pair data points between sequences that might not align in time, speed or length in an optimal way. Given two time series, $S = [s_0, ..., s_n]$ and $T = [t_0, ..., t_m]$, the optimal DTW pairing is the one with the lowest cost that satisfies the following rules:

- Every point s_i must be paired with one or more elements t_j and vice versa.
- s_0 must be paired with t_0
- s_n must be paired with t_m
- The mapping of the indices from S to the indices from T has to increase monotonically and vise versa. That is, if we visualize every pair s_i and t_j as lines, between the two curves S and T, these lines must not cross.

The cost is the sum of the absolute distances between the pairs. In a naive implementation, we get the minimum cost by simply comparing every element in sequence T to every element in sequence S, which would give us a complexity time of $O(mn)$. However, very recently this time complexity was improved to $O(n^2/log(log(n)))$ for the specific case where the two sequences have the same length n (Gold & Sharir, 2018). There are also some other ways to compute the lowest costs, all of them require specific conditions (Müller, 2007).

The main benefit of DTW combined with KNN is that not a lot of data is needed since there is not a model to train. Instead of holding a representation of the different classes of sequences, these two algorithms follow sets of rules and decide on the label. The drawback, however, is that since all the calculations are done on the fly, the classification takes a lot of time and thus does not scale well for large data sets.

There are other such approaches that do not construct models and perform better than DTW over certain data sets. One fairly common technique is the use of shapelets, which are subsets of time series that encapsulate specific patterns indicative of a class (Ye & Keogh, 2009). The idea is similar to DTW, except instead of comparing two entire sequences, we now compare a representative bit of time series with a whole sequence. The shapelet transform is the minimum

distance between the shapelet and all of the subsequences of the time series, and that distance can then be used to decide on a class. It is not straightforward, however, to know which shapelet to use. One way to decide would be to simply craft one manually, but we might not always know what are exactly the features that we need to encapsulate. Another way would be to automatically select a shapelet, which is an active field of research. Another obvious drawback is that, just like DTW, this approachs takes a lot of time.

Interval-based approaches, on the other hand, include Random Forests specifically for time series. Random Forests (RF) (Ho, 1995) are ensembles of Decision Trees (DT). Decision Trees (DT) (Wu et al., 2007) are a method where a model is created, in the shape of an upside down tree, to predict the value of a variable based on several input features. The structure is created by considering all the input features to select a split point that minimizes the cost function. The process is recursive in nature, because the groups divided can be split again by considering the remaining features. This learning algorithm is known as *recursive partitioning* (Strobl et al., 2009) and the partitions created are binary in most cases, since dividing features into a larger size of categories would greatly increase computational complexity. Random Forests attempt to that mitigate overfitting by having each DT take a 'vote' on the output.The final output is decided by majority voting in the case of classification while for regression the average of the DT outputs is returned.

An example of this is a Time Series Forest, which constructs those trees by dividing the TS into intervals and for each getting the mean, deviation and slope.

Dictionary-based approaches are another category of methods used for TSC. As the name suggest, they rely on constructing a dictionary and using it to train a simple classifier, such as a logistic regression. For example, a dictionary could store the number of matches of a shapelet with a time series. In that case, the shapelet would be the key, and the count would be the values stored. BOSS Strobl, Malley & Tutz (sch) is such an algorithm, it uses Fourier transforms to map time series into frequency space, does a low-pass filtering and then translates those into strings. To do this, intervals (bins) are defined and associated with the strings (these would be the keys), the signal is divided and we count how many times those bits fall into the bins (these would be the values of the dictionary).

Finally, we have model ensembles, which are one of the most successful methods, but also the most costly in terms of time. They combine different other approaches

together and so achieve a higher accuracy. The most famous are COTE (Bagnall et al., 2015) and HIVE-COTE (Lines et al., 2016). COTE involved 35 different algorithms and then chose the most common class for each time series. The problem though is that an ensemble only makes sense if the algorithms that contribute to it are uncorrelated, otherwise a class might have a higher score only because some algorithms are similar enough as to draw the same conclusions. COTE suffered from exactly this problem, and its successor HIVE-COTE solved it by grouping algorithms that used similar approaches and then having each category 'vote' on the final class. HIVE-COTE is the best performing algorithm for TSC that does not involve neural networks. Note that, of course, how well an algorithm performs will depend highly on the data set it is used on. The algorithms for TSC are usually benchmarked against the UCR/UEA TS archive [14] (Dau et al., 2019), but there are other datasets that are also used.

All of these approaches might perform well, but do not scale well as data volume increases because their time complexity is simply too high. Also, most of them do not create a model and so the whole process needs to be repeated for each time series that needs to be classified. An exception to that would be TS Forest (or Random Forests in general), but the drawback there is that the features and rules used might not be the most representative of the patterns that we need to catch.

1.3.1.2 Recurrent Neural Networks, Convolutional Kernels and Convolutional Neural Networks

Deep Learning differs from traditional approaches in that it does not rely heavily on feature engineering, but instead attempts to extract useful features through the process of learning itself. Since the process of hand-crafting features is not straightforward and often time consuming, this approach is relevant for several fields of data analysis, and promising for TS analysis in particular (Gamboa, 2017).

DL also has the advantage of creating models instead of just applying algorithms and so although the training process is costly, the actual classification is very fast. The cost, however, is that the amount of data needed to train a NN is higher, so it might not always be feasible to build a robust model. In the forthcoming surveys though, the data volume will increase greatly and so, even though we

[14] http://timeseriesclassification.com/

might not have enough data to do what we would like at first, eventually we will.

It is important to highlight that TSC is not the same as TS forecasting (TSF), and so it is not necessarily the case that an approach that works for one of the tasks would work for the other. While for classification it is necessary to find patterns in the data that are different between different classes, in forecasting the aim is to find patterns in the data that help predict the near future, so the focus would be on the end of the time series. In particular, there is debate on whether Recurrent Neural Networks (RNNs) work well for time series classification, even though they are well known to achieve high performance for prediction.

RNNs (Sherstinsky, 2020) are a type of neural network that process each time step t of a sequence X in a different copy of a neural network, also called unit or hidden state. Each unit h_t gets information not only from the input X_t, but also from the unit h_{t-1}that processed the time step X_{t-1}. In other words, the output of each unit is:

$$h_t = tanh(W_{hh}h_{t-1} + W_{hx}X_t)$$

Where $tanh$ can be replaced by a different activation function. Figure 1.5 is a graphical representation of the whole RNN and Figure 1.6 shows a single unit h_t.

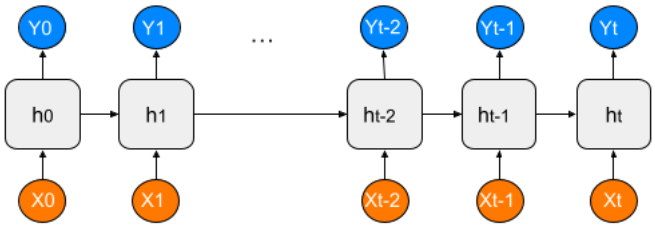


Figure 1.5 Recurrent Neural Network and how it processes sequence X

The main issue with RNNs is that they suffer from the vanishing gradients problem (Informatik et al., 2003). This complication arises when, during back-propagation, the gradients get so increasingly small that the network fails to learn. The RNN is especially susceptible to it because back-propagation is applied through time. In other words, the RNN is *unrolled* so that it becomes a neural network made of N duplicates where each of them represents the original network at a different time step. That makes an RNN incredibly deep, causing the gradients to vanish and the network to forget its first steps. This also means that it is very expensive to train and difficult to parallelize (Pascanu et al., 2013).

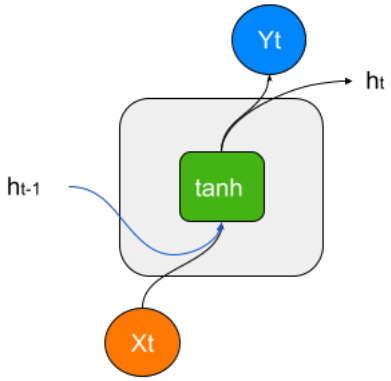


Figure 1.6 Unit h_t of an RNN.

To solve this issue, there are two popular types of RNN units: LSTMs (Long Short-Term Memory cells) and GRUs (Gated Recurrent Units).These units both have in-built mechanisms to remember information (*cell states*) from previous units. These mechanism work through *gates*. A gate essentially looks like:

$$g_t = \sigma(W_g X_t + R_g h_{t-1} + b_g),$$

where W, R and b are matrices and vectors learnt during training, while σ is an activation function.

Both LSTMs and GRUs are widely used for TSF and, generally, GRUs outperform LSTM networks on low complexity sequences (Cahuantzi et al., 2021), while on more complex sequences it seems to be the other way around, which makes sense, given that LSTMs have more parameters. GRUs will be described next in more detail, as they are used in Chapters 4 and 5 of this thesis.

GRUs (Cho et al., 2014) have two gates, the update gate u_t, that decides whether cell state should be updated with the current activation value, and the reset gate r_t, that determines how much of h_{t-1} should be forgotten. A GRU (in Figure 1.7)is defined by:

$$\begin{aligned}
z_t &= \sigma(W_z X_t + R_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r X_t + R_r h_{t-1} + b_r) \\
\dot{h}_t &= tanh(W_h X_t + R_h(r_t \circ h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \dot{h}_t
\end{aligned}$$

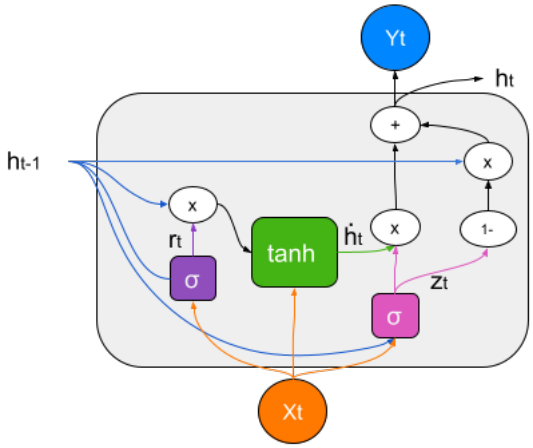


Figure 1.7 Gated Recurrent Unit (GRU). It receives the hidden state output h_{t-1} of the previous unit (blue lines) and a time step of a sequence X_t (in orange) and performs operations with these through gates: the update gate u_t (in purple), the reset gate r_t (in pink) and a gate that outputs the candidate cell state $\dot{h}_t$ (green). Each gate has trainable matrices and vector associated with it and the label on them denotes an activation function.

While there is literature making the case that features that are useful for TSF are related to those needed for achieving high performance in TSC (Hüsken & Stagge, 2003) (Malhotra et al., 2017), most reviews of the state of the art for TSC either show that RNN architectures do not perform particularly well when compared to other approaches or fail to include RNNs at all (Wang et al., 2016a) (Ismail Fawaz et al., 2019) (Fawaz, 2020) (Bagnall et al., 2017). I think it is unwise to make a general statement as to whether the same features would necessarily work for both tasks: as with most things in ML, it will depend on the data. It makes sense that for classification of SNe RNNs would work though, since they focus on the sequentiality of the data.

Although the reviews mentioned consider several DL architectures and try them over many data sets, only a few approaches perform consistently well across studies and benchmark data sets. Most of them involve *Convolutional kernels*.

A Convolutional kernel (or filter) is a mechanism that can potentially capture important features in an input. A kernel has the same structure as the input, but much smaller. In the context of TSC, a kernel is a vector of weights that is convolved with an input sequence by performing a sliding dot product operation (See Figure 1.8). A bias term is added to the result of the convolution. The result is a *feature map*. The amount by which a kernel slides through an input is

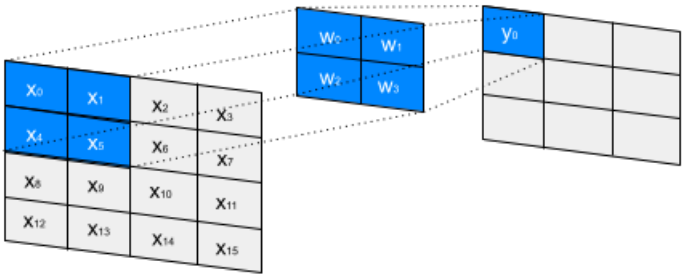


Figure 1.8 Convolutional filter with weights w_i slides over a section of the input.

called *stride*. A kernel is also defined by its size (length), *dilation* and *padding*. Dilation spreads the kernel over an input so the weights in it are not convolved with every point in the sequence, but only with some elements at a given interval. This helps capture the same pattern at different scales and enhances the ability of the kernel to find relations between different parts of the input (Antoniou et al., 2018). Padding, on the other hand, is concatenating zero values to the start and end of the input TS so that the middle of the convolutional kernel is aligned with the first point in the sequence.

Convolutional Neural Networks (CNN) (Lecun & Bengio, 1995) first achieved popularity in the ImageNet Large Scale Visual Recognition Competition in 2012 (Russakovsky et al., 2015). It was much later theorized that since TS are esentially one dimentional images CNNs could be used for these as well.

The basic idea of a CNN is to stack *convolutional layers* (that contain sets of convolutional kernels) together to extract increasingly smaller feature maps and then have a mechanism at the end (a fully connected layer, for example) that performs the actual classification. Both the convolutional layers and the final layer are learnt through back-propagation (Lecun et al., 1998).

Usually between each convolutional layer there are *pooling layers*, where parts of the feature maps are pooled together to reduce their dimensions (Szegedy et al., 2015) (Ranzato et al., 2007). Some typical pooling operations are taking an average (average pooling) or taking the maximum value (maxpooling) (Ciresan et al., 2011). Intuitively, we can see how a maxpooling operation would focus on the sharpest features of a section, while average pooling would be smoothing those features. From the signal processing perpective, average pooling would be analogous to a low-pass filter, while maxpooling would be similar to an envelope extraction. The aim of pooling and dimensionality reduction of feature maps is

to allow deeper layers in a CNN to construct complicated features from simple building blocks. Thus, CNNs can be viewed as hierarchical structures where the first layers extract information in the local context of the input while deeper layers indirectly interact with larger parts of it and work on a broader context (Albawi et al., 2017). This was inspired by the organization of the animal visual cortex (Fukushima, 1980), so in a way, the part of a CNN that extract features would act as the eyes of the network, while the final end would act as the brain that makes the decision.

Pooling, however, is not the only way to achieve dimensionality reduction across layers. Increasing the stride of a convolutional kernel decreases the overlap of the convolutional operation and thus effectively amounts to downsampling (Springenberg et al., 2015). It is important to note that, since the information recovered in the deeper layers of a CNN directly depend on the information gathered on the lower layers, having a big stride on the shallower layers not only decreases local detail in the captured features but detail overall, although this varies depending on the architecture. Also, the bigger the stride, the smaller the feature map resulting from that convolutional layer will be, thus decreasing computational costs. Dilation also has a part to play in changing the dimensions of feature maps, since dilating the kernel means broadening the context of a network while also producing bigger feature maps, which increases computational time (Antoniou et al., 2018).

There are mainly two CNN architectures that stand out in TSC reviews (Bagnall et al., 2017) (Fawaz, 2020) (Ismail Fawaz et al., 2019) (Wang et al., 2016a) because they achieve similar performance to ensemble methods (in less time) and significantly outperform other DL approaches: Fully Convolutional Neural Networks (FCNs) and Residual Neural Networks (ResNet).

Although there are different definitions in the literature, loosely speaking, FCNs are a type of CNN that only performs convolutions. They were originally proposed for semantic segmentation of images (Long et al., 2015), that is, the task of identifying different objects withing an image. The main idea is to get rid of the final fully connected layer that would usually decide on a class in a CNN so that the network can have outputs of variable size. Instead of giving a single class as an output, in a FCN there is a different label for each pixel, so the output is a coarser version of the original input that outlines different things within it.

It essentially works as encoder-decoder model: one part of the network performs

convolutions and down-sampling to extract feature maps and another part takes those feature maps and performs deconvolutions (up-sampling) to obtain the final output. In a regular CNN, only the convolutions would be performed and then the outputs of those would be flattened and given to a fully connected layer, which has a fixed size. In a FCN, that flattening is not applied, instead, the encoder part outputs n feature maps, where n is the number of classes and these are used as a code to reconstruct the image.

How could this be useful for TSC? Well, in the context of TSC, the proposed architecture (Wang et al., 2016a) (Ismail Fawaz et al., 2019) uses only the encoder part to extract meaningful features in a TS and those are used for the final classification. How is this any different from a regular CNN? In TSC literature, an FCN does not have pooling operations after each convolutional layer, but has a global pooling operation at the end of the network that acts as a substitute for a fully connected layer (Lin et al. (2013a)). The idea is that the number of feature maps produced by the global pooling is the same as the number of categories relevant for the classification, so that each of those feature maps acts as a confidence map for each of the categories. Since fully connected layers are prone to over-fitting, a global pooling operation (that does not need to learn weights and hugely reduces the number of parameters) reduces that risk. It is not clear to me why the architecture presented originally (Long et al., 2015) and the one derived from it for TSC (Wang et al., 2016a) have the same name, since they seem widely different except for the fact that they do not use fully connected layers.

Residual Neural Networks, on the other hand, are NNs where layers are stacked in blocks, where each block has a shortcut to the next one. This has two advantages: it helps reduce the problem of vanishing gradients and mitigates degradation of the input, that is, when a network is deep enough for the final layers to miss important information from the lower levels (He et al., 2015). In this thesis we will be using a Convolutional Residual Neural Network.

While some standard NN architectures were just described, in Chapters 4 and 5 specific models and mechanisms will be described in more detail. In particular, Attention, Multi-headed Attention and the Transformer will be explained. They are not mentioned further here simply because it makes more sense from a narrative perspective to read about them just as they are about to be used.

It is important to note that the approaches discussed here work well across several

data sets, but that does not necessarily mean they will definitely be useful for the classification of light curves. Not only are light curves multivariate TS, which means we need to consider not only the sequences themselves but their relation to one another, but also they are sparse, noisy and not standarized across datasets. Indeed, in one of the TSC reviews (Bagnall et al., 2017) the PLAsTiCC dataset, which simulates LSST light curves, is considered and even the best performing approaches do not do well. In the next section we will go over the difficulties of dealing with light curves and how machine learning has been applied to them.

1.3.2 Brief survey of light curve classification of explosive transients

The analysis of variable astronomical phenomena is an important instrument to study the formation and evolution of the universe, as well as an interesting subject in itself. As the volume of observational data continues to increase, machine learning methodologies for the specific issues that arise continue to be researched and improved upon.

Machine learning has been applied to studying light curves for more than a decade and it has mainly been used for exoplanet detection, variable star analysis and SNe classification (Yu et al., 2021). We will focus on the latter in this section, since it is the case that concerns this thesis.

Dealing with light curves is uniquely challenging, as they present many of the characteristics that makes TSC difficult. Observation constraints, weather conditions and technical factors result in light curves being unevenly sampled and noisy, which complicates the process of feature extraction and places a huge importance on pre-processing. Differences between surveys make it hard to have a standard, robust and correctly labeled data set to train classifiers with, or to use as a benchmark. Often simulations do not resemble real-life light curves, and this discrepancy between training and test sets is known to impact the performance of supervised methods negatively. Additionally, classifications need to be updated dynamically as new data comes in. Besides, for each transient that we observe, we usually consider more than one light curve: the observations are done in different pass bands, which results in two or more light curves that belong to the same object, and that relate to each other in ways that might be relevant for the identification of the type of the event in question. This makes the problem

a Multi-Variate Time Series Classification problem (MTSC), which is even less researched than TSC.

Both traditional and deep learning methods have been applied to light curve analysis. The need for the development of specifically tailored methodologies and the possible advantages of neural networks (Belokurov et al., 2003) was recognized early on, as we knew with at least a decade in advance that the next generation of survey telescopes would observe so many events that as to make it unfeasible to spectroscopically confirm the type of each.

In the context of SNe classification, many works (Richards et al., 2011) (Karpenka et al., 2012) (Ishida & de Souza, 2013) (Brunel et al., 2019) (Burrows & Vartanyan, 2021) (Carrick et al., 2021) (Möller & de Boissière, 2019) (Pasquet et al., 2019)(Qu et al., 2021)(Charnock & Moss, 2017a) have remarked on the importance of distinguishing SNe-Ia from other types of SNe, because of the need of having samples of real SNe-Ia with high purity in order to used them for cosmological inference, as discussed in 1.1.3.

Early studies of light curve classification (LCC) using machine learning (Mahabal et al., 2008) (Richards et al., 2011) (Bloom et al., 2012) (Karpenka et al., 2012) (Ishida & de Souza, 2013) focused on classical approaches relying heavily on feature engineering. The crafting of these features was by no means straightforward, as the field of multi-dimensional TS has just recently been looked into in more depth. In order to deal with this type of data, much attention was placed on dimensional reduction techniques, that had the additional advantage of making computational costs cheaper (Richards et al., 2011) (Karpenka et al., 2012) (Ishida & de Souza, 2013). While some approaches focused on statistical features, others gave much more importance to the modeling of light curves and used the fitting parameters as features.

Researchers have also given some consideration to feature reconstruction: Gaussian processes have been used to reconstruct the light curves themselves (Dhar Gupta et al., 2016) (Boone, 2019) and Bayesian networks have been employed to create graphs of feature dependencies that help to predict missing measurements and so craft features (Pichara & Protopapas, 2013). Both these methods have some disadvantages: Gaussian processes are costly and do not scale well with data volume, while modeling a hierarchy of features with Bayesian networks implies that we know what features matter most and their importance, which is not at all true. Simpler methods, such as linear interpolation (Muthukrishna et al., 2019)

(Möller et al., 2016) have also been tried. The need to address this issue is clear: since surveys observe in different bands and since they are constantly observing different parts of the sky, light curves present gaps in which no observations are available. These parts that lack information might just be important enough to cause the classification of the event to fail. In the context of SNe classification, maybe peak luminosity was not observed, or maybe bumps in the light curves were missed, etc.

Since it is not straightforward to figure out what features matter most when classifying light curves automatically, ML techniques such as Random Forests have been used to learn ranking of features. A lot of attention has been paid to photometric redshift. While some work states that it is one of the most significant features (Boone, 2019), especially at high redshifts (Möller et al., 2016) (Carrick et al., 2021), other analysis find that it does not impact classification results significantly (Karpenka et al., 2012) (Lochner et al., 2016) (Charnock & Moss, 2017a) It is important to note that the works that concluded that photometric redshift was not impactful for classification results all focused on SNe light curves, specially on SNe-Ia, and used the SNPCC (Supernova Photometric Classification Challenge) (Kessler et al., 2010) simulated dataset.

Works regarding LCC differ in the taxonomy they consider. Early on the different classes given to light curves were broarder: there were distinctions between periodic, cataclysmic or explosive events (Bloom et al., 2012), between galactic vs extra galactic events, and between SNe-Ia and all other SNe types (Karpenka et al., 2012) (Ishida & de Souza, 2013) (D'Isanto et al., 2016).

More recent articles differentiates between several types and sub-types of objects (several different types of supernovae for example) (Malz et al., 2019) (Sánchez-Sáez et al., 2021) (Gabruseva et al., 2019) (Boone, 2019)(Muthukrishna et al., 2019) (Förster et al., 2021)(Malz et al., 2019) (Sánchez-Sáez et al., 2021)(Burhanudin et al., 2021), with the motivation of making the analysis of data easier for several different fields. In the case of SNe, while the binary classification of SNe-Ia versus all other types remains relevant (Brunel et al., 2019) (Burrows & Vartanyan, 2021) (Carrick et al., 2021), the identification of rarer SNe types such as SLSNe and SNe-Iax gained importance, both because of the possibility of them being useful in cosmology and because of the interest in understanding the mechanisms underlying the events themselves. Being able to identify different types of CCSNe also became relevant, both to map observations to current sophisticated models and to gauge their similarity and relation to

SLSNe.

While early on some NN models were tried (Belokurov et al., 2003) and compared to other approaches to seek best performance (Mahabal et al., 2008)(Karpenka et al., 2012), RFs were the most popular approach and are still much used today (Sánchez-Sáez et al., 2020)(Sánchez-Sáez et al., 2021)(Carrick et al., 2021). When compared to other ML algorithms, RFs and their variations consistently perform well, not only for simulated SNe light curves (Richards et al., 2011)(Möller et al., 2016)(D'Isanto et al., 2016)(Ishida et al., 2018)(Lochner et al., 2016), but also for real ones (Bloom et al., 2012)(Pichara & Protopapas, 2013), and for simulated exoplanet transients as well (Mislis et al., 2015).

It makes sense that in earlier works where smaller real datasets were available RFs would outperform NNs, since it is known that this is the case for small data volumes (and the reverse is true for larger datasets). This would also be the case with more recent work that often considers small training sets that are meant to simulate the scarceness of real light curves with their types confirmed through spectroscopy. NNs require larger amounts of training data to reach peak performance.

Other classic approaches tried are Support Vector Machines (SVMs) (Mahabal et al., 2008)(Lochner et al., 2016)(Dhar Gupta et al., 2016)(Carrick et al., 2021) and some unsupervised approaches, such as K-nearest neighbours clustering algorithm (D'Isanto et al., 2016)(Lochner et al., 2016)(Carrick et al., 2021). Given that the amount of unlabeled data the new surveys are going to give us, perhaps more attention needs to be paid to unsupervised or semi-supervised approaches.

As time progressed and GPUs became more accessible, deep learning became popular in other disciplines and was eventually applied to light curve analysis and LCC. DL showed promise of being useful for the task, as we knew that more data was going to become available, and because it has the advantage of not relying on feature extraction.

One of the first articles (Charnock & Moss, 2017b) relevant to DL applied to SNe LCC used a bi-directional RNN to distinguish between different types of SNe. This work was not only aimed at the differentiation of SNe themselves, but also tried to gauge whether DL, thriving in other disciplines, could also benefit this kind of problem. It was, to my knowledge, the first time only a minimal amount of pre-processing was applied to the LCs, something that would be a definite advantage in the future and could save us a lot of computational time.

Since RNNs are supposed to work well with unevenly sampled sequences (although this is in the context of prediction tasks), they have not only been used for SNe classification (Möller & de Boissière, 2019), but also for variable star LCs (Naul et al., 2017) and different transient types (Muthukrishna et al., 2019). These works focus on GRUs specifically, perhaps because GRUs are known to do better than vanilla RNNs but are less expensive to train when compared to LSTMs.

CNNs started being used around the same time, in a variety of different ways. Since CNNs originally became popular to deal with images, some studies transformed LCs into images to then train CNNs with (Mahabal et al., 2017). We will see later in Chapter 3 that this approach is negatively impacted the more sparse the sampling on the LCs is. CNNs have also been used to detect signals in noisy TS (Pasquet et al., 2019), such as gravitational waves (Chan et al., 2020), and to classify SNe using inception modules (Brunel et al., 2019).

Although all these method work comparatively well, up to this point most of them have been tried on simulated LCs, with the exception of a few that used the Catalina Real Time Transient Survey (CRTS) (Mahabal et al., 2017) (Naul et al., 2017) or that tried their approach in only a handful of real LCs (Muthukrishna et al., 2019). This has changed in the last few years (Sánchez-Sáez et al., 2021) as ZTF (Bellm, 2014) allowed for real light curves to be used in bulk. The use of real light curves as test set for classification methods proved what had been predicted by many researchers: the differences between simulated training sets and real LC test sets has a huge negative impact on the performance of classifiers and so the bridging of that gap is crucial to keep up with labelling upcoming data. The creation and availability of a simulated data set for LSST LCs (PLAsTiCC) (Malz et al., 2019), which we will describe further in Section 3.1, allowed for more research dedicated to this particular topic.

In the last few years, more complex and interesting DL approaches have been tried, such as combined architectures. To focus on the sequentiality of LCs while at the same time giving similar importance to the whole of the light curve, an architecture that mixes a CNN with an LSTM was applied to the classification of variable stars (Bassi et al., 2021), in hopes that it would bypass the need of calculating a period.

A combination of a CNN and an LSTM was also used in the form of an ensemble to process both LCs and composite images at the same time. This work (Morgan

et al., 2021) aims to encapsulate both spatial and time related features by using these two types of NNs. Their goal is to be able to both identify and classify Lensed SNe, and they aim to do so by simulating images resembling various surveys (such as the Rubin LSST) and extracting simulated LCs from them.

Another ensemble approach considered unsupervised auto-encoders to learn feature representations, a module extracting class-based features by using a CNN and a Fully Connected NN to do de actual classification (Pasquet et al., 2019). Although this work uses simulated LCs only and considered just two types of events (SNe-Ia from everything else), it is I think particularly interesting because it focuses on what is perhaps the most pressing issue: the fact that data sets from different surveys are varied enough as to not be representative of one another. In other words: a data set from one survey cannot easily be used as training set for another survey without incurring in a significant performance penalty. Since we hope to use the labelled data we already have from past or ongoing surveys as training sets for the cascade of data that will come with the new generation telescopes, this is of major importance. In this article they find that their approach mitigates this discrepancy between simulated data sets, but the mismatch is more pronounced when the model is tested using real data, which can be reduced by using a portion of the real data in the training set.

Attention mechanisms (Pimentel et al., 2022)(Morvan et al., 2022)Allam & McEwen (2021)(Ibsen & Mann, 2020) are another type of architecture that is becoming more popular, as it has happened in other fields, we will look into this further in Chapters 4 and 5.

Although there has been significant advances, there are still many issues that need to be dealt with and much room for improvement. Not only is it important that more effort is dedicated to ensuring that the training data we use is more representative of the data that we will get in the future, but it will also be important to pay attention to the reconstruction and standardization of LC data sets and their accessibility.

1.4 Outline of the thesis

In this Chapter we established the context of the work that is to follow by stating the problem and its challenges, describing some necessary concepts and reviewing

relevant literature.

In the following Chapters, we will continue our journey into this field of research: In Chapter 2 we will describe the experimental set-up used in the rest of this thesis; in Chapter 3 we will study the PLAsTiCC dataset and how the challenges it presents are representative of those found in real life scenarios; in Chapter 4 we will compare NN architectures commonly used in TSC and see how they compare to an Architecture that uses Attention, and how this mechanism can be useful in early light-curve classification; in Chapter 5 we will look further into Attention, Multi-headed Attention, the Transformer, and adapt the latter to a Variational Auto-Encoder version and see how, by framing an NN as a probabilistic model and adding a regular latent space, the issue of having two different datasets can be mitigated; and finally, in Chapter 6 we will summarize the conclusions obtained from all this work.

I hope, dear reader, that you find the work that I am about to present to you to be interesting and engaging. I certainly did my best to make it so, both for you and myself.

Chapter 2

Experimental framework

2.1 Motivation

As I am sure any researcher working in machine learning will know, trying out different models and tuning them takes time and can easily get messy. The same model is usually tested not only with different hyper-parameters, but also with the same ones several times to account for the effects that random initialization might introduce to the final performance metrics. In addition to this, each experiment run should ideally be reproducible, in case they need to be checked or built upon. All of this means that some care needs to be taken when designing an experimental set-up so hopefully few resources are wasted and all results are kept organized.

While working on this thesis, I found myself building a framework for running experiments out of necessity, which is now publicly available[1]. In this Chapter, I will first briefly describe its structure, I will then describe the experimental set-up and the metrics used throughout this thesis and finally, I will list the most important tools used.

2.2 Description

The framework I built makes use of several other popular machine learning frameworks and tools (listed in Section 2.6) to create a piece of software that runs

[1] https://github.com/aibsen/phd

and organizes experiments while ensuring that the results are not tremendously influenced by the random elements that define a single run. It also makes sure that the experiments are reproducible. This framework is meant to be run on a GPU or collection of GPUs.

The experiments are organized in three different classes: `SeededExperiment`, `CVExperiment` and `Experiment`.

`SeededExperiment` receives the hyper-parameters of both the model and the experiment, sets a random seed generator, and passes the parameters to `CVExperiment`. If a list of `N` random seeds are given (or if `SeededExperiment` is told to create `N` random seeds), then the process is repeated `N` times. A directory with the name of the experiment is created, and within it one directory for each different seed. For every seed directory, there are two folders: `folds` and `result_outputs`.

`CVExperiment` receives the parameters and splits the data into `K` folds, where each fold will be used as validation set once. This is essentially cross-validation. `CVExperiment` calls `Experiment` `K` times and each time a new instance of `Experiment` receives the parameters and starts a learning process. For each fold there is a directory `folds_K`, and within it there are three folders: `folds`, `result_outputs` and `saved_models`. Figure 2.1 shows an example of how the directories are organized.

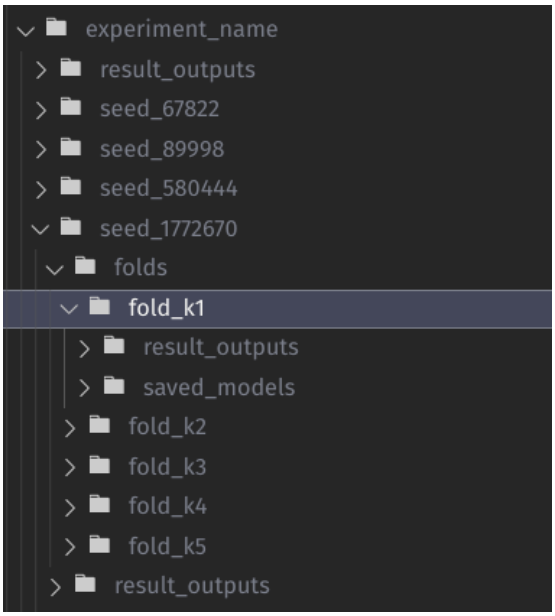


Figure 2.1 Example of how the outputs of an experiment are organized in a directory.

The class `Experiment` has four main methods: `run_experiment`, `run_train_phase`, `run_final_train_phase`, and `run_test_phase`. While these methods are defined, `Experiment` can also be extended into specific type of experiments and override them. In this thesis there are two types of experiments: `ClassificationExperiment` and `Seq2SeqExperiment` (sequence to sequence).

As a side note, this framework uses the OOP (object oriented programming) principle of Composition over inheritance (Gamma et al., 1995). This means that, in order to achieve a polymorphic behaviour and reuse code, classes extend methods and store instances of other classes without there being a strict hierarchy of parent classes. This is useful because, while I built this with the aim of running experiments with different seeds where each time cross-validation would be used, A single experiment can be run on its own, or a cross-validation experiment can be run without there being several seeds, or several seeds can be used to run experiments without cross-validation.

2.3 Running an experiment

For each seed and for each k-fold, the learning process starts by calling `run_experiment`. This consists of three phases: the training phase, the final training phase and the test phase.

During the training phase a previously-specified number of epochs `e` is used as an initial budget and the model starts learning by looking at the data. Every `s` number of epochs, the model is tested against the validation set. This is given as an hyper-parameter called step. The scores of every epoch are stored in an array for later.

During each validation iteration, the scores of the performance of the model over the validation set are retained in cache, as well as a snapshot of the model itself. Each consecutive time the validation set is seen by the model, a new score for a specific metric is compared to the previous one. If the new score is better than the best one so far, a snapshot of the model and the score in cache are replaced. If the new score is worse than the best one, `Experiment` counts one strike. After `p` strikes the training stops.

The scores obtained for all metrics for all epochs are saved, along the model itself, at the state at which it was during the best validation iteration. In the case of

a classification experiment, the predictions for each sample in the validation set are also saved.

The method described above is called *early stopping* (Orr & Müller, 2003) and it has two purposes: the first one is to save computational time. This might not be an issue for people with access to a cluster, but it is certainly an issue for people with one or two GPUs, especially if they are shared. The second purpose is to stop the model before it starts to over-fit the training set and performance worsens. Rather than having to probe the number of epochs like one would probe any other hyper-parameter, `Experiment` stops after it realizes that it has been a while since the model has learnt anything useful. It is possible that if the learning process was too unstable, it might have been stopped while the model could still learn, which is why some thought has to be given as to how patient one is going to be with the model. If the model and hyper-parameters make sense, however, the process should not be unstable. The validation step `s`, the patience `p` and the initial budget of epochs `e` are all hyper-parameters given by the user.

During the final training phase the model parameters are initialized again and the learning process starts anew. This time there is no validation and we use the average number of epochs at which the model stopped learning before. The idea of having this phase is to use the entirety of the training set, instead of only a fraction of it. While this is run only once, instead of `K` times, all `K+1` runs are considered when calculating the final test scores.

During the test phase, the test dataset is passed through all `K+1` models saved in the previous phases. The results and scores obtained for each model are saved, along with a final summary that contains of the average scores over all saved models.

2.4 Experimental set-up

In this thesis, I often present results over several test sets, which are the mean of several runs using several seeds and k-folds. Unless stated otherwise, the hyper-parameters used and why they were chosen as default are described in Table 2.1. For classification experiments, whenever a confusion matrix is presented, it is usually calculated using the predictions given in the best run of the experiment. The learning algorithm used is usually Adam (Kingma & Ba, 2014).

Hyper Parameter	Value	Reason
Learning rate	10^{-3}	It is a common starting value, since it would not make sense to start an experiment with a much lower value and spend computational time if not sure whether it will be worth it. A higher value usually means that a model will be prone to running into local minima when minimizing the loss function.
Weight decay coefficient	10^{-2}	Chosen to be larger than the usual default value to prevent over-fitting.
Batch size	64	Small enough to allow for more shuffling of the samples without incurring in too much performance overhead.
Epochs	100	Large initial budget.
Validation Step	3	Chosen so we frequently check that the network is not over-fitting over the training set without adding too much training time.
Patience	5	Chosen so no more than 15 epochs will go by without the network actually learning.
K-folds	5	Allows for several divisions over the training data without diminishing the amount of training samples too much.
Number of seeds	5	The results presented will be over 5 seeds and 5 k-folds, plus the final run. This means that for each experiment, the results are the mean of 30 individual runs.

Table 2.1 Default values for hyper-parameters for the experiments.

2.5 Performance metrics used

In this thesis there are two types of experiments: classification experiments (in Chapters 3, 4 and 5) and sequence-to-sequence experiments (in Chapter 5). The loss function that we are going to be minimizing is specified in each Chapter or Section. However, for classification experiments it is usually cross-entropy, as described previously in Section 1.2.

Aside from loss, other metrics used in classification experiments are accuracy and F1-score. Accuracy refers to the total number of correct predictions over the total number of samples. While this metric is useful to see what portion of a dataset got classified correctly, it is not enough to gauge the performance of a model if the

dataset we are using as test set is unbalanced, as it is the case in all experiments in this thesis, since a model could be labeling everything as the same class and still get a high accuracy score.

The main metric considered for classification experiments is the F1-score, which takes into account both Precision and Recall.

Precision is a measure of the purity of classification, that is, the number of samples that got classified as a certain class that actually belong to that class. For example, the number of light-curves marked as SN-Ia that are actually SN-Ia and not something else, which is important, since they are used as standardizable candles.

Recall, on the other hand, is a measure of sensitivity. It considers how many items of a certain class were actually marked as that class. For example, how many SLSN were labeled as such and not something else, which is important for that type of rare object.

The F1-score, which we will be focusing on, is the harmonic mean of Precision and Recall. The harmonic mean is mathematically the reciprocal of the arithmetic mean of the reciprocals. It is commonly used to get a sense of how more than one measure contribute to a certain output. In the case of precision and recall:

$$\text{F-1} = \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.1}$$

In this thesis, I calculated the F1-scores, Precision and Recall separately for each class and then averaged them to obtain the final score. The F1-scores will not usually be as high as other metrics, since it is often the case that when Precision is high, Recall tends to be lower, and vice versa. Also, since I am taking the mean of the F1-scores for each class, missing one class will mean that the F1-score will go down significantly. Regardless I decided to use this metric because of the nature of the problem: we want to both to obtain as many correctly labeled samples of a class as possible without too much contamination. Given that some SN, like SN-Ia, appear much more often in datasets than other classes, I felt it was importance to penalize scores whenever any one class was missed.

Aside from these metrics, I often present confusion matrices to illustrate the performance of a model. In a confusion matrix each row represents a true class and each column represents a predicted class. Samples are then binned in this

matrix according to the label they were given by a classifier versus the label that actually corresponded to them. Thus, when a classification experiment goes well, the diagonal in the matrix will be marked. This visualization often allows for more interpretation of the scores, as it lets us see where the models made the most mistakes. An example (taken from the validation results of one of the experiments in Chapter 5)is given in Figure 2.2.

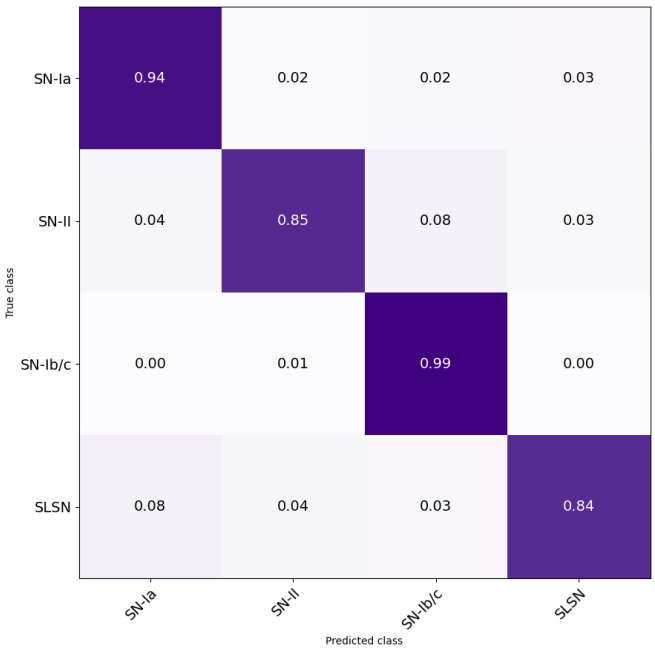


Figure 2.2 Example of a normalized confusion matrix.

2.6 Tools used

The main tools used throughout this thesis to build models, store statistics and plot figures are presented in Table 2.2. I thought it was important to at least list them, since when I myself looked into the work of other people, I often wondered about their implementations and what tools they used, which was not always specified. All the code used was written in Python version 3.9.12.

Tool	Version	What for
astropy	5.1	General functionalities to deal with astronomical file formats
cudatoolkit	11.3.1	To interface with GPUs
h5py	3.2.1	To store compressed data
matplotlib	3.5.2	Plotting library
numpy	1.21.2	Package for scientific computing
pandas	1.3.4	Library for data analysis
pytorch	1.10.0	Library used for all deep learning functionalities
scikit-learn	1.0.1	Used to calculate metrics such as confusion matrices
scipy	1.7.3	Package for scientific computing
sfdmap	0.1.1	For manipulating dust map FITS files
simsurvey	0.7.5	For running survet simulations
sncosmo	2.8.0	For obtainig SNe templates
torchmetrics	0.8.2	For calculating metrics on GPU during experiments.

Table 2.2 Main tools used for the Experimental framework and the versions I used at the time.

Chapter 3

PLAsTiCC Challenge

3.1 Introduction

In order to prepare the scientific community for the new data-driven astronomical era, the PLAsTiCC (Photometric LSST Astronomical Time-Series Classification Challenge) (Allam Jr et al., 2018) competition was launched in Autum 2018 at Kaggle [1], an online community dedicated to the analysis of data. Participants were asked to classify simulated light curves modeled to resemble the multi-variate time series that we expect to get from the photometric observations of the LSST at the Vera C. Rubin observatory (Ivezić et al., 2019b).

The objective of the challenge was to categorize astronomical objects into fifteen classes, each representing a different type of transient. The data set was given to participants divided into training data and test data, where labels were initially provided for the training data only. Although the challenge has already ended, the problem of light curve classification remains an open and relevant one, so the labels for the whole of the data set were later (January 2019) released to the public (Hložek et al., 2020).

The PLAsTiCC challenge is relevant to my work because it provides a big enough data set to experiment with. Whatever experiment is devised needs to take into account that as little time as possible should be spent in the data pre-processing steps during classification time, but that time constraints during training are less restrictive. It is also important to note that whatever classifier is implemented

[1] https://www.kaggle.com/c/PLAsTiCC-2018

in the future, if it were to be trained using the PLAsTiCC data set, there will have to be some transfer learning involved: in other words, it will be necessary to train the system with one data set in such a way that it would still work when giving it data from another data set.

This Chapter is structured as follows: I first describe the PLAsTiCC data set in more detail, outlining its peculiarities and how they contribute to make the task at hand challenging. I then describe my initial attempt at the task, explaining the reasoning that took me to make certain decisions. After analysing the results obtained, I go on to examine the top three winning solutions and other relevant literature, with the aim of finding important improvements and lessons that might help in my future work.

3.2 PLAsTiCC Data Set Characteristics

The PLAsTiCC data set (Allam Jr et al., 2018) is composed of a training set and a test set that differ in many characteristics. The training set is made up of 7,846 labeled astronomical objects, while the test set comprises almost 3.5 million unlabeled objects, as can be seen in Table 3.1 and in Figure 3.1.

The training data set is meant to represent the nearby, low-redshift, brighter astronomical objects for which obtaining expensive spectroscopy is possible. The test data set represents objects that are distant, have higher redshift, appear fainter, and cannot be spectroscopically confirmed, so only photometry is available for most of them.

While the training set is composed of 14 different classes, the test set has 15: there is one extra class, meant to encode objects that we might discover. Contrary to intuition, the labels of the classes do not correspond to numbers from 1 to 14, but are the numbers displayed in Table 3.1 instead. The 'mystery' class 99 (the class that does not appear in the training set) was later revealed to be four different kind of objects (Kessler et al., 2019).

In Figure 3.1 we can also visualize one of the main difficulties in this challenge and in LCC in general: the data set is unbalanced. While there is a rough correspondence in this case regarding the proportion of different types between the training and the test set, there are clear differences as well. One of the main ones is that SNe-Ia appear way more often when compared to other classes (except

SN-II) in the test set than in the training set. In Figure 3.2 we can see how the number of objects per class compares to the total number of objects for both the training and the test set, and whether these ratios are within the same order of magnitude.

Class	Object represented	n training set	n test set
90	Supernova Type Ia	2,313	1,659,831
67	Supernova Type Ia-91bg	208	40,193
52	Supernova Type Iax	183	63,664
42	Supernova Type II	1,193	1,000,150
62	Supernova Type Ibc	484	175,094
95	Super Luminous SN	175	35,782
15	Tidal Disruption Event	495	13,555
64	Kilonova	100	131
88	Active Galactic Nuclei	370	101,424
92	RR lyrae	239	197,155
65	M-dwarf stellar flare	981	93,494
16	Eclipsig Binary stars	924	96,572
53	Pulsating variable stars	30	1,453
6	μLens-Single	151	1,303
991	μLens-Binary	0	533
992	Intermed. Lum. Optical Transient	0	1,702
993	Calcium Rich Transient	0	9,680
994	Pair Instability SN	0	1,172
	Total objects	7,846	3,492,888

Table 3.1 Class numbers as shown in the training set CSV files, astronomical objects they represent and number of times a particular type appears in the training set. The rows in light blue are classes that represent galactic models

The PLAsTiCC data set also considers two different survey regions: Deep Drilling Fields (DDF) and Wide Fast Deep (WFD). DDFs are small parts of the sky that are sampled often, so the objects coming from DDFs have lower flux error. Objects coming from WFD, on the other hand, have greater uncertainties, because this observation mode covers a lot more of the sky much less frequently. These two simulated survey modes are aligned with how the LSST will operate. The training set is not representative of the test set regarding the proportion of objects coming from DDF/WFD. As can be seen in Figure 3.2, while more than a quarter of the training set belongs to the DDF category, less than 1% of the test set does so.

Class	samples/total$_{train}$	samples/total$_{test}$	Same order
Supernova Type Ia	2.9×10^{-1}	4.7×10^{-1}	✓
Supernova Type Ia-91bg	2.6×10^{-2}	1.2×10^{-2}	✓
Supernova Type Iax	2.3×10^{-2}	1.8×10^{-2}	✓
Supernova Type II	1.5×10^{-1}	2.9×10^{-1}	✓
Supernova Type Ibc	6.2×10^{-2}	5.0×10^{-2}	✓
Super Luminous SN	2.2×10^{-2}	1.0×10^{-2}	✓
Tidal Disruption Event	6.3×10^{-2}	4.0×10^{-3}	×
Kilonova	1.3×10^{-2}	3.8×10^{-5}	×
Active Galactic Nuclei	4.7×10^{-2}	2.9×10^{-2}	✓
RR lyrae	3.0×10^{-2}	5.6×10^{-2}	✓
M-dwarf stellar flare	1.3×10^{-1}	2.7×10^{-2}	×
Eclipsig Binary stars	1.2×10^{-1}	2.7×10^{-2}	×
Pulsating variable stars	4.0×10^{-3}	4.2×10^{-4}	×
μLens-Single	1.9×10^{-2}	3.7×10^{-4}	×
99	0	3.7×10^{-3}	×
total objects	7,846	3,492,888	

Table 3.2 ratios of number of objects per class and total objects in train set and in test set and whether these ratios are the same order of magnitude or not

Aside from the light curves themselves and the survey mode information, there are other pieces of information given in the data set for each object: Milky Way light extinction due to dust and redshift.

Milky Way Extinction (MWEBV for MW E(B-V)) is dependent on the celestial coordinates of the object and represents the reddening of the light due to dust along the line of sight. It is also passband-dependent, since the absorption of light due to dust in our galaxy is stronger in the UV (u-filter) and weaker in the infrared filters (izy). The light curves simulated are corrected considering this value. In Figure 3.3 we can see a histogram showing the distribution of MWEBV for objects in the training set (in blue) and in the test set (in orange) for each type of object. MWEBV tends to be more evenly distributed in the test set than in the training set, where values seem to cluster near 0, with some notable exceptions. Both histograms are normalized for visualization purposes. It makes sense that the test set would have a more even distribution, if nothing else because of the sheer amount of objects considered. Aside from that, the training set represents data that comes from fewer fields that were followed up spectroscopically, while the test set is simulated to portray data drawn from a wider survey region. The mean and standard deviation values of MWEBV where

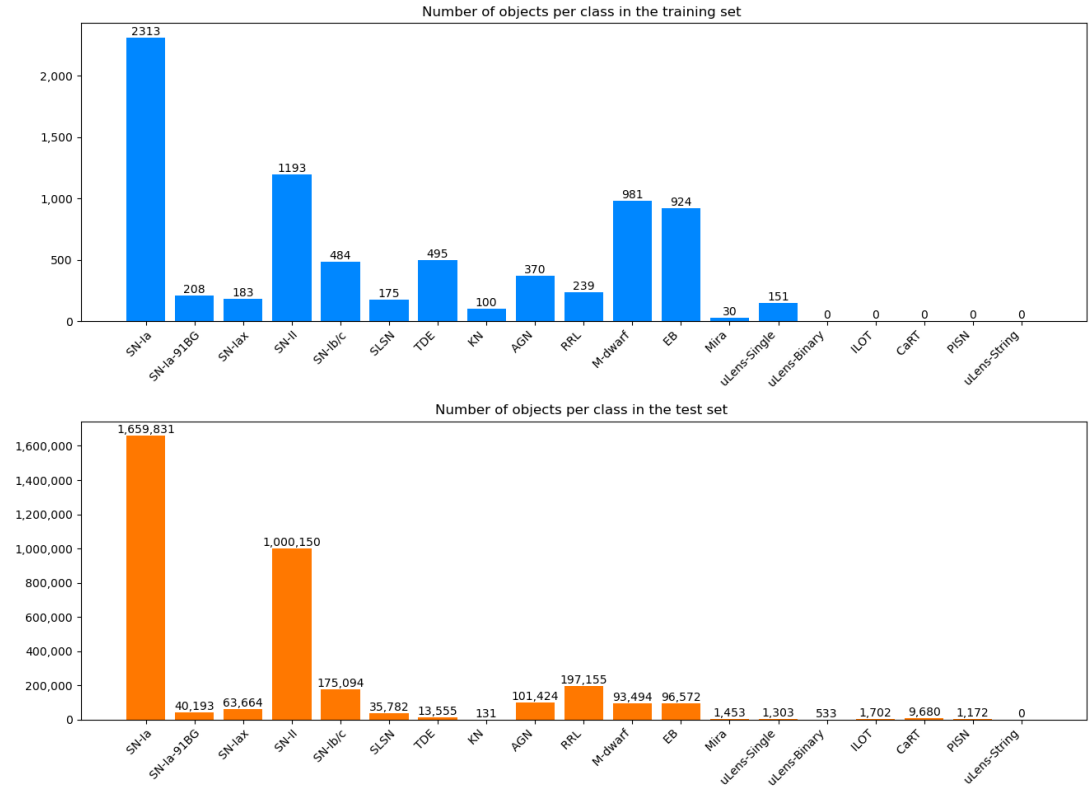


Figure 3.1 Number of objects per class in the PLAsTiCC training set (blue) and test set (orange)

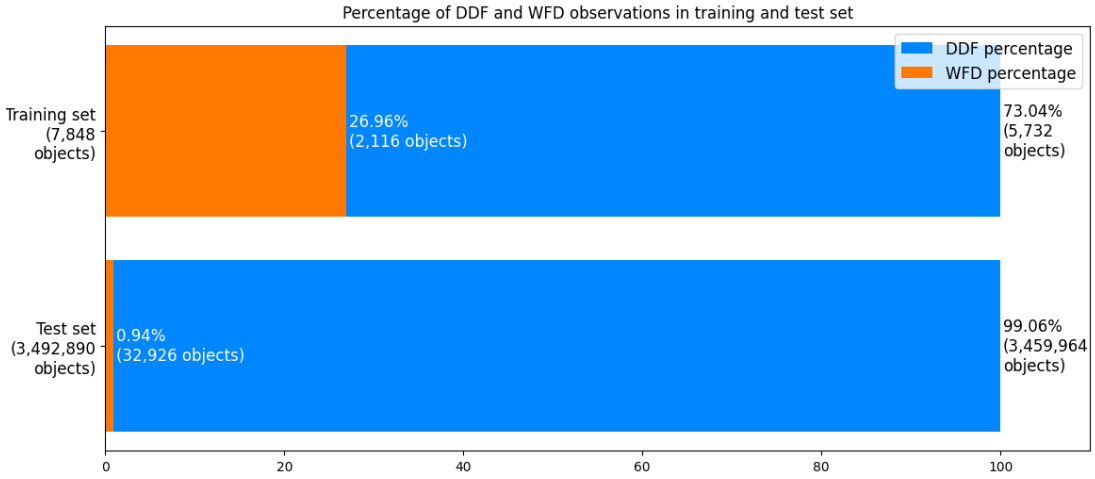


Figure 3.2 Percentage of simulated objects coming from DDF (orange) and coming from WFD (blue) in the training set (bar above) and test set (bar bellow)

also similar for the training set and the test set ($\mu = 0.082$ and $\sigma = 0.150$ for the training set and $\mu = 0.091$ and $\sigma = 0.154$ for the test set).

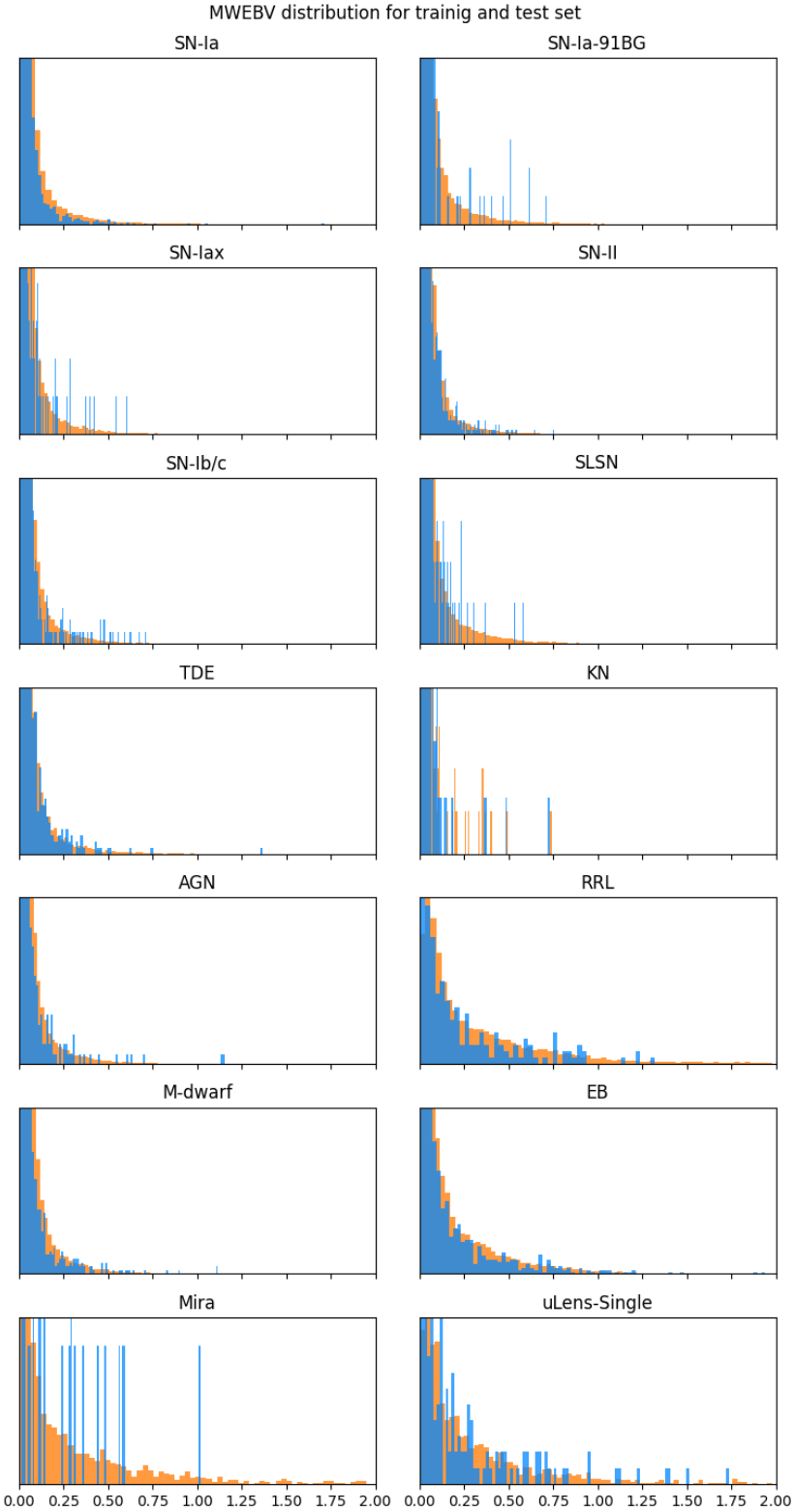


Figure 3.3 Distribution of Milky Way dust extinction for objects in the traing set (blue) and in the test set (orange)

Redshift for each object is given in three values: spectroscopic redshift (sz) of the host galaxy, photometric redshift (phz) of the host galaxy, and photometric redshift uncertainty. All objects in the training set have known spectroscopic and photometric redshift. For the test set, however, only a few objects have spectroscopic redshift available. Additionally, in the data set description given to participants, it was remarked that some photometric redshift measurements for the test set were catastrophic: widely inaccurate. Although it is not exactly specified, it seems to be the convention that an error in phz estimation is considered 'catastrophic' when $|sz - phz|/(1 + sz) > 0.15$.

Figure 3.4 shows how host galaxy photometric redshift compares to spectroscopic redshift in the training set and in the test set. For the test set, only the few $(120, 393)$ objects that had sz available are considered. We see that the objects are distributed more or less the same. For the test set, all objects that have $phz > \approx 1.5$ are catastrophic outliers. The percentage of catastrophic outliers η in both cases is similar, although it is actually a little bit higher in the case of the training set.

Each object in the data set is associated with six light curves and each of them represents the flux associated with a different passband. The passbands correspond to: u ($\lambda \in [300-400]$nm), g ($\lambda \in [400-600]$nm), r ($\lambda \in [500-700]$nm), i ($\lambda \in [650-850]$nm), z ($\lambda \in [800-950]$nm) and y ($\lambda \in [950-1050]$nm). Figure 3.5 is an example of what these light curves look like.

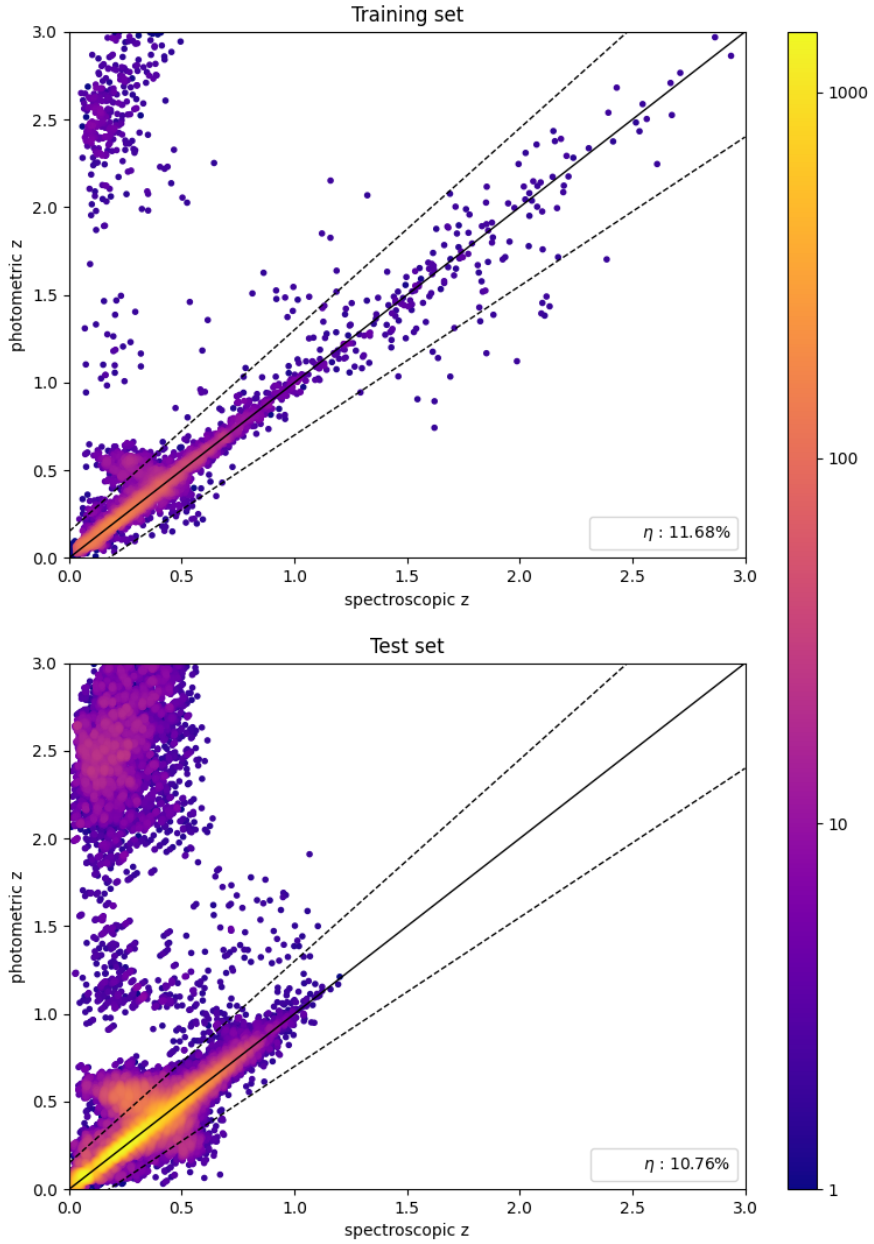


Figure 3.4 Spectroscopic and photometric redshift for the extra-galactic objects present in the training set and test set. Colours represent object density in logarithmic scale. The solid diagonal shows where $sz = phz$. The dashed lines represent the threshold where an error is considered 'catastrophic'. η represents the percentage of outliers of the objects considered.

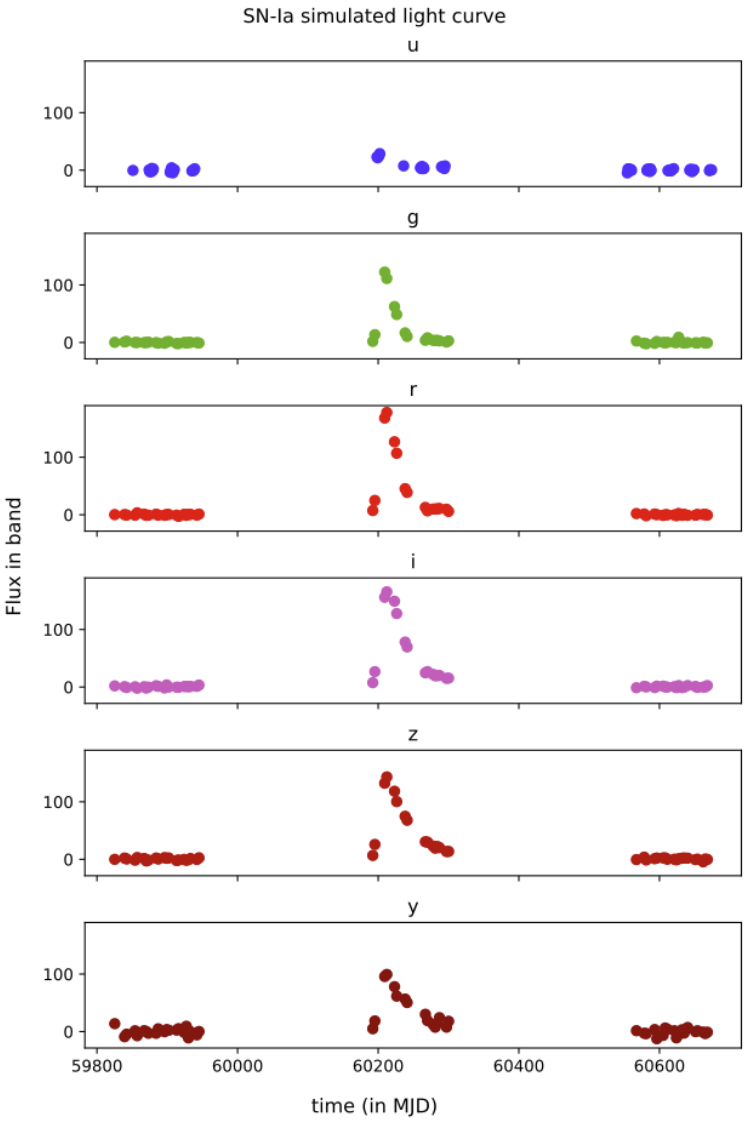


Figure 3.5 Example of astronomical object and its light curves. This object is a simulated SN-Ia in the training set.

3.3 Attempt at the PLAsTiCC challenge

While the competition was still ongoing, I decided to try an approach that had been recently proposed and that consists of mapping the light curves into two dimensional histograms (referred to as DMDTs) (Mahabal et al., 2017) and then running those images through a CNN (Lecun & Bengio, 1995).

I decided to test the DMDT method on the PLAsTiCC dataset for two reasons. The first one is that, because of the way the histograms are constructed, it seemed to me that the technique would help homogenize the differences between observations from different surveys, thus potentially making transferable learning possible.

The second reason was a practical one: before the challenge even started, at the very beginning of my PhD, I used transient light curves from the Catalina Real-Time Transient Survey (Djorgovski et al. (2011b)) data set to test different classification methods. While doing so, I initially used the FATS library, a tool designed for time series feature extraction, and specifically for astronomical light curves (Nun et al., 2015). I tried combinations of these features and compared with the ones most often used in the literature, but did not achieve great results.

It became apparent that feature extraction and feature selection was indeed not straight forward. I then tried the DMDT approach for feature extraction only and the classification results markedly improved, at least for some machine learning algorithms, such as RFs and their variations. Up to that point, however, I had not tried any DL classification models. Given that the DMDT approach was proposed to be used with CNNs and that I had already seen that it worked well with ML algorithms that establish a hierarchy of features, I decided to try it for the PLAsTiCC dataset. It must be said, however, that the class divisions in the PLAsTiCC data set are very different from the ones I had available with the CRTS data set. The CRTS taxonomy did not distinguish, for example, between different types of SNe, while PLAsTiCC focuses heavily on this distinction.

Other reasons for trying out a CNN as classifier were: they are known to perform well when classifying grid-like data (like the DMDT images) and they take a relatively short time to train (when compared with other approaches such as RNNs, Random Forest, etc). That said, there are several different architectures a CNN might have (a CNN is strictly defined as a Neural Network that uses a

convolution in some of its layers) and they can give very different results.

In order to try the PLAsTiCC challenge with this approach, a number of decisions had to be made regarding the pre-processing of the data and the design of the CNN to be used, which required running some experiments. Since the test set is not labeled at this point, only the training set was used for these experiments. The training set itself was divided into three (training, validation and test sets) using cross-validation with five folds (the experimental set-up used throught this thesis is described in more detail in Chapter 2.

It is important to note that although the PLAsTiCC challenge defined a metric by which the different classification results would be gauged, participants could not compute this metric without uploading the entire test set classification results to Kaggle, since the labels were not yet public. This metric, *weighted multi-class loss*, considers different weights for each class, that represent how important it is to classify that particular class. During the competition, these weights were also not disclosed. The weighted multi-class loss is very similar to the cross-entropy loss described in 1.2 except for these weights and is defined specifically as:

$$L = \frac{\sum_{j=1}^{M} w_j \cdot \sum_{i=1}^{N} \frac{1}{N_j} \tau_{i,j} ln(P_{ij})}{\sum_{j=1}^{M} w_j}. \tag{3.1}$$

Where $P_{i,j}$ is the probability of the object i belonging to class j, N_j is the number of objects in a given class, M is the number of classes, $\tau_{i,j}$ is 1 if the i^{th} object comes from the j^{th} class and 0 otherwise, and w_j are weights per class that reflect how desirable it is to have objects of the class j classified correctly.

3.3.1 Pre-processing

The DMDT approach I chose to try involves a fair amount of data pre-processing. Not only is it necessary to construct the DMDT images, but it is also important to manipulate the data set as to make it more similar to the test set, and enlarge it, given the small amount of training samples.

3.3.1.1 DMDT construction

Let $L = [l_1, \ldots, l_P]$ be an astronomical object in our photometric data set, where P is the number of passbands, $l_p = [(m_0, t_0), \ldots (m_{n_p}, t_{n_p})]$ is a light curve of length n_p and each point in the light curve as a magnitude m and a time t coordinate. To map L into a two dimensional histogram, for each pair of points in each of its light curves we need to compute the differences in brightness (DM) and time (DT). This yields $N = \binom{n}{2}$ points, for a light curve of length n. These differences are then binned into predefined ranges bm and bt, so we get P histograms per object. These histograms are normalized and stacked into a tensor, which is the final DMDT image. The process is also described in Algorithm 4).

Algorithm 4 *DMDT Mapping*

Input: L, bm, bt

DMDT$\leftarrow$ []
for $p \in 1$ to P **do**
    $m \leftarrow [m_0, \ldots, m_{n_p}] \in l_p$
    $t \leftarrow [t_0, \ldots, t_{n_p}] \in l_p$
    $dm \leftarrow [(y - x)$ **for** $x, y \in$ combinations$(m)]$
    $dt \leftarrow [(y - x)$ **for** $x, y \in$ combinations$(t)]$
    $dmdt \leftarrow$ histogram2D(dm, dt, bm, bt)
    DMDT.append$(dmdt)$
end for
return DMDT

The construction algorithm just described requires binning ranges for both magnitude and time points. It is then necessary to decide two things: how these ranges will be distributed and how many of them we will have, that is, what will be the resolution of our DMDT images.

3.3.1.2 DMDT binning strategies and resolution

To define a binning strategy, it is necessary to run a few initial experiments and see what works best. Since it is not clear yet what CNN architecture will be used to classify the test set, I started by trying two different architectures described by Mahabal et al. (2017). One of them represents a very shallow CNN (with only one convolutional layer that outputs 32 filters and two dense layers), while the second one has many more parameters (three convolutional layers with many more filters as well as two dense layers).

For the DMDT images to be useful, they need to represent the changes occurring in the light curves of the astronomical objects they belong to in a way that highlights what is relevant. To figure out what DMDT resolution would be useful, I first defined binning ranges in logarithmic scale for both the flux and the time dimension, as shown in Figure 3.6. Since most of the objects in the PLAsTiCC dataset are explosive transients, it makes sense to use a logarithmic scale (as Mahabal et al. (2017) also described). The idea is that changes in magnitude that occur in short periods of time and are thus relevant features for rapid transients would be well represented by being distributed in many bins, while magnitude changes that occur over a longer periods of time would be clumped together. In retrospective, while this is useful for distinguishing between explosive objects and other type of objects, it is probably not the best decision for telling apart different kinds of SNe, because large magnitude changes that occur in small periods of time, which could have been useful features, are also combined. The binning ranges go up to the highest DM and DT values in the dataset. I tried different resolutions on both CNN architectures mentioned. Results can be seen in Table 3.3.

We can see in Table 3.3 that for the shallow CNN, the F1-score decreases with image resolution. This is perhaps because the shallow CNN does not have a large number of parameters (weights) and so a higher resolution might cause it to learn less important details about the image at the cost of missing more important aspects. The deeper CNN, on the other hand, benefits from having a higher resolution, perhaps because it has more parameters. However, the F1-score decreases also at the highest resolution tried (48×48), suggesting that adding more details to the images does not prove beneficial. We also see that a better accuracy does not necessarily mean a better F1-score, since it is possible to obtain a higher accuracy by just correctly detecting a significant portion of a prominent class.

Overall, performance does not seem to change all that much with image resolution, but run-time for the experiments increases significantly, especially for the deep CNN. For this reason, for the following experiments we will only consider the lower resolutions: 24×24 and 32×32.

I then also tried slightly different binning strategies. While I maintained the logarithmic spacing of the bins, I did change where the logarithmic scales start and end; any value that falls outside of the defined bins would then fall inside of the first or last bin.

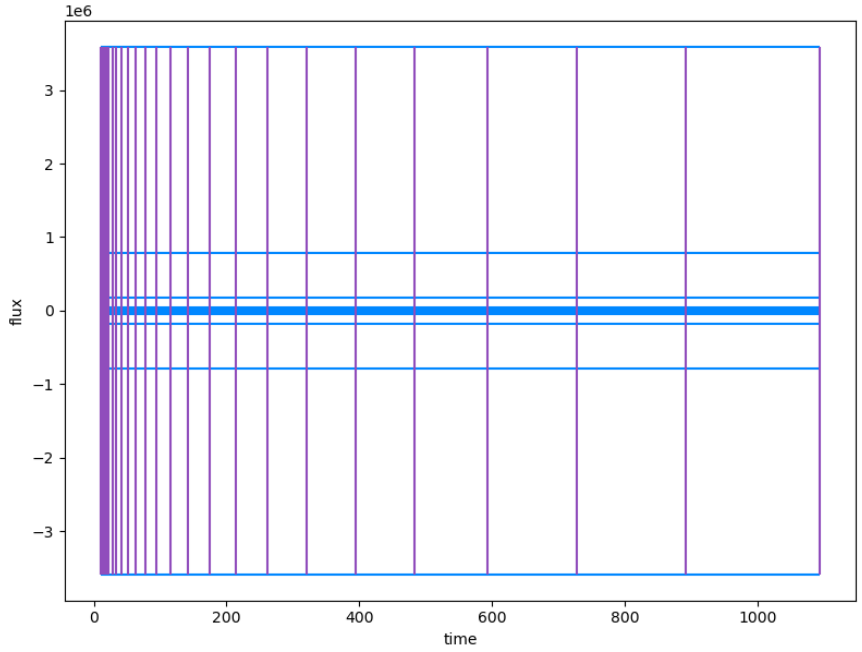


Figure 3.6 Binning ranges for the construction of the images. Vertical lines represent the time binning ranges while horizontal lines represent the binning ranges for flux differences.

	Shallow CNN			Deep CNN		
Resolution	F1-Score	Accuracy	Time(s)	F1-Score	Accuracy	Time(s)
24×24	0.494	0.585	41.5	0.498	0.644	147.9
32×32	0.491	0.584	63.9	0.515	0.643	347.7
40×40	0.479	0.587	73.6	0.523	0.656	653.0
48×48	0.472	0.574	112.6	0.507	0.650	705.3

Table 3.3 Resolutions of the constructed images and how that impacted the performance and training time of two CNN architectures. The F1-scores, accuracy scores and training times represent the mean of these values over 5 different runs. Higher (better) F1-scores and accuracy scores are presented in darker shades (light blue hue). Higher image resolutions are also presented in darker shades (orange hue)

In the time dimension, the first binning range starts from 1 day of observations. I did this because I thought it made sense to cluster changes in shorter periods of time, since they could amount to noise and would not need to be well represented. I also considered that the final binning range for DTs could be the upper quartile, since changes in longer time may not be as important, and since shortening the binning ranges at the tail would mean that first binning ranges would be more

widely spaced.

In the flux dimension, I considered two different binning strategies: having the first bin start from 1, and having it start from 10. The idea is that flux changes smaller than 10 would perhaps be noise and not encapsulate that much meaning. Results from these experiments are presented in Table 3.4.

Binning Strategy		Shallow CNN		Deep CNN	
DM	DT	24×24	32×32	24×24	32×32
0: ±[1-highest value]	[1-highest value]	0.494	0.491	0.498	0.5152
1: ±[1-highest value]	[1-75 percentile]	0.490	0.488	0.494	0.5281
2: ±[10-highest value]	[1-highest value]	0.501	0.482	0.517	0.5166
3: ±[10-highest value]	[1-75 percentile]	0.499	0.485	0.519	0.5218

Table 3.4 Different binning strategies for both flux and time differences and how they impacted the performance (F1-Scores) of two CNN architectures, when considering two different image resolutions. The F1-scores represent the mean of these values over 5 different runs. Best F1-scores are highlighted in blue.

We can see in Table 3.4 that when the CNNs are trained with images of 24×24 bins, they perform better when flux changes smaller than 10 are clustered together, while for images of resolution 32×32 this is not the case. We also see that the deep CNN seems to benefit from having a smaller time range considered in the bins, while the opposite is true for the shallow CNN. The differences however does not seem to be significant. Given that the objects in the test set are supposed to be farther away and noisier, I will be considering the binning strategy number 2, which does not represent well really small changes and considers longer time periods.

It is immediately obvious that the DMDT method is not giving us great results, as the F1-scores we have obtained so far are not great (The highest possible F1-score is 1, while the lowest is 0, so these scores are just around the midpoint). However, these performance metrics are averages of the performance metric for each individual class. This means that although the overall metric might be low, perhaps some classes are correctly classified while others are consistently missed and severely impact the metric.

We can inspect the confusion matrix in Figure 3.8a to see that indeed this is the case. Most of the confusion seems to be around different types of SNe. The miss-

classified samples are mostly predicted to be SNe-Ia (the most frequent class in the data set), though some SNe (Ib/c and SLSNe) are incorrectly labeled as SNe-II (the second most frequent class). Kilonovae are confused with M-dwarf stellar flares, which is the third most prominent class. It is clear that how unbalanced the data set is has a significant impact, since we also see that the classes that are missed the most are those with fewer samples. Perhaps performance could be improved by adding more of these under-represented objects to the data set, as well as by increasing the overall data volume.

3.3.1.3 Data augmentation

Hoping to improve the performance of the models, I chose to increase the data volume by adding light curves constructed using the original objects as templates. Essentially, three different operations were applied: shift in time, noise and degradation. The aim of this is not only to enlarge the training data set, but also to add light curves that are closer to those in the test set.

The original light curves were shifted in time by a random amount, but the cadence of them is maintained. The noise added to the flux of the light curves is proportional to the flux error given for each point. Additionally, the objects were degraded by dropping 20% of the points per light curve, whenever the total number of points permitted it. Care was taken so no passband was left with no points.

Although initially I considered balancing the data set, I eventually decided against it, since the test set is also unbalanced in a similar way to the training set. I did, however, make some changes to the class imbalance in order to make it is less dramatic and hopefully teach the models to identify a bigger fraction of the less frequent objects. The idea was to add more samples of every class, but add significantly more samples for classes that were either infrequent or consistently missed while somewhat maintaining the class imbalance. The number of samples per class of the new augmented training set and how it compares to the original training set can be seen in Figure 3.7.

I then run experiments to see whether data augmentation improves performance. Results for the shallow architecture can be seen in Table 3.5, while results for the deep architecture are presented in Table 3.6. We can see that for the shallow architecture, F1-sores significantly improve with the augmented data

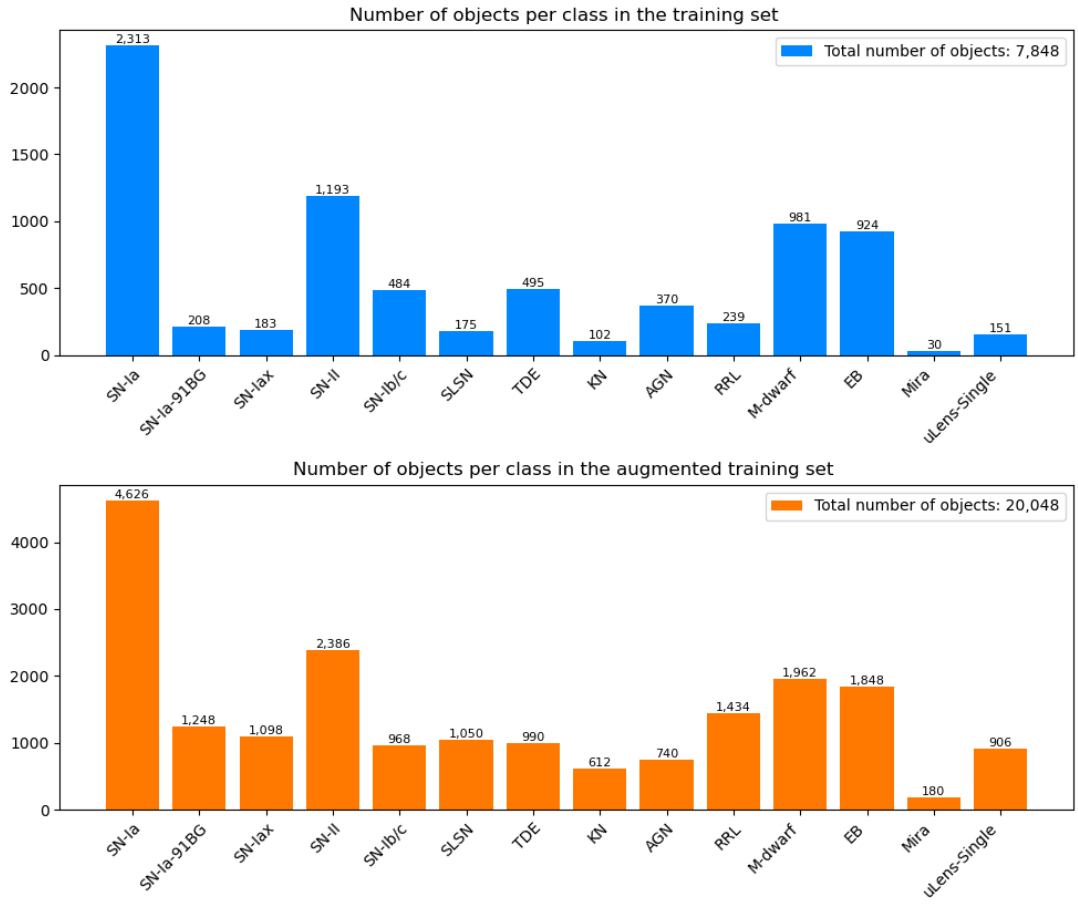


Figure 3.7 Objects per class for the original training set (top figure in blue) and for the augmented training set (bottom figure in orange).

set, while accuracy either improves slightly or remains the same. This means that augmenting the data set helps the shallow architecture better differentiate between the classes while not significantly improving the number of samples that are missed.

For the deep architecture, on the other hand, there is improvement in the F1-scores but it less dramatic. The accuracy scores, however, get worse. This means that the deep architecture can distinguish better between different classes at the cost of missing a larger fraction of samples than it used to. Perhaps having a larger number of parameters is a disadvantage when dealing with this data set, since it would give the network room to learn information that is not necessarily relevant to define a class.

Shallow CNN				
	24×24		32×32	
Data set	F1-score	Accuracy	F1-score	Accuracy
Original	0.499	0.594	0.485	0.584
Augmented	0.6284	0.6403	0.6371	0.6592

Table 3.5 Comparison of performance (F1-scores and accuracy) of the shallow CNN when trained with the original data set and the augmented data set. The best scores of the two cases (rows) are highlighted in blue.

Deep CNN				
	24×24		32×32	
Data set	F1-score	Accuracy	F1-score	Accuracy
Original	0.519	0.647	0.522	0.661
Augmented	0.5846	0.6082	0.594	0.610

Table 3.6 Comparison of performance (F1-scores and accuracy) of the deep CNN when trained with the original data set and the augmented data set. The best scores of the two cases (rows) are highlighted in blue

If we inspect the confusion matrix in Figure 3.8b, obtained from one of the runs using the augmented training set, we see a diagonal that is more marked than the one we have in Figure 3.8a. The model now seems to be better at telling different types of SNe apart, like SNe-Ia-91BG and SNe-Iax, that got missed

almost completely before. Other classes, like SLSNe and Kilonvae are classified correctly more often though still get frequently confused. Most classes improved their accuracy, but this comes at a cost: a higher percentage of SNe-Ia get labelled incorrectly (mostly as other types of SNe).

For the following experiments only the lower DMDT resolution (24×24) will be considered. This is because the CNNs have very similar performance on both resolutions but the lower resolution makes experiments significantly faster.

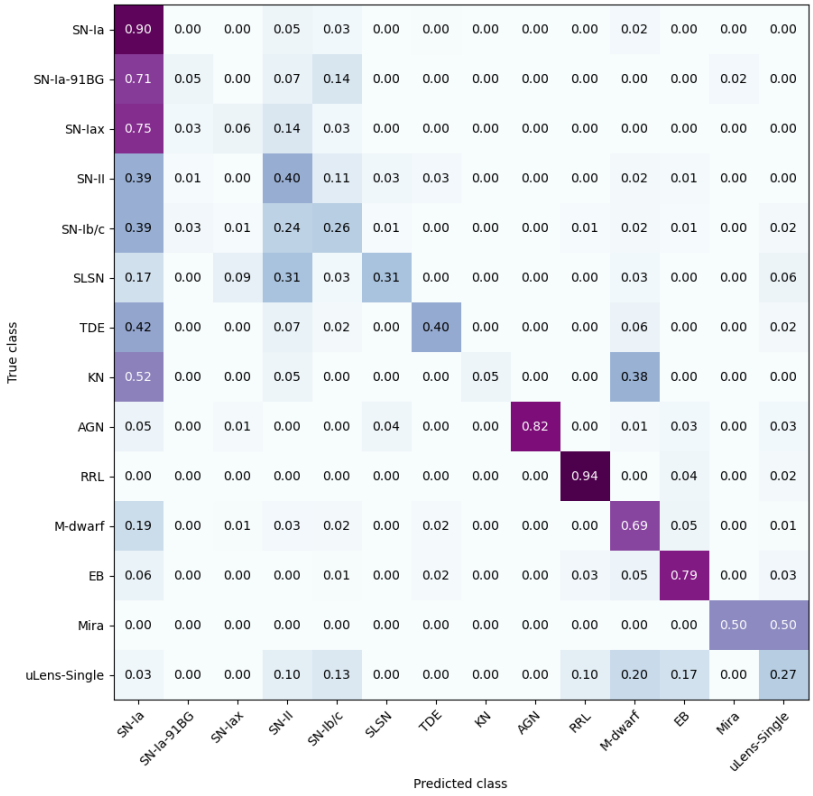
3.3.2 Tuning the model

So far we have been using two architectures described by Mahabal et al. (2017) and so far the more shallow one seems to be working better for the task. In this section, we will start with this shallow CNN and see if changing some of its hyper-parameters helps improve performance. Since trying out every single combination would be too time consuming, instead we will focus on one hyper-parameter at a time, chose the value that yields the best performance, and go on to the next hyper-parameter. This exploratory methodology is quite common in machine and deep learning. Figure x shows the starting point shallow architecture.

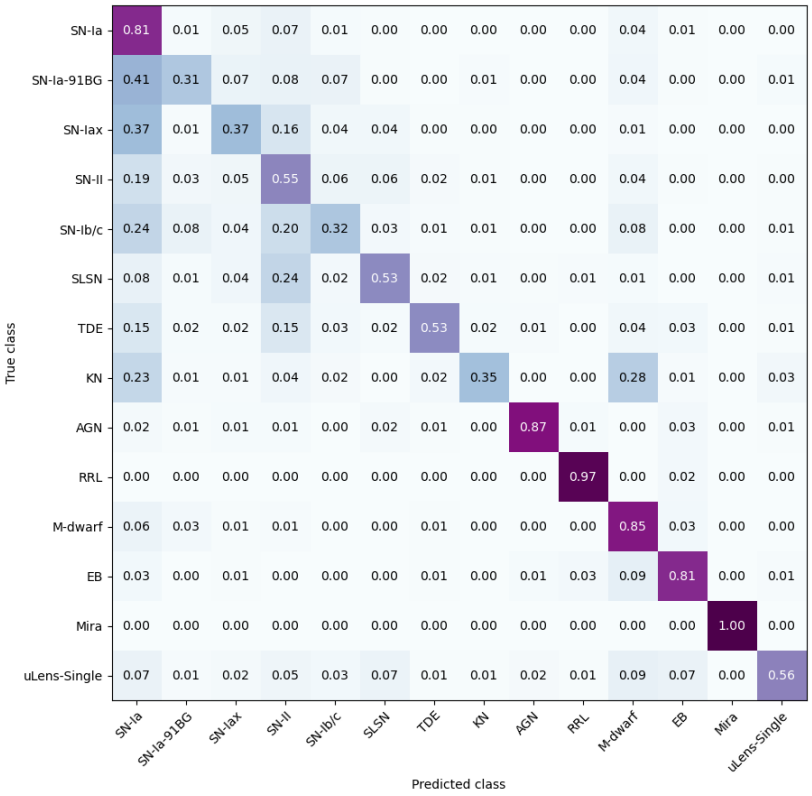
3.3.2.1 Number of convolutional layers and kernel size

As we saw in 1.2, a traditional CNN is made of several consecutive convolutional layers stacked together, usually with pooling operations after each of them. This configuration makes it possible for each consecutive convolutional layer to extract features with different levels of detail. Since after each pooling operation the feature maps would decrease in size, usually the kernel size (convolutional window) of the first layers is wider than the those in the following layers. Since these hyper-parameters are closely related, it makes sense to look at them together.

We start by taking the shallow CNN we had and changing the kernel size of its one convolutional layer, to see if this changes performance. Once we chose the kernel size for this first layer that perform best, we fix it and try different kernel size configurations for the case of having two layers and then we do the same for the case of three layers. Clearly the kernel sizes that are possible to try for the deeper layers are constrained by the output size of each pooling operation



(a) Confusion matrix for the original dataset.



(b) Confusion matrix for the augmented dataset.

Figure 3.8 Normalized confusion matrices of a shallow CNN when trained the original dataset (a) and with the augmented dataset (b).

after a convolution. The results of these experiments are presented in Table 3.7, where we see that again the shallower architectures perform better. Since the best F1-score considered a kernel size 5 for one layer, that is the configuration we will be using for the following sections.

Kernel size				
Layer 0	Layer 1	Layer 2	F1-Score	Accuracy
3	×	×	0.631	0.640
4	×	×	0.634	0.649
5	×	×	**0.641**	0.647
6	×	×	0.640	0.647
5	5	×	0.6077	0.622
5	4	×	0.6157	0.627
5	3	×	0.619	0.629
5	2	×	0.620	0.633
5	2	2	0.561	0.580

Table 3.7 Different number of convolutional layers and kernel sizes and how they affect performance. Better scores appear in darker shades.

3.3.2.2 Number of filters

A convolutional layer yields a collection of feature maps. How many of these maps we get will depend on how many filters we are using for our convolutions. Table 3.8 presents how performance changes when we alter the number of these filters. It seems that there is no gain in increasing the number of filters, so I will keep the value considered initially (32 filters).

3.3.2.3 Input of the linear layer

In a traditional CNN, the convolutional blocks are often followed by a linear (dense) layer, that uses the feature maps extracted to distill values to give to the final output layer, which will yield the actual classification. How many of these values we extract from the feature maps is also a hyper-parameter that we can tweak. Table 3.9 presents how increasing this value increases performance,

Number of filters	F1-Score	Accuracy
24	0.638	0.642
32	0.641	0.647
40	0.640	0.645
48	0.640	0.645

Table 3.8 Number of filters and how it changes performance. Best results are highlighted in blue.

though the difference is marginal for higher values. Since our test unlabelled set is very different from the labelled validation set that we are using at this point, we will use 128, which gives one of the better performances while at the same time not increasing the number of parameters in the CNN too much.

Input size of the linear layer	F1-Score	Accuracy
32	0.629	0.643
64	0.631	0.642
128	0.641	0.647
256	0.639	0.649

Table 3.9 Input size of the linear layer and how it affects performance. Best results are highlighted in blue.

3.3.2.4 Dropout layers

Dropout layers are one of many regularization techniques used to prevent the network from over-fitting the training set. Since our training and validation set are not very representative of the test set, this is particularly important. A dropout layer basically entails randomly *dropping out* elements with a given probability. Dropout layers are usually only active during the training phase.

Since our training and validation sets are alike, it makes sense that reducing the dropout probability would increase performance. However, because our validation sets and the test set are very different, it makes sense to see how much we can increase the dropout probabilities without impacting the performance over the

validation set too much. In our current configuration there are two dropout layers: one after the convolutional layer and one after the linear (dense) layer. How changing their probabilities affects performance can be seen in Table 3.10.

Dropout 0	Dropout 1	F1-Score	Accuracy
0.1	0.25	0.641	0.649
0.2	0.25	0.635	0.637
0.25	0.25	**0.636**	**0.644**
0.3	0.25	0.617	0.630
0.1	0.3	0.637	0.634
0.1	0.5	0.629	0.612
0.25	0.3	0.618	0.634
0.25	0.5	0.628	0.602

Table 3.10 How changing the probabilities of the dropout layers alters performance. Dropout probabilities are presented in orange, with higher ones in darker shades. F1-Scores and accuracy scores are shown in blue, with higher values also in darker shades.

Although the best performance is achieved with lower dropout probability values for both the output of the convolutional and linear layers, I will use for the test set the configuration presented in bold font (0.25 and 0.25), since this combination does not sacrifice performance a lot, but still increases the dropout probabilities.

3.3.3 Tunning hyper-parameters

The last thing we need to look at before trying out the test set is the learning process itself, that is to say, the hyper-parameters related to the experiment. There are many variables that could be tweaked, but I will focus on three that have to do with how fast the model learns.

3.3.3.1 Learning rate and weight decay coefficient

As we saw in 1.2, the learning rate (η) and weight decay coefficient (wdc) influence the learning algorithm we are using, in this case, *Adam*. I tried a few different

values and decided to keep the ones in bold font in Table 3.11. Although smaller weight decay coefficients had better performance over the validation set, by having this value be small, the risk of over-fitting is higher. The value chosen does not sacrifice a lot in performance while at the same time being higher than the usual default value. The learning rate will be kept small, since we want to avoid the learning algorithm from taking big steps and get trapped in local minima.

η	wdc	F1-Score	Accuracy
10^{-2}	10^{-2}	0.403	0.468
10^{-3}	10^{-2}	0.636	0.644
10^{-4}	10^{-2}	**0.679**	**0.687**
10^{-5}	10^{-2}	0.617	0.631
10^{-4}	10^{-3}	0.690	0.694
10^{-4}	10^{-4}	0.696	0.703

Table 3.11 How changing the learning rate and weight decay coefficient of the experiments changes performance.

3.3.3.2 Batch Size

Finally, we see whether the batch size affects performance. I will keep the value I was using by default, since it gives the best results while at the same time not incurring in a high training time penalty, as can be seen in Table 3.12. Changing this value does not seem to significantly impact performance.

Batch size	F1-Score	Accuracy	Time (s)
32	0.673	0.683	229.8
64	**0.679**	**0.687**	183.9
128	0.672	0.681	141.7

Table 3.12 How changing the batch size during training affects performance and elapsed time. Higher values are presented in darker shades and the best results are shown in bold font

3.3.4 Classification of the test set

After all the previous experiments it was finally time for trying out the model with the test set.

The final model was trained once with all the data available in the augmented dataset, as opposed to using cross-validation like in the previous training phases. This final model was trained using all the previously chosen hyper-parameters. Given that, on average, other runs stopped learning at around 60 epochs, the epoch budget was reduced from 100 to 60.

I first ran the test set classification without considering Class 99. The confusion matrix obtained can be seen in Figure 3.9. This confusion matrix (and all of the following ones) was obtained once the competition ended and the test set labels were released. Performance scores are presented in Table 3.13.

We can see in Figure 3.9 that the model failed to distinguish between different types of SNe and other explosive transients and was heavily biased towards the class with the most samples: SNe-Ia. The model was also wrongly predicted objects to be SNe-II, Eclipsing Binaries and M-dwarf stellar flares, which are the next most frequent classes.

3.3.4.1 Class 99

After ignoring Class 99 in the first test run, I tried probing the classification results to see whether I could determine some sort of heuristic that could help distinguish objects from the mystery class. The classification results had to be uploaded to the site of the competition as an .csv file, where each row would represent a sample and would contain predicted probabilities for each of the classes. So essentially, I looked at the probabilities the model predicted and tried to determine when the model was confused about an object and whether this meant the object belonged to Class 99.

So I first defined a threshold. An object would be considered to be classified with low certainty if none of the possible classes had a predicted probability higher than 0.8. Second, I considered that since the most prominent classes are SNe-Ia, SNe-II and M-Dwarf stellar flares, special attention needed to be paid to the confusion between these particular classes. The first three have similar explosive

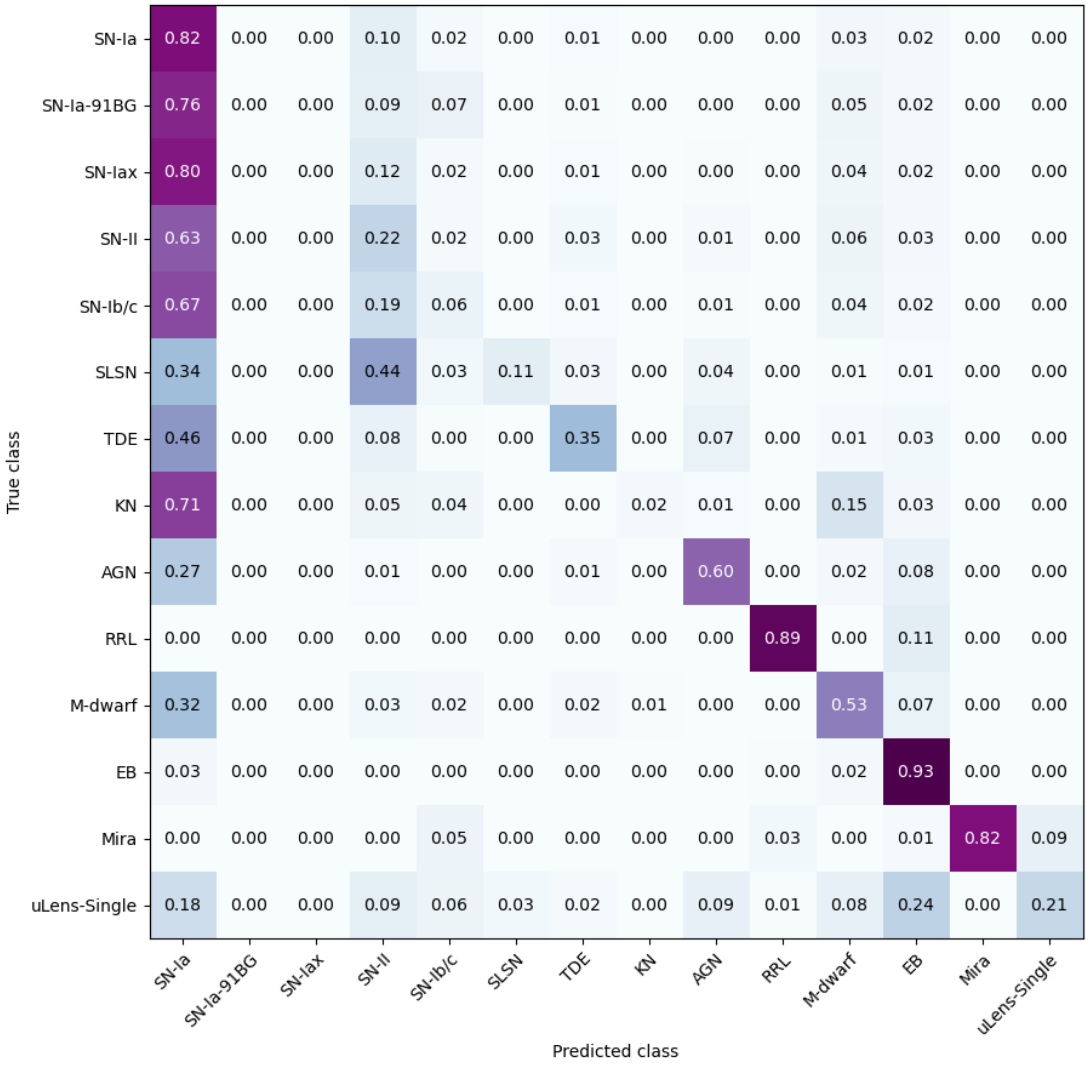


Figure 3.9 Confusion matrix for the classification results, without considering Class 99.

light curves, so probabilities for each of them would have to be roughly similar. I set as a second rule that all this classes had to have predicted probabilities within 0.1 of each other.

Given that other types of SNe and Kilonovae also have explosive light curves, the above method could simply detect these types of objects, instead of Class 99 objects. Hopefully, if this were the case they would also have a somewhat high probability for their actual class, so I only considered to be of Class 99 all objects that satisfied the above conditions and also did not have a prominent probability (above 0.2) for any of the remaining explosive transient classes. If any object met all these conditions, then it would be marked as belonging to Class 99. Only 7,257 satisfied these conditions.

Unfortunately, this worsened all metrics (Accuracy, F1-score and Weighted log loss), since only about 1% of the objects detected actually belonged to Class 99. Results can be seen in Figure 3.10 and Table 3.13. The F1-score is very low, given that for many classes, the objects where completely missed. The weighted log loss is high, especially since classes that had higher weights (weights were inferred by the community by probing the leader board of the competition), such as Kilonovae and Class 99 where missed entirely.

After the test labels were released, it was possible to see that Class 99 was not one, but four different types of objects. Because some of their light curves (Pair instability SNe, for one) ressemble regular SNe, they were actually predicted to be SNe-Ia or SNe-II by the model with high probability. This is why these heuristics failed.

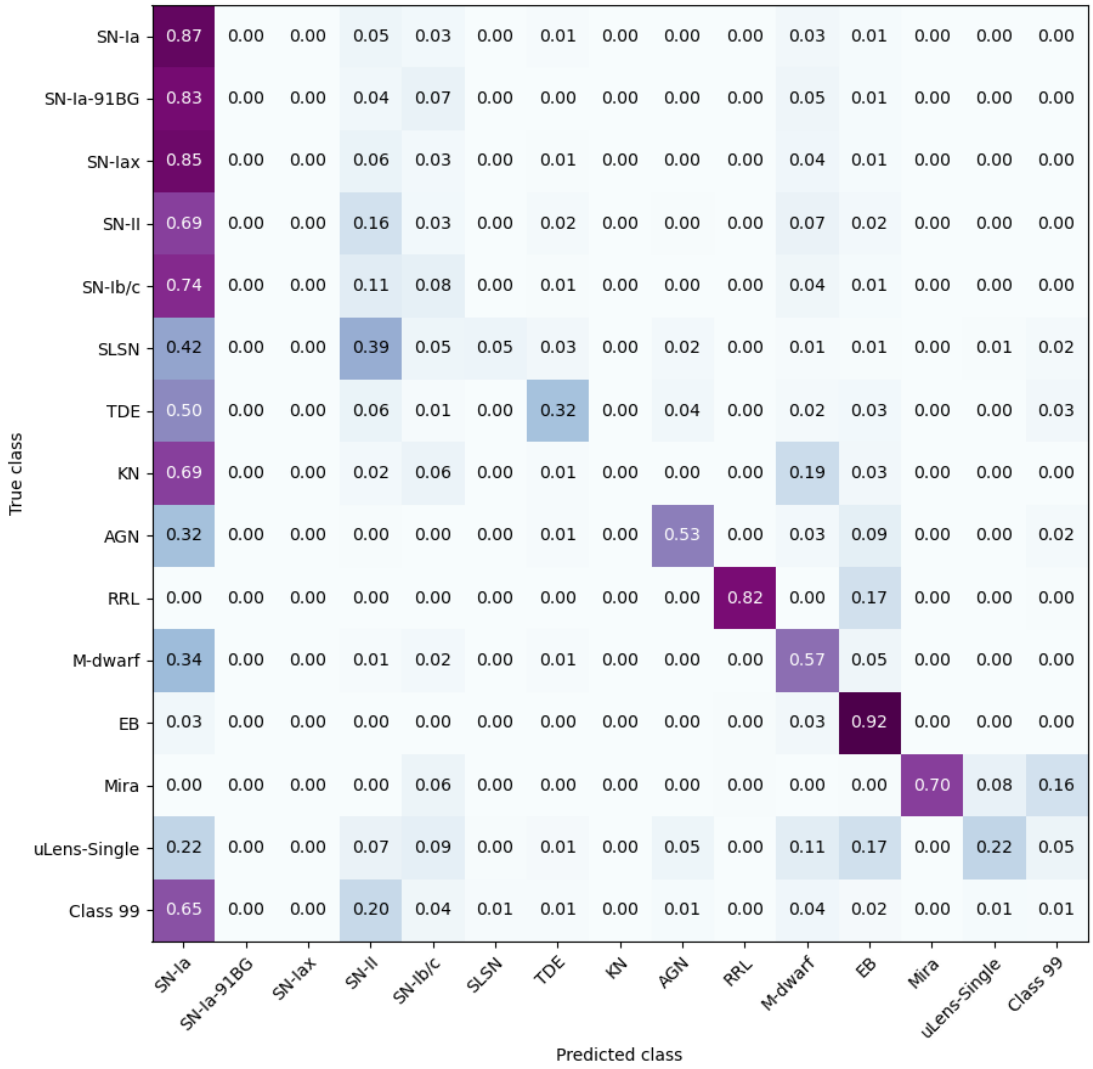


Figure 3.10 Confusion matrix for the classification results, considering Class 99.

Class 99	F1-Score	Accuracy	WLLoss
×	0.366	0.571	1.433
✓	0.321	0.569	1.696

Table 3.13 Result scores for the test set, with and without considering class 99.

3.4 Follow-up experiments: Using the test set for training

After the labels of the test set were released, I decided to run a few experiments to try to figure out why the DMDT approach did not work for the challenge. Looking at the confusion matrix of the test set, one thing is very clear: all objects that have 'SN-like' light curves, that is, light curves that have a sharp rise and then decay, get classified as SNe-Ia, which is simply the most frequent of them. Does this mean that the approach fails to see the differences between these objects? Since the training set is small, perhaps the issue is that there were not enough representative samples for the model to learn to distinguish rarer classes of SNe.

In order to test this, we will see what happens when the model is trained with a portion of the test set, which contains more data that should be representative. For this I used $300,000$ objects randomly selected, and also considered cases where these objects were given to the model in addition to the original training set and the augmented training set. For all experiments, the training data was divided into 5 K-folds, where in each fold 70% of the data was used as training set, while the remaining 30% was used as validation set. Scores can be seen in Table 3.14, and the corresponding confusion matrix is presented in Figure 3.11.

We see that although the mean accuracy and loss improved a lot, the F1-score got only a little bit better. Looking at the confusion matrices it is clear to see why: again most SNe get classified incorrectly, but the accuracy and loss got better because a much bigger number of SNe-II was correctly labeled. More SLSNe were classified correctly as well, but some objects, like TDEs, are missed almost entirely when previously a third of them could be distinguished. What happened?

The model is clearly biased towards the classes that have the most frequent number of samples. The test set has many more SNe-Ia and SNe-II than anything

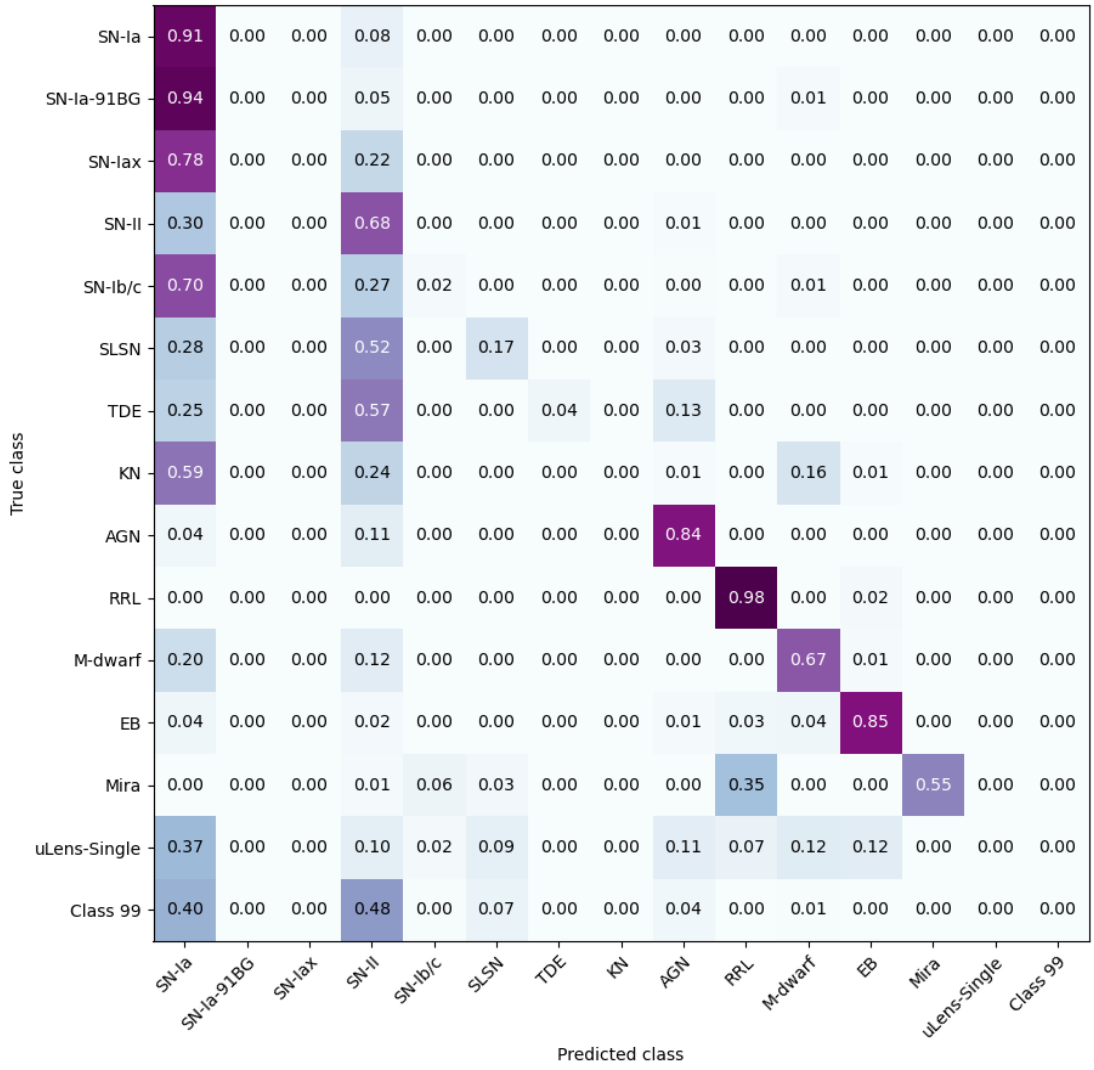


Figure 3.11 Confusion matrix for the classification results of a model trained with a random portion ($\approx 1/10$) of the test set, plus the augmented training set.

else, and that gap is much more marked than it was in the training set, which the classification results reflect. The question that follows is: what happens when the training set has the same number of samples for all classes?

Unfortunately, some classes have very few samples, so having a completely balanced training set is not possible without some data augmentation. For this experiment, however, I will use only data from the test set, so the new training set will not be balanced completely. For each class that has over $10,000$ samples, that number will be considered in the training set. If a class has fewer objects than that, half of the samples will go on the training set and the rest will be left in the test set. The resulting confusion matrix can be seen in Figure 3.12. Cases

where the semi-balanced dataset is merged with the original training set and then with the augmented training set are also considered. Results are also presented in Table 3.14, and a corresponding confusion matix is presented in Figure 3.12. We see that the F1-scores are very similar to the ones we had in the unbalanced cases, but the accuracies and losses got worse.

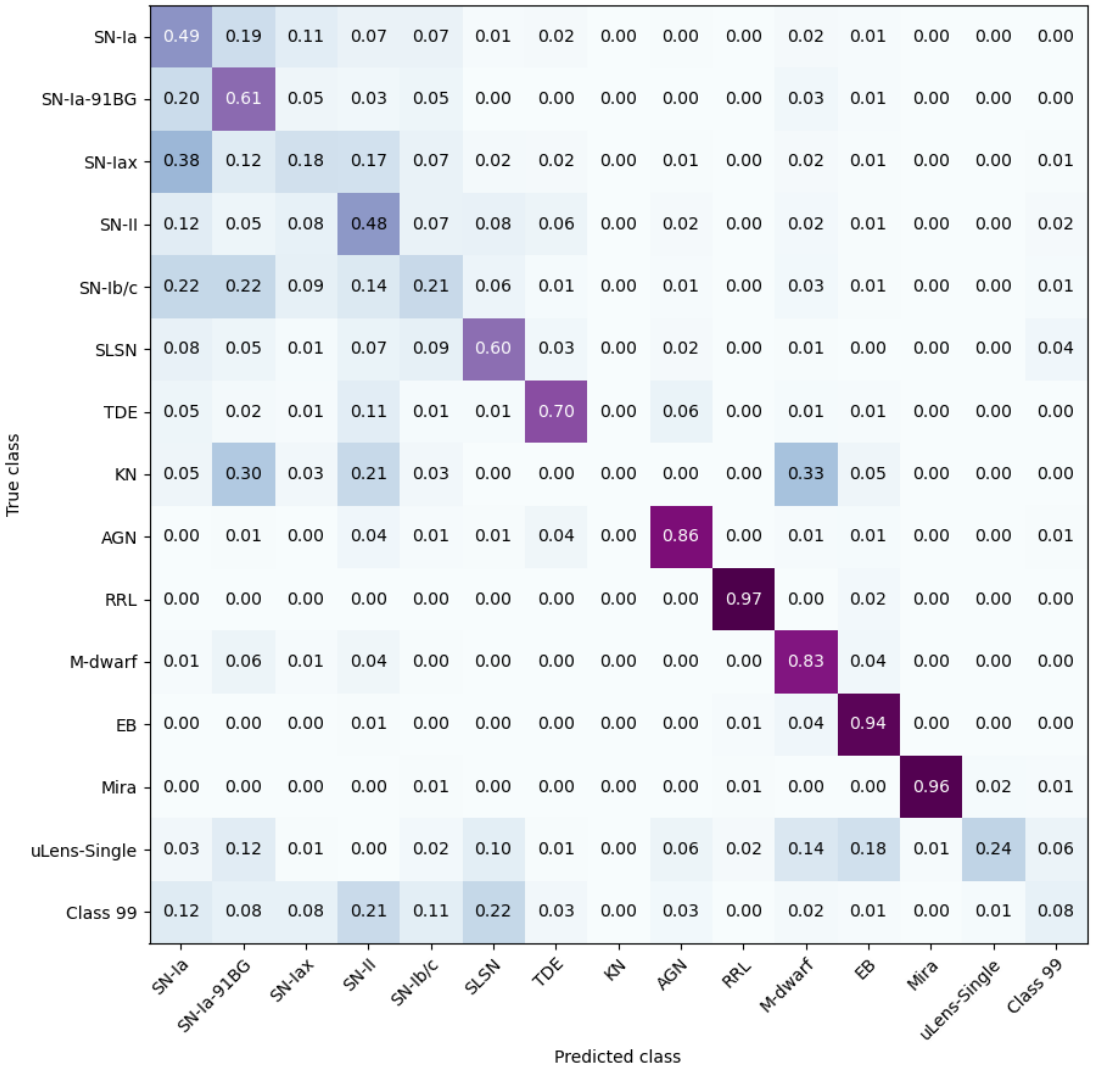


Figure 3.12 Confusion matrix for the test classification results of a model trained with a semi-balanced portion of the test set, plus the augmented training set.

Checking the confusion matrix, we see that this is probably because now a big fraction of the SNe-Ia are missed, and these objects constitute the bulk of the test dataset. At the same time, some classes that were missed before, like SNe-Ia-91BG, SLSNe and TDE are now detected a lot more often. Other classes SNe-Iax and SNe-Ib/c are still seldom distinguished, but not completely overlooked. It is clear than when the data is more or less balanced, there is less (but significant

)confusion between different types of SNe.

In all cases we still see that classes with really few samples, like class 99 and Kilonovae are not labelled correctly at all.

Training data							
N Objects	Or.Tr.	Aug.Tr.	Frac.T.	Bal.T.	F1-score	Accuracy	WLLoss
7,846	✓	×	×	×	0.309	0.504	1.599
20,048	×	✓	×	×	0.321	0.569	1.696
300,000	×	×	✓	×	0.383	0.751	0.804
307,846	✓	×	✓	×	0.398	0.752	0.803
320,048	×	✓	✓	×	0.395	0.750	0.803
117,986	×	×	×	✓	0.378	0.410	1.500
125,832	✓	×	×	✓	0.383	0.451	1.445
138,034	×	✓	×	✓	0.408	0.528	1.345

Table 3.14 Result scores for the test set when training the model with different training sets: the original one (Or.Tr), the augmented one (Aug.Tr.), a random sample of the test set (Frac.T.), a semi-balanced fraction of the test set (Bal.T.) and combinations of those. Obviously, when training with parts of the test set, the portion used for training is not used as test set after. The best scores for each metric are highlighted in blue.

3.5 Overview of the winning solutions

Before going on to discuss why the DMDT approach failed, what could have improved it, and how to go forward, let us briefly review what were the winning solutions to see what we can learn from them.

3.5.1 First Place

The winning solution of the PLAsTiCC competition used prediction and classification combined. First, the training data set was augmented by degrading the light curves so they would more closely resemble the test set. This degradation process

included modifying the brightness of galactic objects, modifying the redshift of the extragalactic objects, deleting chunks of data from the light curves and adding noise to the light curves. After this data augmentation procedure, the training grew to around 270,000 objects.

This augmented noisy data set was then pre-processed in order to make the light curves more distinct: Gaussian process (GP) regression was used to predict the 'missing' points in the light curves and thus create features. The winner stated in the discussion forum at Kaggle that this prediction alone took his machine three days of computation. This is not surprising, since GPs are known to be very slow, but handle large uncertainties very well in return. Once the sparse light curves were transformed into neat ones, the winner engineered around 200 different features: some of them were statistical features (peak brightness, width of the light curve, etc) and some of them specifically related to how well the GPs were expected to perform, given the original number of points in the time series.

A Gradient boosting machine (GBM) was then trained using 5-fold cross-validation, which obtained a score of 0.726 on the leader board for the weighted log loss. In order to improve the score, the winner of the competition chose to probe the leader board. This is within the rules. This way, he found out that the weighted average of predictions for classes 42, 52, 62 and 95 (SN-II, SN-Iax, SN-Ib/c and SLSN) gave a good estimate of class 99. With this, the winning score was improved to 0.670.

More information can be found in the winner's description of his method at Kaggle[2].

3.5.2 Second Place

The second place solution consisted of nine trained classifiers, two of which were Gradient Boosting Machines (GBMs) and seven of which were RNNs. The final score was a combination of all of them. As a data augmentation step, they added random variations to the light curves, which helped to improve the score of the NNs, but not of the GBMs.

To train the GBMs, some adjustments were made to the fluxes of the training data set. Although trying to normalize the fluxes did not help, multiplying them by

[2]https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75033

the photometric redshift of their host galaxy improved the classification results. Another feature that turned out to be important was the flux/error ratio of the light curves.

On the other hand, to train the RNNs both fluxes and flux error were used as features, along with metadata. Both normalization of fluxes and data augmentation by adding random noise to them was used.

They also trained a separate model to predict the spectroscopic redshift of the sources that did not have one assigned in the metadata. This was used as input feature for both the GBMs and the RNNs. They also probed the leader board to deal with class 99.

More details can be found in the description at Kaggle[3].

3.5.3 Third Place

The third place solution of the PLAsTiCC challenge is the combination of three solutions, which shared features and later combined their scores.

One of these solutions consisted on a CNN that received as an input a 128 long vector with 18 channels. Of those channels:

- 6 channels correspond to a linear interpolation of the flux (one channel per passband).
- 6 channels correspond to a linear interpolation of the flux multiplied by the flag 'detected'. According to the PLAsTiCC competition data note, if this boolean value equals 1, then the object's brightness is significantly different at the 3σ level relative to the reference template.
- 6 channels describe the distance between the sampling points, in other words, if the linear interpolation is done with points that are very far apart, then this channels will present high values.

As additional pre-processing steps, white noise proportional to the flux error was added to the light curves before performing the linear interpolation. After the interpolation, the dataset was augmented by both performing a cyclic shift and skewing the light curves.

[3]https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75059

The CNN was based on a Fully Convolutional Neural Network presented by Wang et al. (2016b), except the output of the last layer is concatenated to a Multilayer Perceptron (MLP) that also takes features engineered manually. There were three versions of this CNN: one with features that belonged to the metadata and other two that took features from the other sub-soltions.

Another solution used GBMs and feature engineering to train three different classifiers. One of the classifiers was trained with 1,000 features relevant to galactic objects, one was trained with 2,000 features relevant to extra-galactic objects and the last one is the same as the second one, except it also considers the spectroscopic redshift of the objects when available. The features used to train these classifiers are varied: some of them are statistical features (such as amplitude, maximum slope, mean, variance, etc of the light curve) and some of them were computed by using GPs to predict the light curves.

The third part of the third place solution also used GBMs, where the features given were calculated by using template fitting.

To deal with class 99, this team also probed the leader board.

3.6 Conclusions

While attempting the PLAsTiCC challenge and after reviewing the solutions, a number of things are clear.

It seems that the DMDT approach was not ideal for this type of classification and that it is also very sensitive to class balance. Although it worked well for coarser taxonomies, it has trouble distinguishing finer details in explosive light curves. I think the reason for this is the loss in sequentiality of the data: it is difficult to appreciate the relations between the peak luminosity and width of a SNe light curve without it.

Evidently the pre-processing of the data is a crucial step for this type of light curve classification problem, both because of the differences between training and test set and because of the sparseness of the data. All winning solutions have in common that they degraded the training light curves in one way or another to make them more similar to the test set, similar to the process described in Section 3.3.

The winning solutions accounted for the sparseness of the light curves by reconstructing them in different ways, without losing their sequential shape. In the case of the first place solution, this was the most computationally expensive step by far: although GPs yield accurate results in general, their complexity is $O(n^3)$ over the number of data points in a light curve. In contrast, the interpolation method used in the third place solution has linear complexity. Since both approaches obtained similar scores, the last one seems like a better option if we consider online classification systems.

One issue that was not resolved during the PLAsTiCC challenge was the detection of anomalies, that is, the labelling of class 99. All winning solutions probed the leader board to figure out heuristics to detect it, which would not be feasible or useful in a real life scenario.

Aside from transients with few samples, SNe seem to be the most difficult to tell apart. It makes sense, since their light curves are very similar, and some of them also have very few samples. In the next Chapter, we will focus on SNe, the relations between the different parts of their light curves and whether they could be detected early on if we pay attetion.

Chapter 4

Paying attention to the flare

4.1 Introduction

We saw in Chapter 3 that the PLAsTiCC challenge included light curves of several different types of simulated astronomical events, but it was clear that there was a group of transients that were the most difficult to tell apart from one another. In this chapter, we will *pay attention* to them: Supernovae.

Supernova light curves are tricky to differentiate simply because they are very similar in shape. Indeed, we saw in Section 1.1.2 that their taxonomy has not much to do with their light curves and is instead based on their spectrum. In spite of this difficulty, it is critical that we are able to classify them with high purity and sensitivity, since the study of these phenomena has

There is substantial research devoted to the classification of supernova light curves (as seen in Section 1.3), both for the binary classification problem (SN-Ia vs all other Types) and for the multi-class problem (SN-Ia, SN-II, SN-Ib/c, SLSN, etc). While most methods presented to date achieve high levels of accuracy and purity, they are usually only applied to simulated datasets and only retrospectively, that is, once an entire simulated event has occurred. There are exceptions, however: RNNs have been used to study early classification of simulated transients (Charnock & Moss, 2017b; Muthukrishna et al., 2019), while RFs and other models have been used to classify real light curves and deal with the difficulties that they present (Sánchez-Sáez et al., 2021). The issue of having small training sets of spectroscopically confirmed SN light curves that are not

necessarily representative of the data we will need to classify in the future remains though, and this makes it difficult to classify real SNe, especially early on. We also saw in Chapter 3 that there is no clear way to reliably detect outliers as of yet.

Many disciplines besides astronomy deal with time series analysis. One field that has made significant advancements is Natural Language Processing (NLP), the study of speech through computational techniques. NLP greatly improved performance in the task of language translation with the introduction of a mechanism called *Attention*.

Attention allows a DL model to asses the relations between different parts of a sequence and take them into consideration when performing a task, which is why it is so useful for automatic language translation. Translation tasks are usually performed by DL models with an *encoder-decoder* architecture, where the encoder converts a sequence (sentence) into a lower dimensional space, called the *latent space*, and the decoder deciphers that code, that in this case would be the same sentence in a different language. The issue with this architecture is that different languages have different grammars, so it is not straightforward to just translate word for word. If we wanted to translate 'I don't speak French' to French ('Je ne parle pas Français') we need our model to understand that 'ne pas' and 'don't' have the same meaning. This is where Attention plays a huge role. With it, we can train a model to 'attend' to the relations between those two negations despite them being placed in different places in the sequences.

Self-Attention is the special case where instead of relating two different sequences, the Attention mechanism attends to the different parts of the same sequence. This has been used, for example, in fields such as *Sentiment analysis*, the study of how language is used to express mood and emotion. For example, if we have the sentence 'I really love the bitter flavour of coffee, it's my favourite beverage', we could use Self-attention to gauge that 'love' and 'favourite' are more closely related to each other than to other words, and that 'bitter', 'flavour' and 'coffee' are likewise aligned. A ML model can use this information to extract meaning or an overall mood from a sequence.

The sequences in our case will be both simulated and real photometric measurements, and what we want our model to do is pay attention to the relations between the flare of the SN and its earlier parts and see whether this information can help us classify the SNe as early as possible.

This Chapter is structured as follows: First, in Section 4.2, I explain Attention and Self-Attention in more detail; then, in Sections 4.3 and 4.4, I describe the baselines and data representations relevant for the rest of the Chapter; in Section 4.5 I propose a Self-Attentive model for light-curve classification; Section 4.6 contains experiments where the Self-Attentive model is tested against the PLAsTiCC dataset, focusing on earliness of classification; Section 4.7 details how the same model is used to classify two new datasets, a simulated one and a real one from the ZTF surveys, both described in Section 4.7.1; and finally, in Section 4.8, we will summarize what we learn and conclude from this Chapter.

4.2 Attention and Self-Attention

Attention was first proposed (Bahdanau et al., 2015) to be used in combination with RNNs (Luong et al., 2015; Cheng et al., 2016), but later on other DL architectures proved to benefit from it (Xu et al., 2015; Ramachandran et al., 2019). It really revolutionized the field though, when it was empirically proved that it was a tool powerful enough to be used on its own (Vaswani et al., 2017).

Attention can be understood as a function that maps a set of queries Q and a set of key-value pairs (K and V) to an output. The output A of this function will be the weighted sum of the values V with alignment scores S, that represent the compatibility of Q and K. This compatibility is measured through an aligment function f_a:

$$S = \text{softmax}(f_a(Q, K)) \tag{4.1}$$

$$A = SV. \tag{4.2}$$

In the case of Self-Attention Q, K and V all come from the same set of values. Given an input $X = \{x_i\}_{i=1}^{t}$ where $X \in \mathbb{R}^{n \times t}$ and $x_i \in \mathbb{R}^n \ \forall i \in [1, ..., t]$, we define Q, K and V as:

$$Q = \{q_i\}_{i=1}^{d} = W_q X \tag{4.3}$$

$$K = \{k_i\}_{i=1}^{d} = W_k X \tag{4.4}$$

$$V = \{v_i\}_{i=1}^{d} = W_v X. \tag{4.5}$$

where Q, K and $V \in \mathbb{R}^{d \times t}$ and q_i, k_i and $v_i \in \mathbb{R}^d \forall i \in [1, ..., t]$. W_q, W_k and W_v are trainable weights.

Note how this mechanism obtains the alignment scores S from its inputs that it then uses as weights over the same inputs. In other words, although there are learnt weights involved, Attention calculates new weights S on the go which are dependent on what it receives. Note also how S involves a softmax function, so these scores can be understood as a probability distribution. We can intuitively interpret this mechanism as a way to score the importance of each element in a sequence, so elements that get a higher score are effectively being highlighted. Hence the name Attention.

4.3 Baseline Architectures

In order to explore how attention can help in light curve classification, we first need to define how a light curve will be represented and what architectures we will be comparing. We will review three models that will serve as baselines and later a fourth one that will incorporate a self-attention mechanism.

The architectures I chose as baselines were loosely described in Section 1.3.1.2, but we will now focus on the specifics.

4.3.1 Fully Convolutional Neural Network (FCN)

This architecture was chosen because it is considered to be a strong baseline in time series classification(TSC) (Wang et al., 2016a) and because an FCN-like architecture won the third place in the PLAsTiCC challenge, as previously described in Section 3.5.3. A FCN is a type of CNN where instead of performing a pooling operation after each convolutional layer, a global pooling operation is performed at the end of the network. The global pooling layer acts as substitute for the fully connected layer that is usually there. The architecture is set up so the number of feature maps produced by the global pooling layer is the same as the number of categories relevant for the classification task. Thus, the idea is that, through back-propagation, the network will adjust so that each feature map produced by the global pooling layer acts as a sort of confidence map for each of the categories. While fully connected layers are prone to over-fitting, global

pooling layers act as structural regularizers and help prevent this issue (Lin et al. (2013b)).

The FCN architecture used here has three convolutional blocks, each one made up of a convolutional layer followed by a batch normalization layer and a ReLU activation function, as shown in equation 4.6 and Figure 4.1. The convolution operations have kernel sizes $\{8, 5, 3\}$ and number of filters $\{128, 256, 128\}$ respectively. At the end of the network there is a global average pooling operation followed by the softmax function.

$$\begin{aligned} y &= W \otimes x + b \\ s &= BN(y) \\ h &= ReLU(s). \end{aligned} \tag{4.6}$$

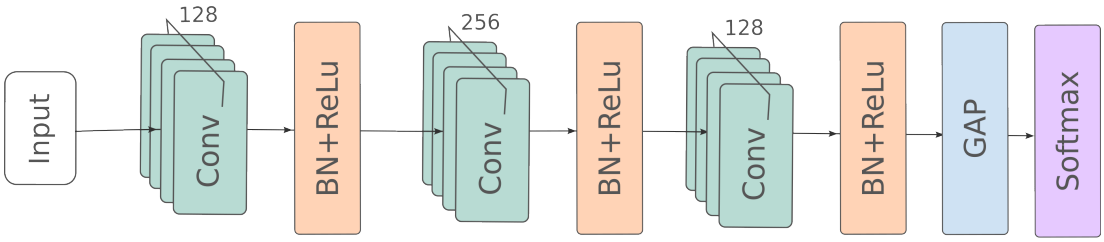


Figure 4.1 Architecture of the fully convolutional neural network (FCN) used.

4.3.2 Residual Neural Network (ResNet)

I chose this architecture because, like the FCN, it achieved good performance in several surveys (Fawaz, 2020; Ismail Fawaz et al., 2019; Bagnall et al., 2017) and it has also been considered to be strong baseline in TSC tasks (Wang et al., 2016a). The particular architecture I will use is made up of three consecutive residual blocks as shown in equation 4.7 and Figure 4.2, where each one is composed of three convolutional blocks (described in 4.6), a shortcut and a ReLU activation function. The idea of the shortcut is to combine the input of the block with the output of the convolutional blocks to enable the gradient flow directly through

the bottom layers.

$$
\begin{aligned}
h_1 &= ConvBlock_1(x) \\
h_2 &= ConvBlock_2(h_1) \\
h_3 &= ConvBlock_3(h_2) \\
y &= ReLU(h3 + x).
\end{aligned}
\tag{4.7}
$$

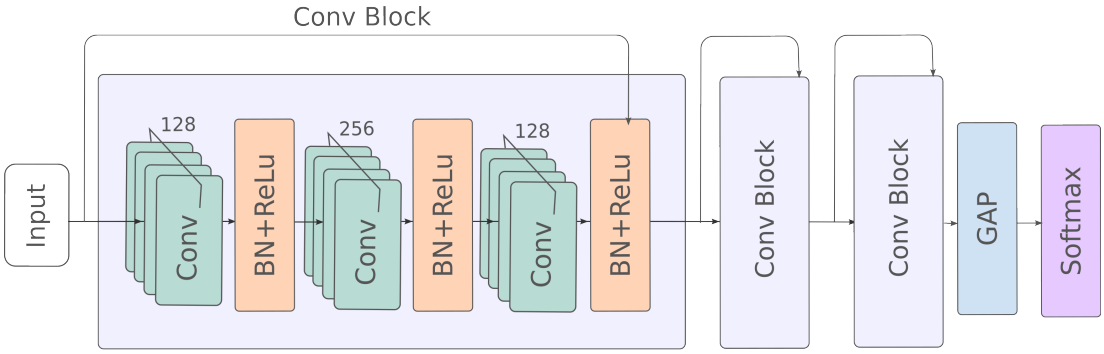


Figure 4.2 Architecture of the residual neural network (ResNet) used.

4.3.3 Recurrent Neural Network with Gated Recurrent Units (RNN GRU)

An RNN was chosen because of the good results this architecture presents in various NLP sequence processing tasks, because an architecture like it won second place in the PLAsTtiCC challenge (as described in 3.5.2) and because it has shown good results in early transient classification tasks (Muthukrishna et al., 2019).

As we saw in Section 1.3.1.2, RNNs are a type of neural network that process each time step of a sequence recurrently in a different copy of a network. There exist different variants, but for these experiments, we will be using Gated Recurrent Units (GRUs).

A GRU computes the following for each element x_t in the input sequence:

$$z_t = \sigma(W_z X_t + R_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r X_t + R_r h_{t-1} + b_r)$$
$$\dot{h}_t = tanh(W_h X_t + R_h(r_t \circ h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \dot{h}_t.$$

Where r_t, z_t and $\hat{h}_t$ are called reset, update and new gate respectively and $\circ$ is the Hadamard product. Ws, Rs and bs are weight matrices and biases learnt through back propagation.

The architecture used in this work is based on the one published in (Muthukrishna et al., 2019), where it achieved good performance. It has two stacked layers of GRUs, each with hidden size of 100 (that is, the size of the hidden state h_t is 1×100), followed by a dropout layer (with dropout rate of 0.2) and a batch normalization layer. At the end of the network there is a fully connected layer followed by a softmax activation layer, as can be seen in Figure 4.3. Another reason why I chose this architecture is because it is intuitive to add an attention layer to it.

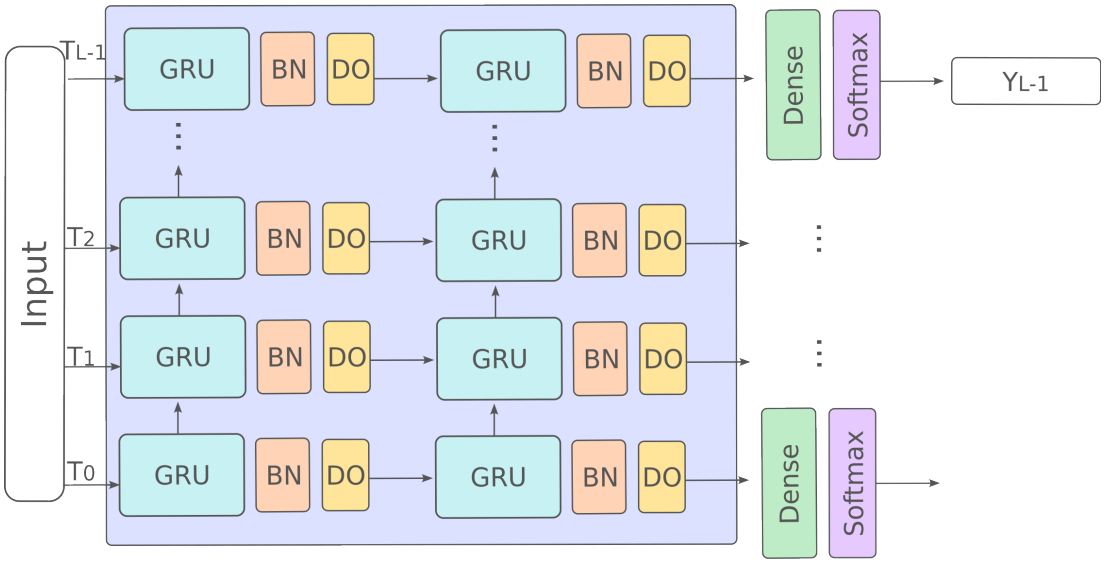


Figure 4.3 Architecture of the RNN-GRU used.

4.4 Self-Attentive Architecture

Self-attention, also known as intra-attention is a machine learning mechanism that teaches a model to extract relevant meaning from a sequence by learning correlations between different parts of it. This technique has been successfully applied to tasks such as sentiment analysis, automatic reading comprehension (Cheng et al. (2016)) and abstractive summarization(Paulus et al. (2017).

The Self-Attentive model proposed here will extend the RNN just described. The basic idea is to compute an alignment vector a that encodes a relevant representation of the hidden states of the RNN $H = (h_0, h_1, ..., h_n)$. This vector is then combined with the hidden states to obtain the context vector c, where the parts that matter most will stand out. The alignment function used to compute a is called additive Attention (Bahdanau et al., 2015), and it was one of the first Attention mechanisms proposed. It is defined as:

$$f_a = \text{softmax}(\left(w_2 \tanh\left(W_1 H^T\right)\right) \tag{4.8}$$

$$a = f_a \tag{4.9}$$

$$m = Ha^T. \tag{4.10}$$

Here W_1 is a matrix of size $d \times n$ and w_2 is a vector of size $r = 1 \times d$, where both r and d are hyper-parameters than can be chosen arbitrarily. The resulting vector m usually learns to focus on a specific component of a sequence, such as a set of related points, so it is expected to reflect one aspect or meaningful relation within the sequence. If we wanted to pay attention to more than one relation in a sequence to obtain an overall representation of it, we need to perform multiple hops of attention. In practice, this can be achieved by changing the hyper-parameter r to be $r > 1$. This would make w_2 a matrix instead instead of a vector. The output of the self-attention layer M would also change in dimensions, where each vector in the second dimension of size r would hopefully encode a relevant relation. While activation function tanh is usually used in this context, it can be changed to a different one provided it also introduces non-linearity. A acts as a form of representing the hidden states H and their relation to themselves. This approach is very similar to the one proposed in (Lin et al., 2017).

Essentially, the alignment vector a is computed by training weights W_1 and w_2 through back-propagation, as one would do on any kind of layer. This differs from a linear layer in that there are no biases. The bottom half of Figure 4.4 visually represents Equation 4.8.

The model presented consists of three parts: the first part is a one-directional GRU such as the one described above in 4.3.3. The second part is a self-attention mechanism, that computes weight vectors to be combined with the hidden states that the GRU outputs to create a representation of the relevant parts of them. The third part is a fully-connected layer that receives this representation and decides on the class of the light curve, followed by a softmax function, which ensures that the final outputs represent a probability distribution. Since this *Self-Attentive* model is essentially the same RNN (GRU) described before, but with an added attention mechanism, we will also refer to it as RNN with Self-Attention (RNN-SA). Figure 4.4 represents the whole architecture.

4.5 Data representation

As we saw in Chapter 3, choosing a data representation is not straight forward. Reviewing the winners of the PLAsTiCC challenge, it is clear that maintaining the sequentiality of the light curves in the representation is important. This leads us to the issue that there must be some sort of interpolation or reconstruction, since the time series present gaps because of the cadence of the observations.

While Gaussian Processes have been used extensively for light curve reconstruction, this approach is expensive in terms of computational time. A naive implementation of this type of regression has $O(n^3)$ complexity, which means that the computational time increases cubically with the number of data points in a sequence. This obviously makes GPs impractical for large datasets, specially those that are continually updated. There are, however, many research articles that look into the issue and come up with methods for the case where the data comes from an online stream that lower complexity significantly (Solin et al., 2018; Koppel et al., 2020). However, some of them require the data points to be equidistant, or assume the data to be one-dimensional. While some work does not seem to have these limitations (Zhang et al., 2019), there is still the practical issue of having a working implementation that can be easily embedded and put to work. Because of all these reasons, GPs will not be considered in this Chapter.

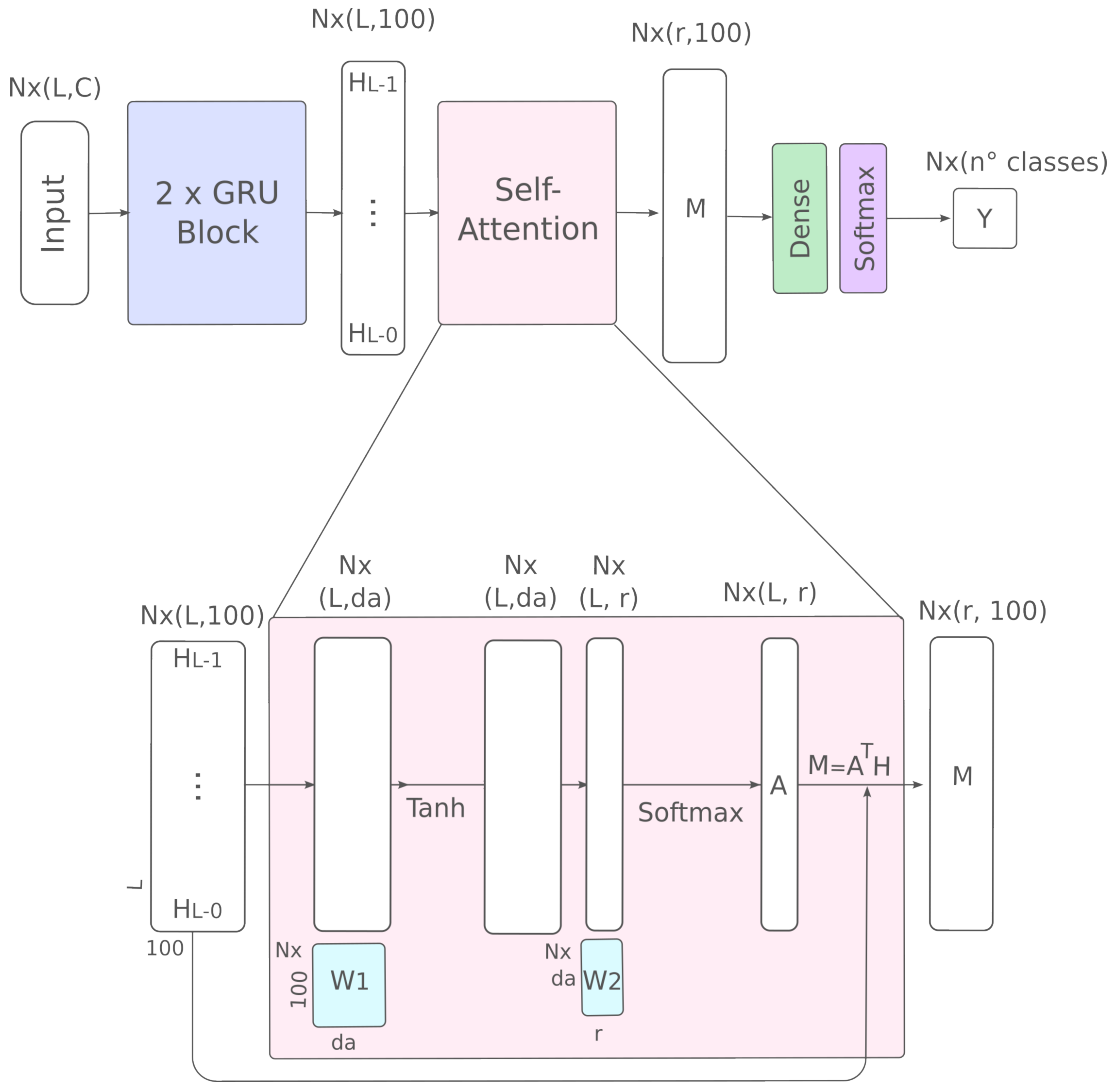


Figure 4.4 Architecture of the Self-Attentive architecture.

The third place solution of the PLAsTiCC challenge uses a much simpler approach that appears to yield similar results, at least for the PLAsTiCC data set, that is, linear interpolation. However, while interpolating the flux values in the light curve is easy, the gaps can be grossly missrepresented as straight lines, so it is important to add information that encodes how close the sampled points are to the actual observation points. The third place solution does this by adding extra dimensions to the time series where each point is the distance to the nearest real point. In this Chapter we will be using this approach due to its simplicity. It is important to note that this data representation does not consider flux errors, which in noisy data could prove to be a major downside.

4.6 Self-Attention applied to the PLAsTiCC dataset

4.6.1 Using the original training set

To test whether the idea of Self-Attention being helpful in transient classification had any sense at all, I started out by testing all three baseline models plus the Self-Attentive architecture and seeing how they perform when confronted with the original PLAsTiCC challenge. I did not explore data augmentation in this case, because I wanted to get an idea of performance when all odds were stacked against the models. The confusion matrices for this experiments are presented in Figure 4.5, and results are in Table 4.1.

In Figure 4.5, we see that the results are similar to the ones obtained in Chapter 3, in the sense that most of the confusion happens around SNe and that the models are heavily skewed towards the class with the most samples (SNe-Ia). Both RNNs do better than the CNNs. In Table 4.1 we see that when considering all classes, the Self-Attentive architecture got the best results.

	All classes				Extra-galactic			
Model	N_e	Accuracy	F1-Score	Loss	N_e	Accuracy	F1-Score	Loss
FCN	55	0.527	0.184	1.623	44	0.574	0.186	1.546
ResNet	19	0.541	0.242	1.438	13	0.033	0.025	6.697
RNN	83	0.540	0.248	1.410	78	0.550	0.230	1.366
RNN-SA	80	0.551	0.308	1.336	62	0.534	0.263	1.303

Table 4.1 Resolutions of the constructed images and how that impacted the performance and training time of two CNN architectures. The F1-scores, accuracy scores and training times represent the mean of these values over 5 different runs. Higher (better) F1-scores and accuracy scores are presented in darker shades (light blue hue). Higher image resolutions are also presented in darker shades (orange hue)

I then performed the same experiment, except this time I limited the dataset to the classes that represent extragalactic objects exclusively. The confusion matrices and results are in Figure 4.6 and Table 4.1 respectively.

The change in number of classes altered results significantly. The FCNN, the RNN and the Self-Attentive RNN (RNN-SA) all tend to classify objects as the classes

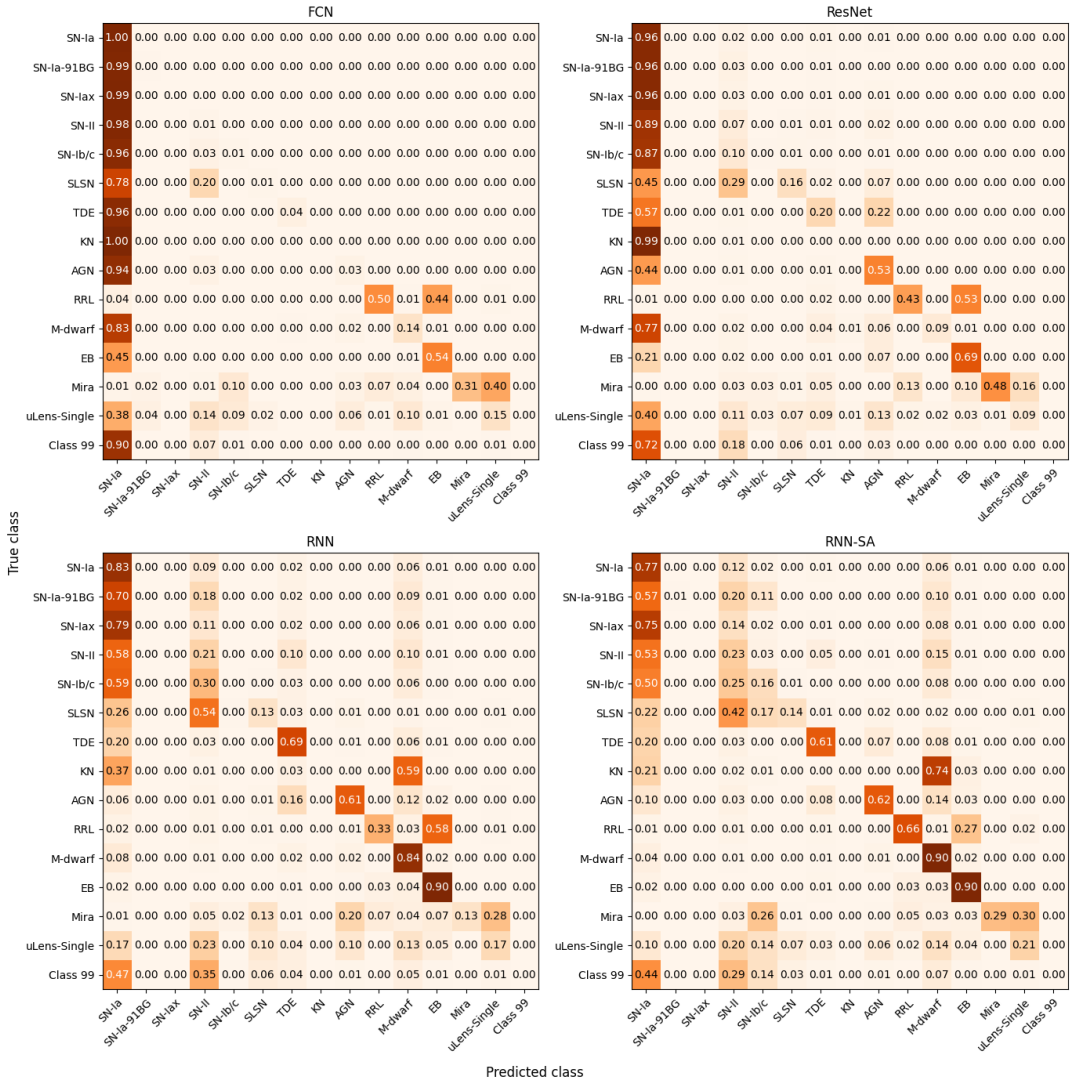


Figure 4.5 Confusion matrices for the test set results of all three baseline architectures plus the self-attentive architecture, when trained with the original PLAsTiCC dataset.

with most samples in the training set, although the Self-Attentive architecture does a bit better. The ResNet, on the other hand, performed remarkably badly. I am not sure why this happened, but my guess is that when there were classes with light curves that looked different from just a rise and a fall, it could learn some features from those classes and then everything else got clumped into the class with the most samples. Now that all classes were much more alike, it failed to learn anything at all. Perhaps it did not classify everything as SNe-Ia because out of all the light curves in this dataset, AGNs are the most random-looking. Since ResNet learnt nothing, all the features it extracted from the test set would have looked random, and so it would have decided to label them as AGN. This

is only a guess. The ResNet is also the model that had the least training epochs, perhaps it needed more *patience* to converge into anything.

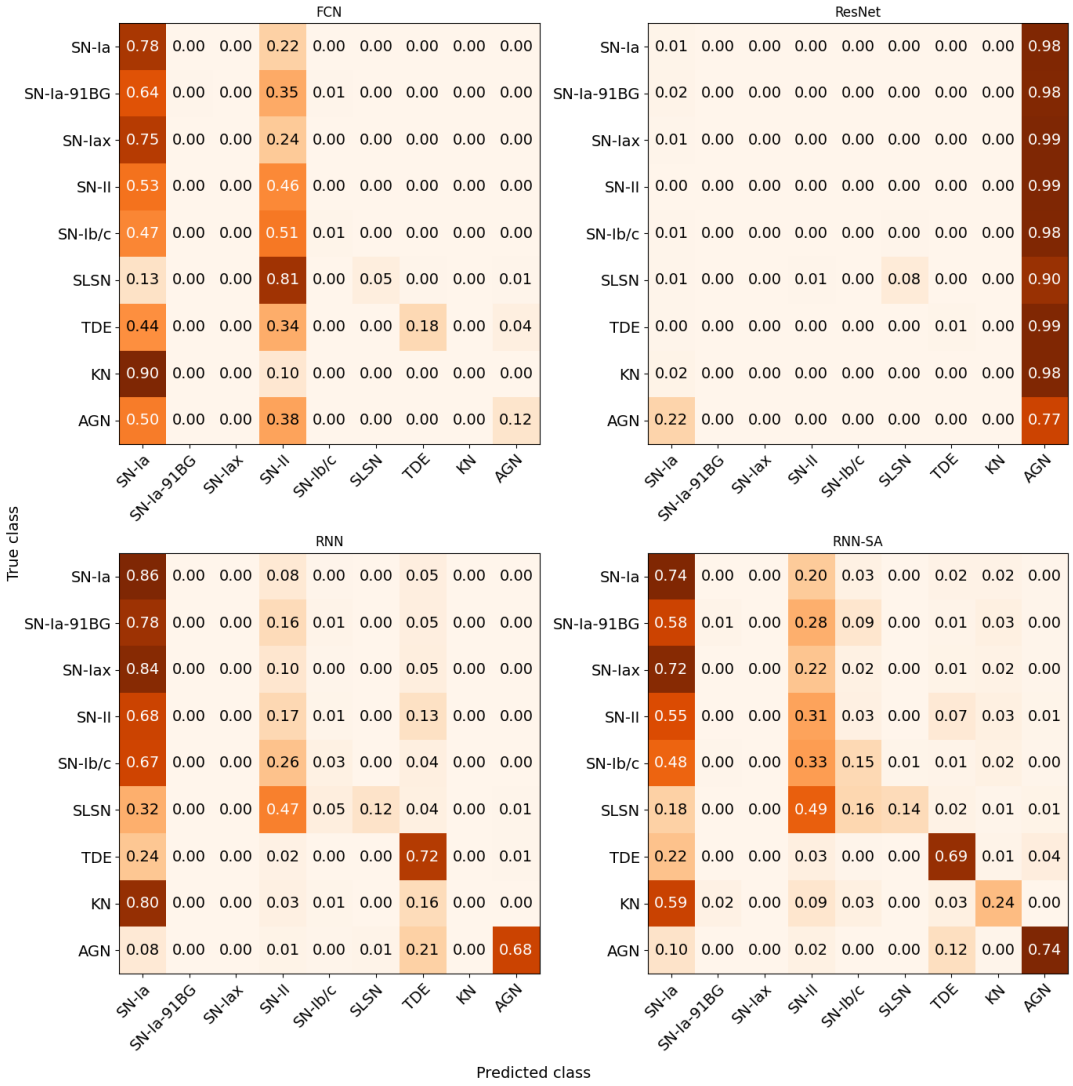


Figure 4.6 Confusion matrices for the test set results of all three baseline architectures plus the self-attentive architecture, when trained with the original PLAsTiCC dataset. Only classes that represent extra-galactic objects were considered.

4.6.2 Using part of the test set as training set

After using non-representative data for training unsurprisingly yielded poor results, I then decided to use part of the test set ($\approx 300,000$ objects) to train all the models. I ran this experiment the for extra-galactic objects only (Figure 4.7).

While for the recurrent models (the RNN and the Self-attentive model) there is

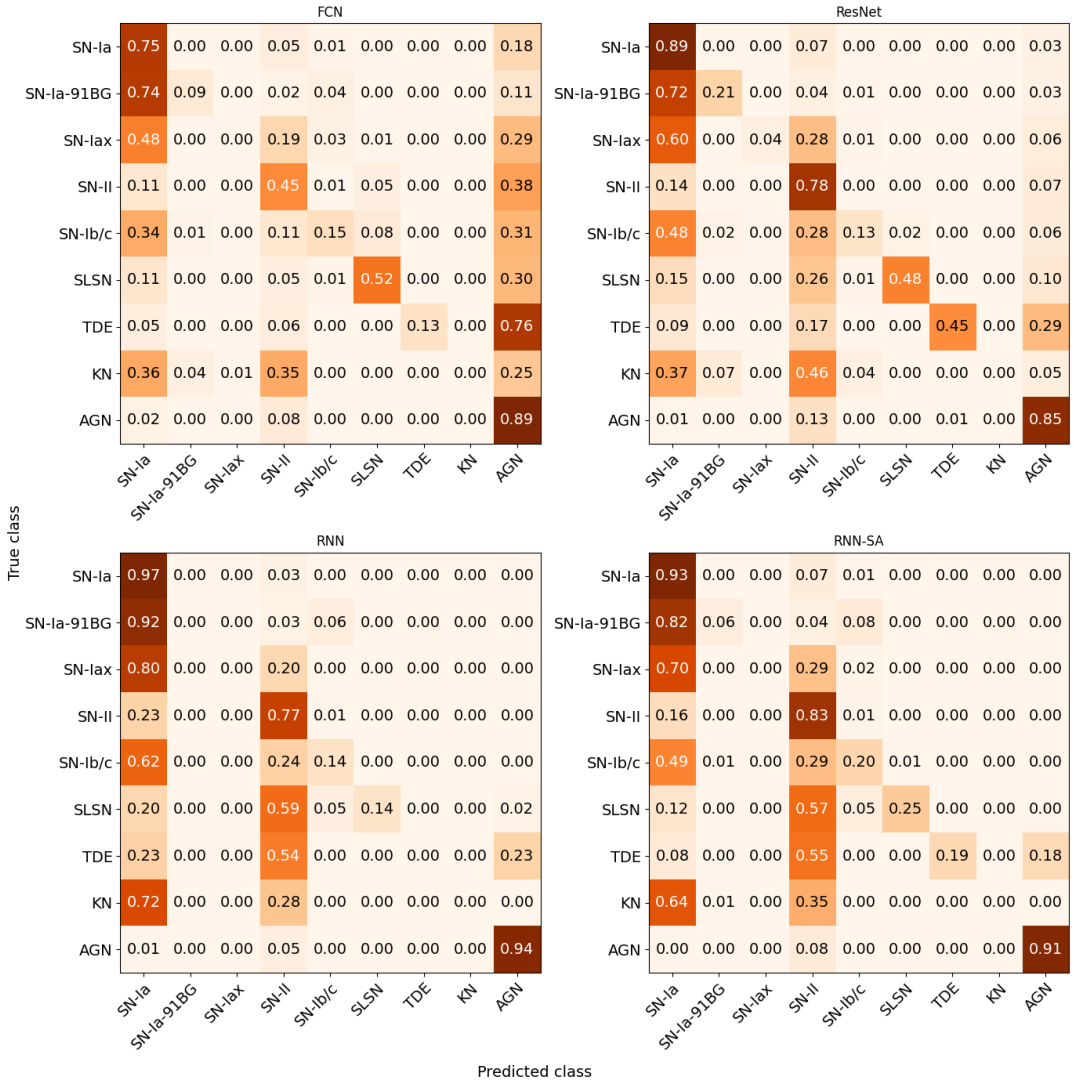


Figure 4.7 Confusion matrices when training with part of the test set for extra-galactic objects only.

some improvement, this is much more marked for the CNNs (ResNet and FCN), which appear to be more sensitive to the differences between training and test sets. Taxonomy again plays a major role, as the models are much better at distinguishing between explosive transients when presented with extra-galactic objects only. The model that improves in performance the most when training with a part of the test set is the ResNet. I think the main reason for this is the significant increase in data volume.

4.6.3 How does balance affect the test results?

We saw in last chapter that data balance influences performance significantly. To test whether this is also the case for the three baseline architectures and the proposed self-attentive architecture, they were trained with a balanced part of the original test set. Figure 4.8 shows the classification results for the case of extra-galactic objects only. We can see that the diagonal is much more marked.

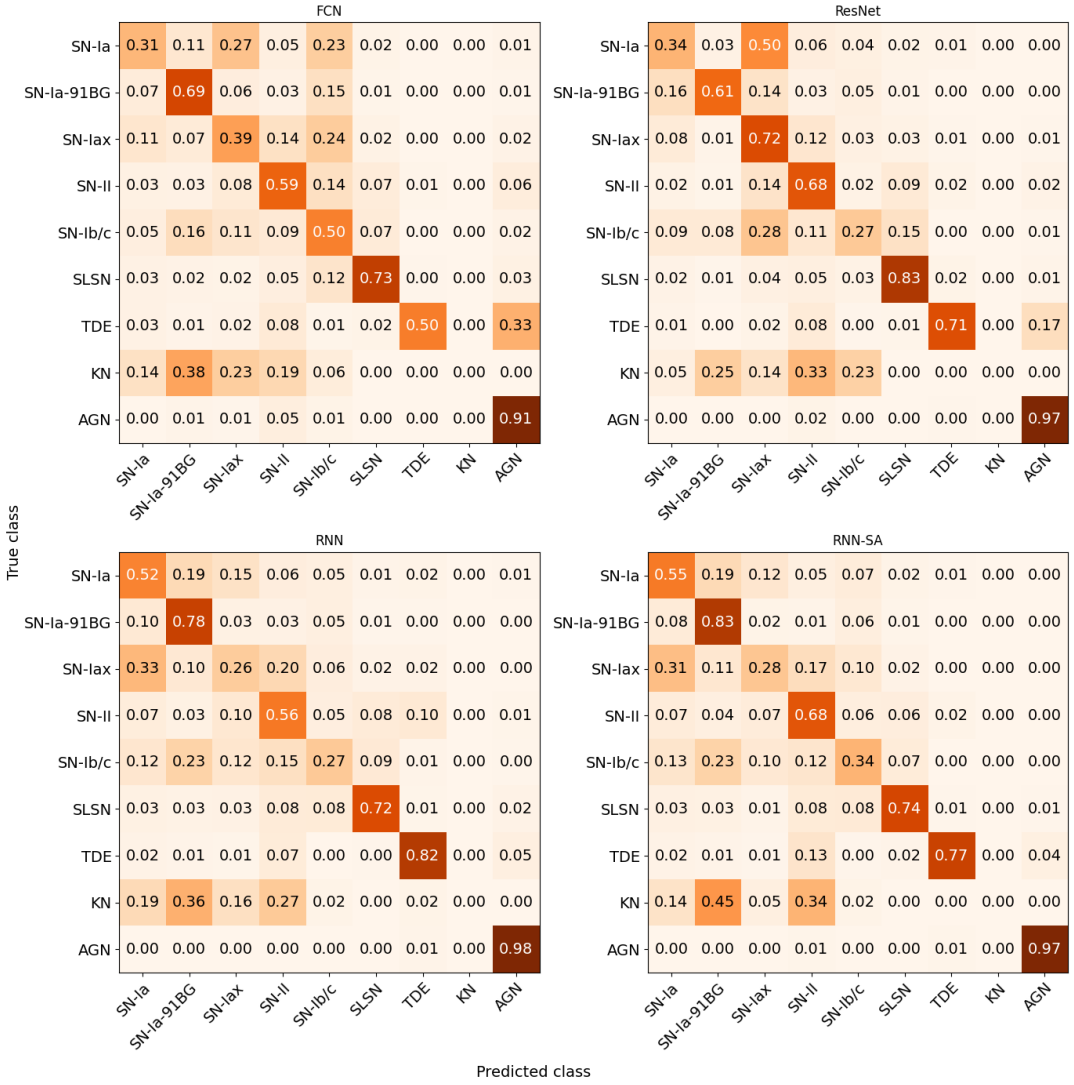


Figure 4.8 Confusion matrices for the test set results of all three baseline architectures plus the self-attentive architecture, when trained with a balanced dataset. Only classes that represent extra-galactic objects were considered.

4.6.4 Earliness in classification

Often when classifying SNe, we do not have light curves of the entire event, that is, the rise and fall in brightness. Given that self-attention is supposed to find relations within a sequence that serve as relevant features, maybe if we used it to pay attention to the flare of a SN, we could find in its rise something that could aid in its early classification.

With this in mind, I trained all four models with entire lightcurves, with 50% of the light curves and with 25% of the light curves. By this I mean I truncated the light curves. I did not attempt to center them around their peak brightness, since they are supposed to begin at the time of detection.

The confusion matrices representing the results of these experiments can be seen in Figure 4.9. All of the four models tried performed similarly well when presented with the complete light curve. However, when presented with less information (50% and 25% of a light curve) the self-attentive show a more marked diagonal.

The F1-scores corresponding to the case of 25% of the light curves are: 0.33 for the self-attentive model, 0.283 for the RNN, 0.238 for the FCN and 0.185 for the Resnet. While these are all low values, we need to keep in mind that this is for the case the networks got very little information. The self-attentive model does better in this case and it even manages to classify a portion of a class that the rest of the models completely miss (Kilo Nova) while also catching objects considered to be more rare, such as Tidal Disruption Events and Super Luminous Supernovae.

While these results look promising, I later realized there is an issue with the design of these experiments. For each case, I gave the classifiers a fixed size of light curves when training. In other words, when trying to classify light curves with half (a quarter) of the information, I gave the classifiers only half (quarter) light curves. While in previous research (Charnock & Moss, 2017b) it was shown that a classifier performs better on early light curves if it has only seen early light curves before (rather than entire light curves as well), I think this scenario is not necessarily practical. In a real life scenario, one would have light curves of different lengths and would expect shorter light curves to come in and be periodically updated. While it would indeed be possible to maintain several

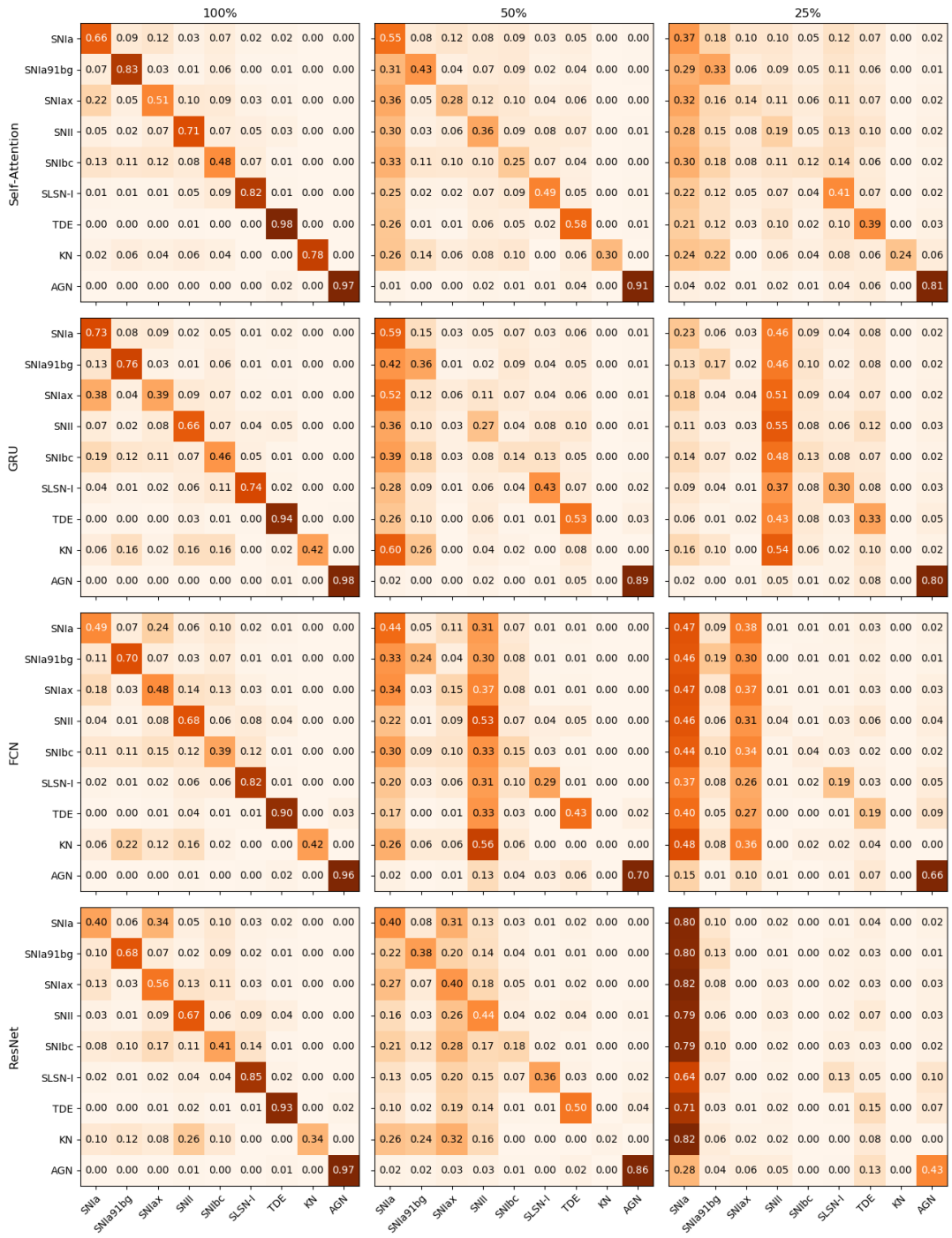


Figure 4.9 Confusion matrices for all baseline models and how they performed over the test set, when presented with 100%, 50% and 25% of a light curve.

models trained with different light curve lengths, or even an ensemble of them, this would require more resources.

I then checked how the models presented in Section 4.6.3 (that is, models

trained with a balanced dataset of entire light curves) performed when classifying incomplete light curves. The results can be seen in Figure 4.10 and Figure 4.11. In both cases performance dramatically worsens for all architectures except the self-attentive one, though it still shows worse performance. This tells us that for the models to be able to classify early light curves, they need to have seen incomplete light curves beforehand. This is consistent with existent research on early light curve classification (Charnock & Moss, 2017b).

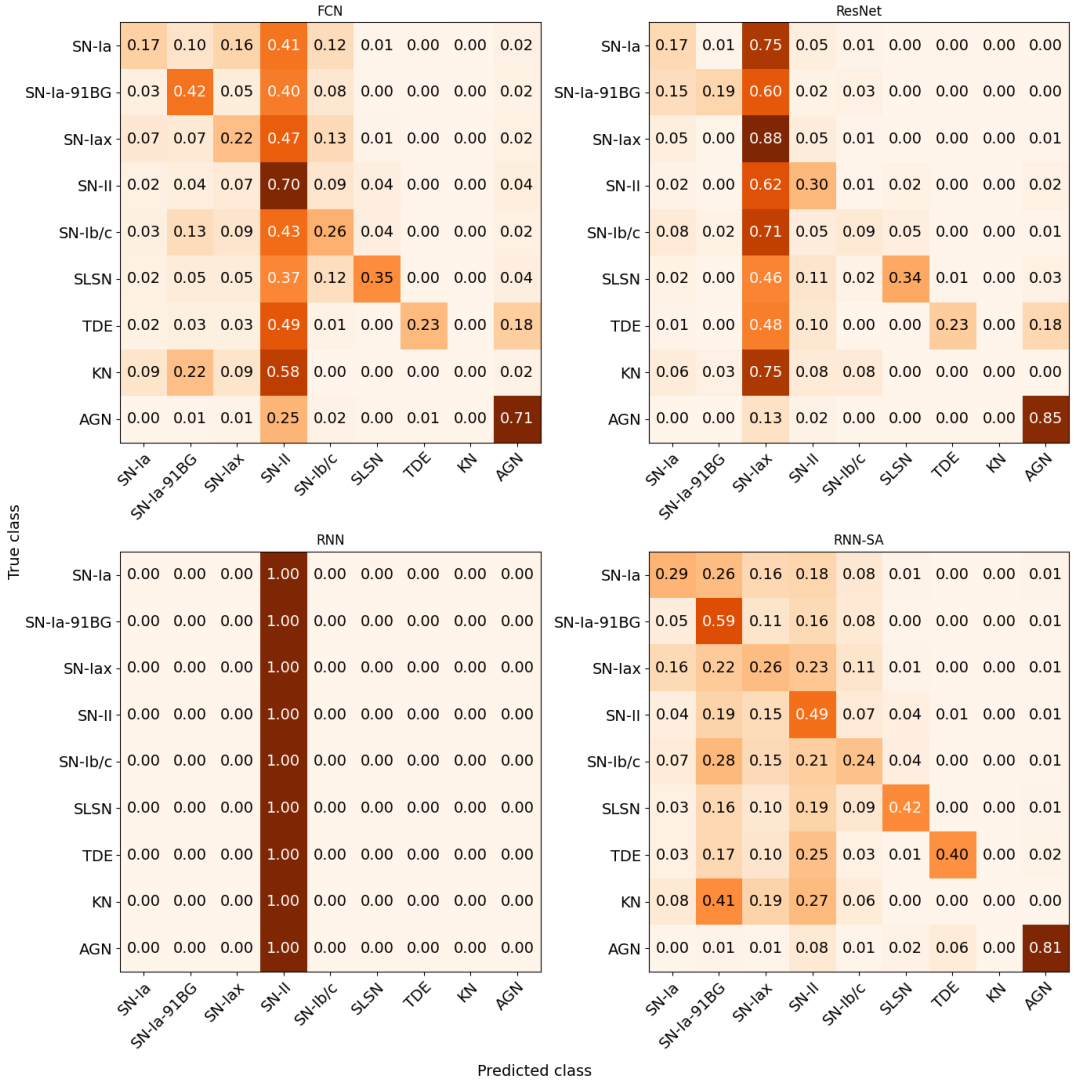


Figure 4.10 Confusion matrices for all models when trained with a balanced dataset, while classifying incomplete light curves (only 50%) of them.

In the following section we will try similar experiments on two different datasets: One simulating the ZTF survey and one made of real light curves from the same survey. This time, however, we will see what happens when we give the models

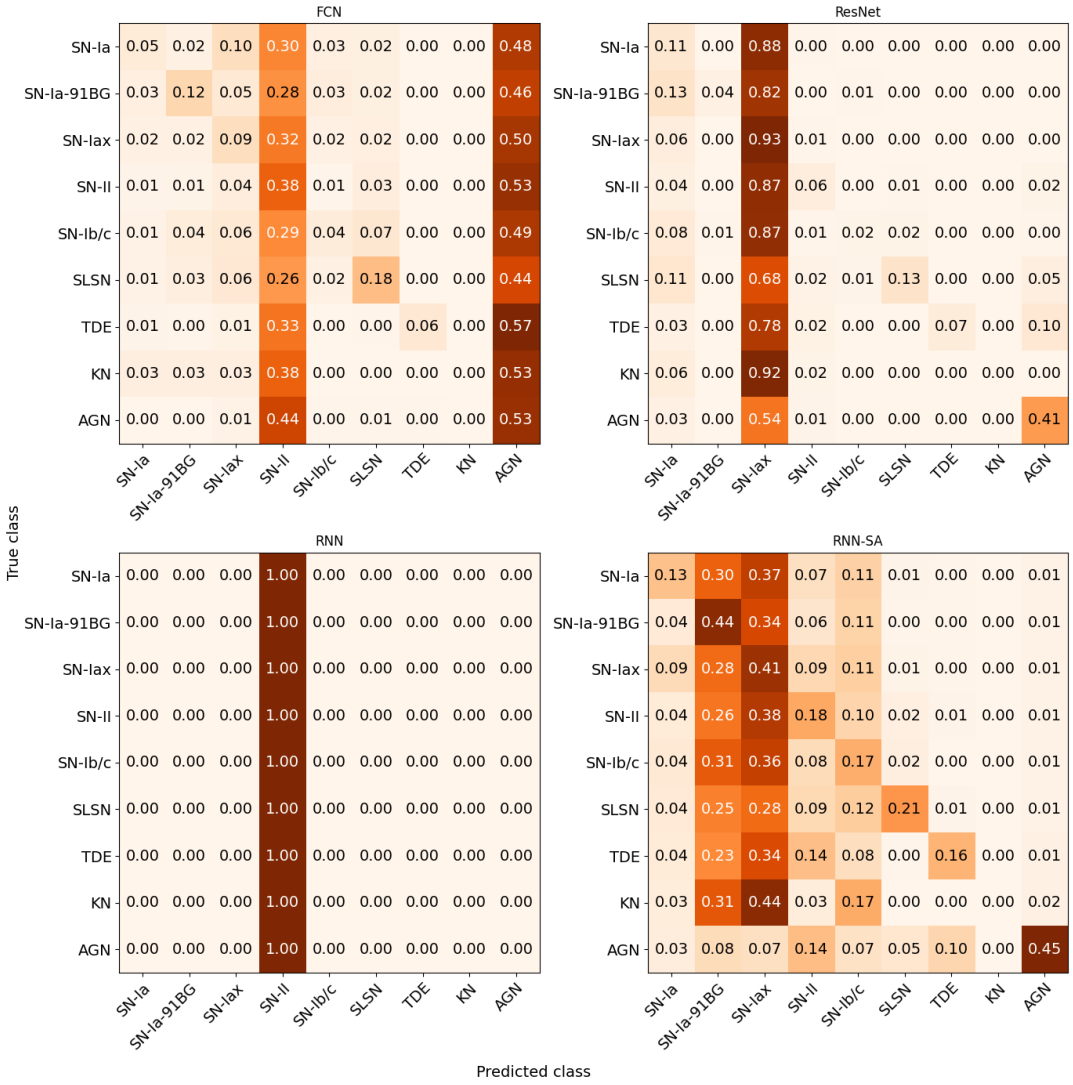


Figure 4.11 Confusion matrices for all models when trained with a balanced dataset, while classifying incomplete light curves (only 25%) of them.

light curves of different sizes.

4.7 Self-Attention for ZTF data

We saw that self-attention had potential to help in early classification of light curves, but up to this point, we have only tested this in simulated data.

In this Section, we will be using both simulated and real data for the ZTF survey. We will focus exclusively on SNe, since we saw that most of the confusion of the models concerns those objects.

4.7.1 Datasets

In order to be able to test how our classifiers perform on real ZTF light curves, it is first necessary to simulate a dataset that resembles that data, since there is not enough real data to train with as of yet (at least not a Neural Network). The basic idea is to use simulated light curves as training and validation datasets while reserving the real light curves to be used as test set. This will of course present some difficulties, as the real data is bound to be different to the simulations. It is however interesting to see how the algorithms perform against this gap, and whether there is some way to mitigate it.

4.7.1.1 The real data

Although the Zwicky Transient (ZTF) time-domain survey has been covering the northern sky visible from Palomar Observatory since 17 March 2018 and has produced as of now (September 2022) $\approx$ 4.3 billion light curves[1], it is difficult to obtain a reliable dataset that is labelled. There are several reasons for this: most of the labeled light curves are not spectroscopically confirmed, classifications done with machine learning are not necessarily correct and the interfaces to access the data are not standardized. Although there are several brokers dedicated to ingesting the ZTF alert stream (such as Lasair, Alerce, Mars, Fink, etc) their APIs for querying their databases differ from one another and they do not allow (to my knowledge) querying by type of object in bulk.

This issue led me to the TNS (Transient Name Server)[2], the official IAU (International Astronomical Uninon)[3] mechanism for reporting astronomical transients. Different groups both report and classify objects such as SNe and upload this information there. The TNS does allow queries by survey and type of object. This queries, however, yield metadata for any given object and accessing their bigger volumes of their light curves is not straight forward.

Through private communication I was also able to acquire the labels and metadata of $7,200$ SN light curves provided and classified by Alerce [4].

I resolved the issue of obtaining labeled SNe light curves by first querying the

[1] https://www.ztf.caltech.edu/ztf-public-releases.html
[2] https://www.wis-tns.org/
[3] https://www.iau.org/
[4] https://alerce.science/

TNS by survey and type, to obtain the ZTF IDs of each object. Once all the necessary IDs where collected, I used both the TNS IDs and the ones provided by Alerce to scrape the websites of different brokers and obtain the corresponding light curves. Although I could have used the APIs of said brokers to query by ID, I found this method to be easier, since it allowed me to query several IDs at the same time and was also agnostic to the broker. The code and dataset obtained from the process are publically available[5]. I collected from this procedure the light curves of $\approx 12,200$ objects ($7,200$ of those labeled by Alerce and $\approx 5,000$ with labels reported at the TNS), reported from 2018 to 2022.

The labels, however, are not always reliable and sometimes they differ depending on the group that classified them. The ones provided by Alerce included a classification probability for different classes. After checking several of those objects in different brokers and seeing that their labels were sometimes different from the ones I was given (even in the Alerce broker itself, where sometimes labels had been updated), I decided to use only the objects that had a classification probability greater than 0.5. This reduced the objects from $7,200$ to only $1,133$.

For all objects (both the objects labeled by Alerce and the objects labeled by other groups in the TNS) I decided to include them in the dataset only if they had at least three detection points in both public bands of the ZTF survey (the g and r bands) and at least a month of observations. This reduced the Alerce objects to $1,084$ and the TNS objects to $2,066$, for a total of $3,151$ objects. Figure 4.12 shows the amount of objects that each group classified, as reported in the TNS. Note that in Figure 4.12, the total number of classifications does not add up to the total number of objects used. This is because several objects were labelled by more than one group.

For simplicity, I decided to use the SN types used in the PLAsTiCC challenge, even though in the process I had collected some other type of objects (such as SNe-Ia91T, SNe-IaCSM, AGNs and TDEs). The final datasets is made of six types: SN-Ia, SN-Ia91bg, SN-Iax, SN-II, SN-Ib/c and SLSN. The number of objects per class can be seen in Figure 4.13.

[5]I need to upload this to github and insert the link here

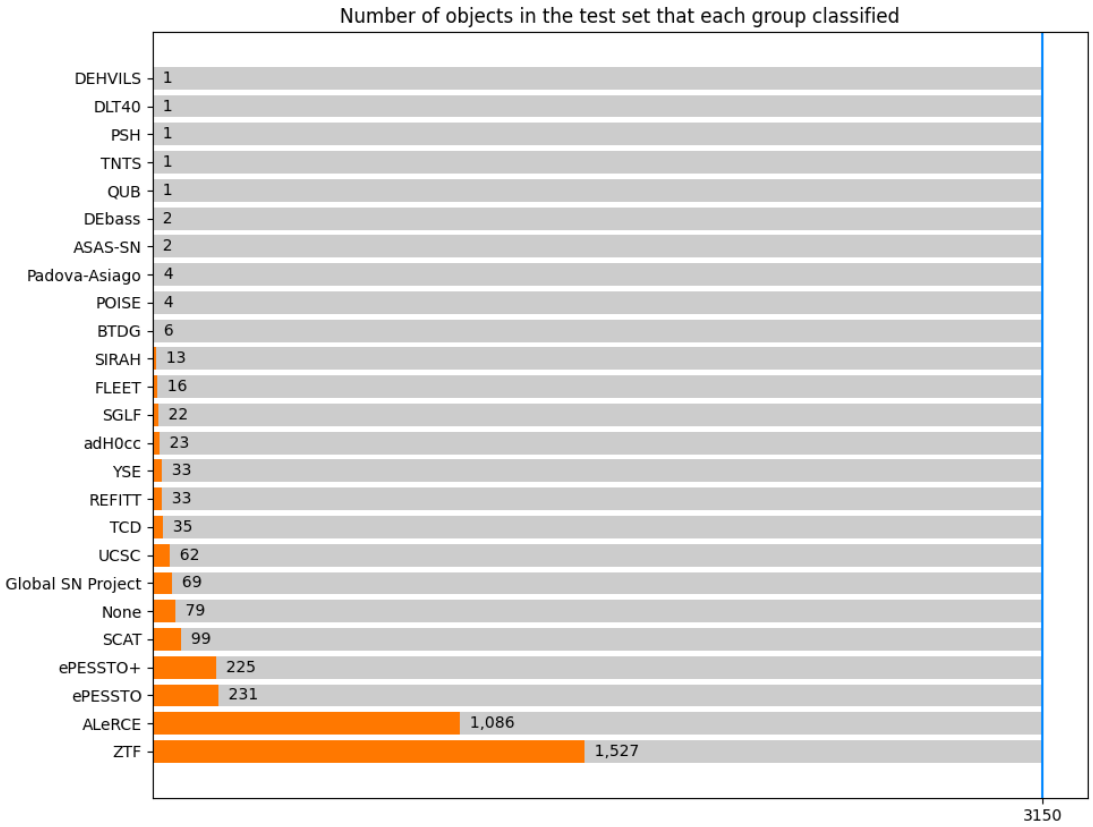


Figure 4.12 Number of objects that each group reported in the test set constructed using real ZTF data. The orange bars present the number of objects, while the grey bars represent the total number of objects in the dataset. The blue vertical line marks the total number of objects in the dataset.

4.7.1.2 Light curves simulated using Simsurvey

In order to create a training set with enough samples, I used a python package called Simsurvey [6]. This simulation library is built upon the well known library SNCosmo [7], which uses several different templates and models to simulate photometric data. It also provides built in magnitude systems, bandpasses and sources. Simsurvey, on the other hand, simulates a survey from an observation plan and a transient object, for which a volumetric rate needs to be specified. The survey plan needs to include a time range (in MJDs), a redshift range, the observation fields used and information about the CCDs. Dust maps also need to be given to simulate both Milky Way and host galaxy dust extinction. Simsurvey yields light curves in arbitrary units which represent flux differences.

[6] https://simsurvey.readthedocs.io/
[7] https://sncosmo.readthedocs.io/

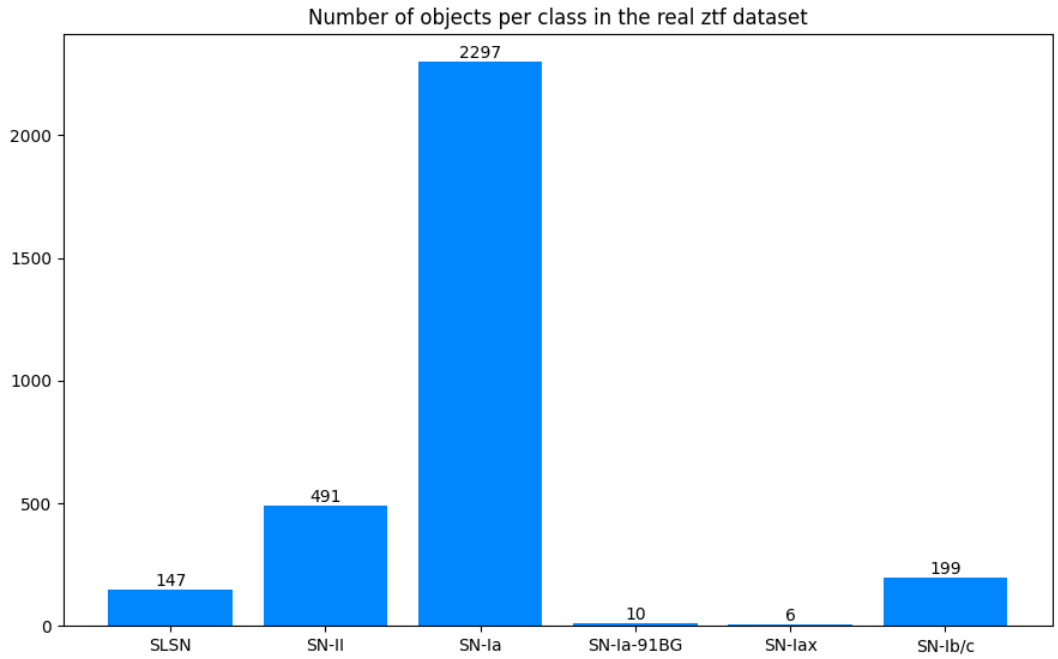


Figure 4.13 Number of objects per type in the test set constructed using real ZTF data.

I used two observation plans for the simlations. The first one was put together by the creators of Simsurvey, in an article where they investigate simulation rates (Feindt et al., 2019). This survey plan does not coincide with the actual ZTF simulation schedule, although many fields overlap with it. The second survey plan I used is based on the ZTF observing log database. This log considers both the Northern Sky (NSS) and the Galactic Plane (GPS) surveys. These are known as ZTF-MSIP surveys (Bellm et al., 2018). I found this plan in a public repository [8] and it was produced using the scheduler library ztfsim [9].

The survey schedule produced by the creators of Simsurvey contemplates two types of surveys: a public schedule which observes the whole visible sky at a three-day cadence and a high-cadence survey which observes$\approx 1,600\text{deg}^2$ with up to six g band observations per night that are separated by at least 30 minutes. Although this plan considers three bands (r,g and i), I used the public bands (g and r) only. In Figure 4.14 we can see a visualization of this plan.

The survey schedule based on the ZTF observing log, on the other hand, considers $\approx 13,000\text{deg}^2$ of the Northern sky at a three-day cadence and $\approx 1,500\text{deg}^2$ of the galactic plane at a one-day cadence. In Figure 4.15 we can see a visualization

[8] https://github.com/gnarayan/ztfsimlib
[9] https://github.com/ZwickyTransientFacility/ztfsim

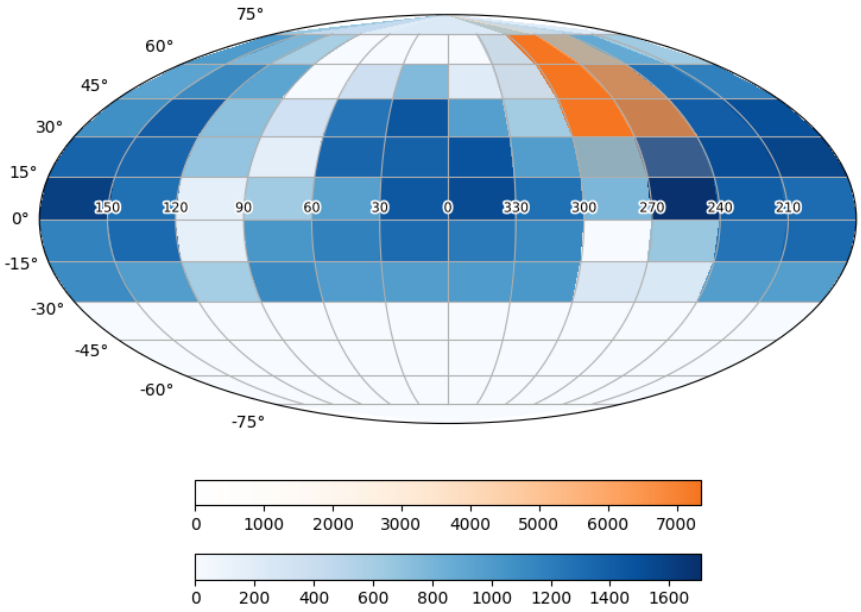


Figure 4.14 Survey plan obtained from the creators of Simsurvey. Blue tiles represent the public survey, while orange tiles represent a high-cadence survey, superimposed. The color intensity represents the number of observations per tile during the time period considered.

of this plan. Note that in both Figure 4.14 and Figure 4.15 each tile does not represent an observation field, but several of them. This is so purely for visualization purposes.

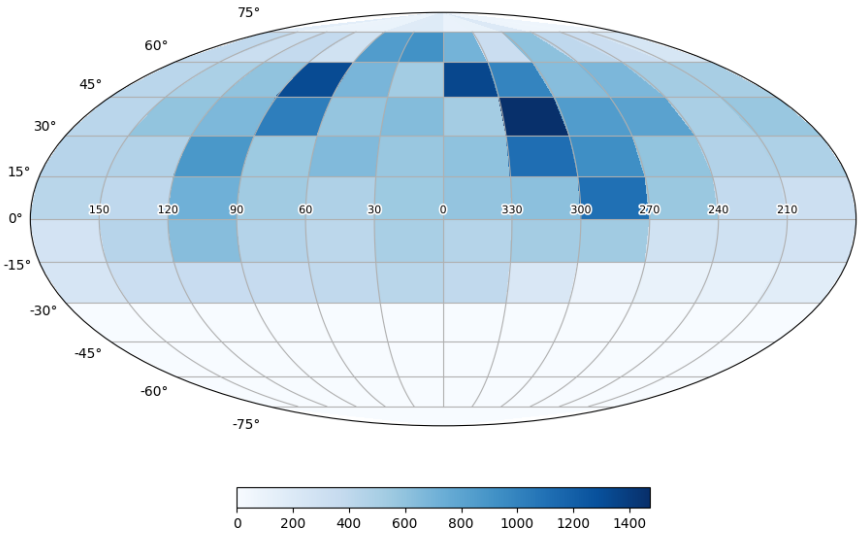


Figure 4.15 Survey plan obtained based on the ZTF-MSIP surveys. Blue tiles represent the public survey and the color intensity represents the number of observations per tile during the time period considered.

To simulate extinction by galactic dust, publicly available dust map files [10] [11] (Schlegel et al., 1998) (diplayed in Figure 4.16) were used when generating the light curves. Additionally the extinction law of Cardelli et al. (1989) was used to account for extinction by dust in the host galaxy.

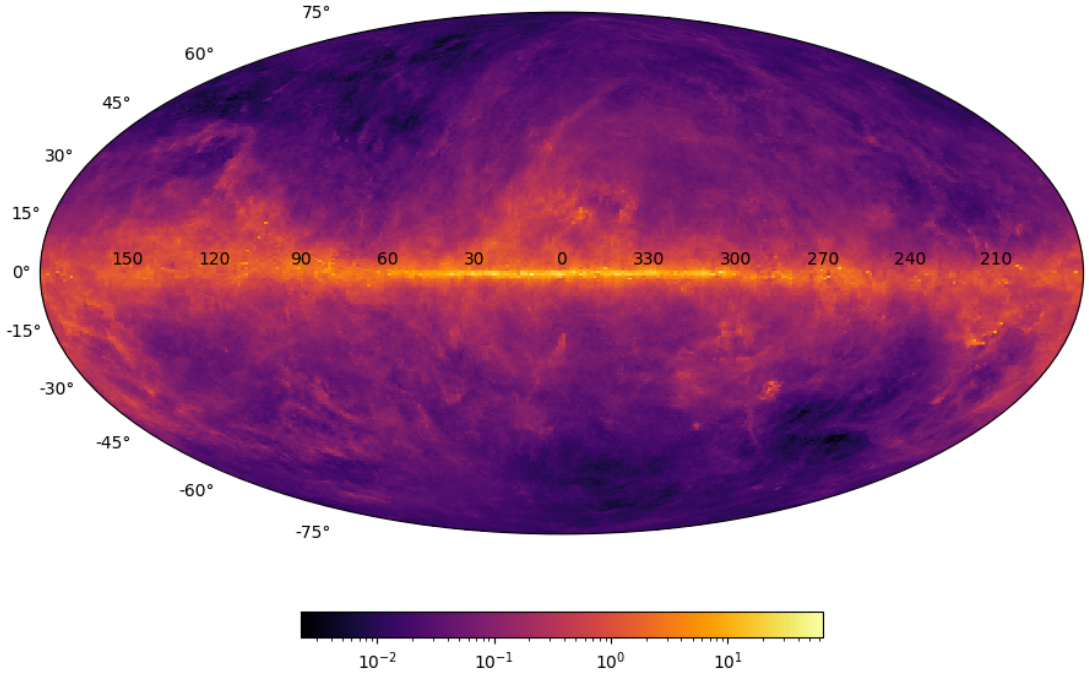


Figure 4.16 Milky way dust map.

While Simsurvey provides built-in templates (extracted from SNCosmo) for several sources (SNe-Ia, Ib/c, IIn and IIP), it lacks templates for the other types of SN used in these simulations. For SNe-Iax, SNe-Ia-91g and SLSNe, I used the templates developed for the PLAsTiCC challenge. I used a redshift range of $[0.01 - 0.2]$, since from the real dataset described above, 99% of the objects either have known z or known host z that fall within that range.

Instead of using the built-in volumetric rates provided by Simsurvey, I mostly used the ones used for the PLAsTiCC simulations (Kessler et al., 2019). For SNe-IIP and SNe-IIn (which were simulated separately but are later in the experiments merged into one class) I changed the volumetric rates so they were at $z = 0$ similar to the ones built-in in Simsurvey, but made them dependent of z instead of just being constant. I did this because SNe-IIP would not appear often enough in the simulations if I kept the rates given in the description of the PLAsTiCC models, probably because they are fainter. Changing this made the number of

[10] https://github.com/kbarbary/sfddata
[11] https://github.com/kbarbary/sfdmap

Object Type	Rate($\mathrm{yr}^{-1}\mathrm{Mpc}^{-3}$)	M_V	σ_V	Template
SN-Ia	$2.5 \times 10^{-5} \times (1+z)^{1.5}$ $\star$	-19.3	0.1	Salt2, Hsiao$^{\triangle}$
SN-Ia-91bg	$3 \times 10^{-6} \times (1+z)^{1.5}$ $\star$	-17.5	0.3	Nugent$^{\bullet}$
SN-Iax	$\texttt{md14}(z_0 = 6 \times 10^{-6}, z)$ $\bowtie$	$-13\mathrm{to}-18$	-	RutgersSN$^{\diamond}$
SN-Ib/c	$\texttt{s15}(z_0 = 1.9 \times 10^{-6}, z)$ †	-17.5	1.2	Nugent$^{\bullet}$
SN-IIP	$\texttt{s15}(z_0 = 1.4 \times 10^{-4}, z)^{\dagger}$	-16.5	1	Nugent$^{\bullet}$
SN-IIn	$\texttt{s15}(z_0 = 7.5 \times 10^{-6}, z)$ †	-18.5	1.4	Nugent$^{\bullet}$
SLSN	$\texttt{md14}(z_0 = 2 \times 10^{-8}, z)$ $\bowtie$	-21.5	0.7	MOSFiT‡

Table 4.2 Rates, absolute magnitudes and templates used for the simulations.
$\star$: z dependence as defined by Dilday et al. (2008).
$\bowtie$: z dependence as defined by Madau & Dickinson (2014).
$\dagger$: z dependence as defined by Strolger et al. (2015).
$\triangle$: based on Guy et al. (2007) and Hsiao et al. (2007).
$\bullet$: based on Nugent et al. (2002), Levan et al. (2005) and Gilliland et al. (1999), respectively.
$\diamond$: based on Jha (2017)
$\ddagger$: based on Nicholl et al. (2017)

objects generated closer to the statistics reported by the BTS (Bright Transient Survey) [12] (Fremling et al., 2019). Table 4.2 shows a summary of the rates, templates, and peak absolute magnitudes used. All templates used are publically available.[13][14][15]

A single simulation would create around $\approx 30,000$ SN events. However, I applied the following criteria to filter the events:

1. They had to have at least two 5σ detections in the same night, which corresponds to the discovery process of transient surveys.
2. They did not appear at lower Galactic latitudes ($|b| \leq 7\,\mathrm{deg}$).
3. They had a peak magnitude of at least 19 in both bands.
4. I removed all the points that happened before and after *days* from the time of explosion.

[12] https://sites.astro.caltech.edu/ztf/bts
[13] SN-Ia: https://sncosmo.readthedocs.io/en/stable/source-list.htmlrb0467a492107-n02
[14] SN-Ia-91bg, II b/c, IIP and IIn: https://c3.lbl.gov/nugent/nugent_revision.html
[15] SLSN: https://mosfit.readthedocs.io/en/latest/models.html

5. I removed all the points that had high uncertainty (where their uncertainty was more than 3σ from the mean uncertainty)

6. The objects had to still have at least three observations per band before peak magnitude.

This reduced the number of objects in one simulation to ≈ 900. A visualization of the SN events simulated in one run can be seen in Figure 4.17.

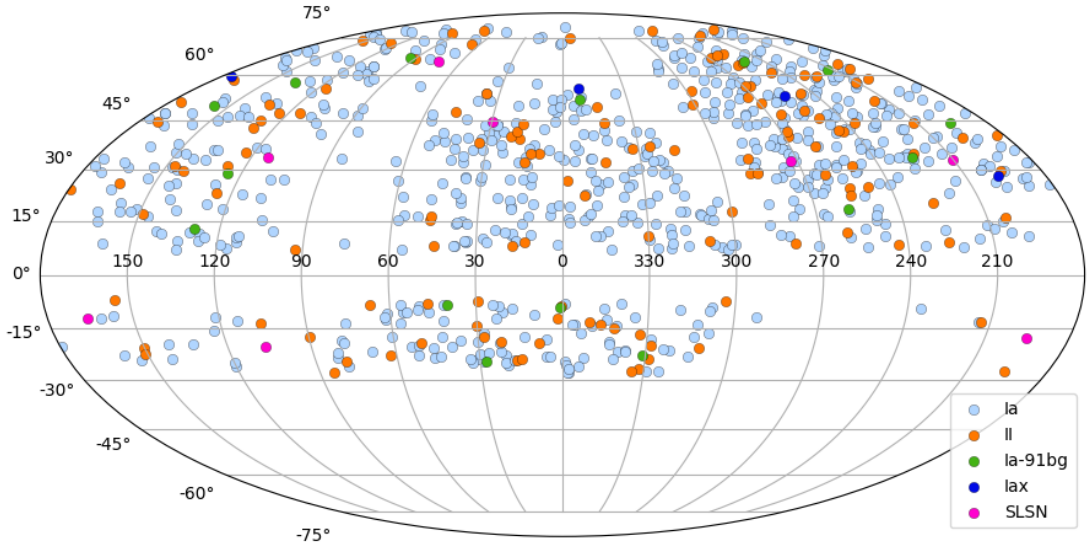


Figure 4.17 Simulated SNe in one run, per type.

As we saw in the previous chapter and in Section 4.6.3, supervised deep learning algorithms are greatly affected by highly unbalanced training datasets, as they tend to classify most objects as the class that appears the most. For this reason, and in order to increase the volume of the dataset, I repeated the simulations several times. The number of repetitions changed depending of the object type, that is, the objects with lower rates required more runs. I obtained a balanced dataset of $16,800$ samples. The code for the process is publicly available[16].

4.7.2 Experiments

To start off, it is necessary to see whether the baseline architectures work at all with the simulated ZTF-like data. I first tried training the models with the balanced simulated data set and tested them both with simulated and real data. The results are visualized in Figure 4.18 and Figure 4.20 respectively.

[16] https://github.com/aibsen/lc_sim

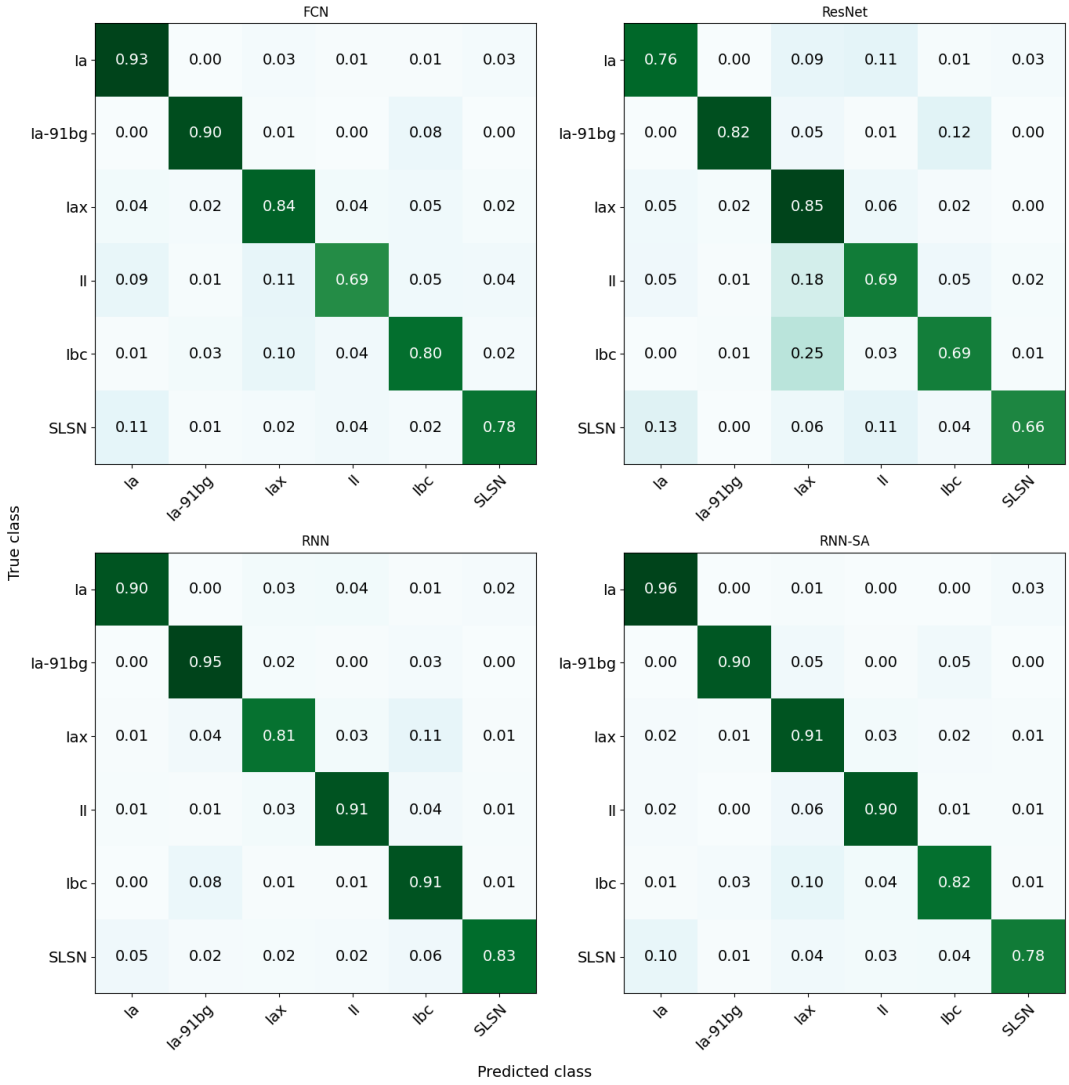


Figure 4.18 Confusion matrices for the results of all four models. The test set in this case is entirely made of simulations

In Figure 4.18 we immediately see that the models have no problem differentiating between the different types of simulated objects. All models exhibit a equally good performance.

The models, however, do not converge in the same way. In Figure 4.19 we can see how the F1-scores change as the training epochs increase for each model. It is immediately obvious that the training process of the CNNs is much less stable. I think this is because they both have significantly fewer weights, so each change would affect the overall performance more. Since we are stopping the experiments at an epoch with a good F1-score over the validation set, this should not be an issue, thought perhaps it indicates that scores will vary more between different experiment runs.

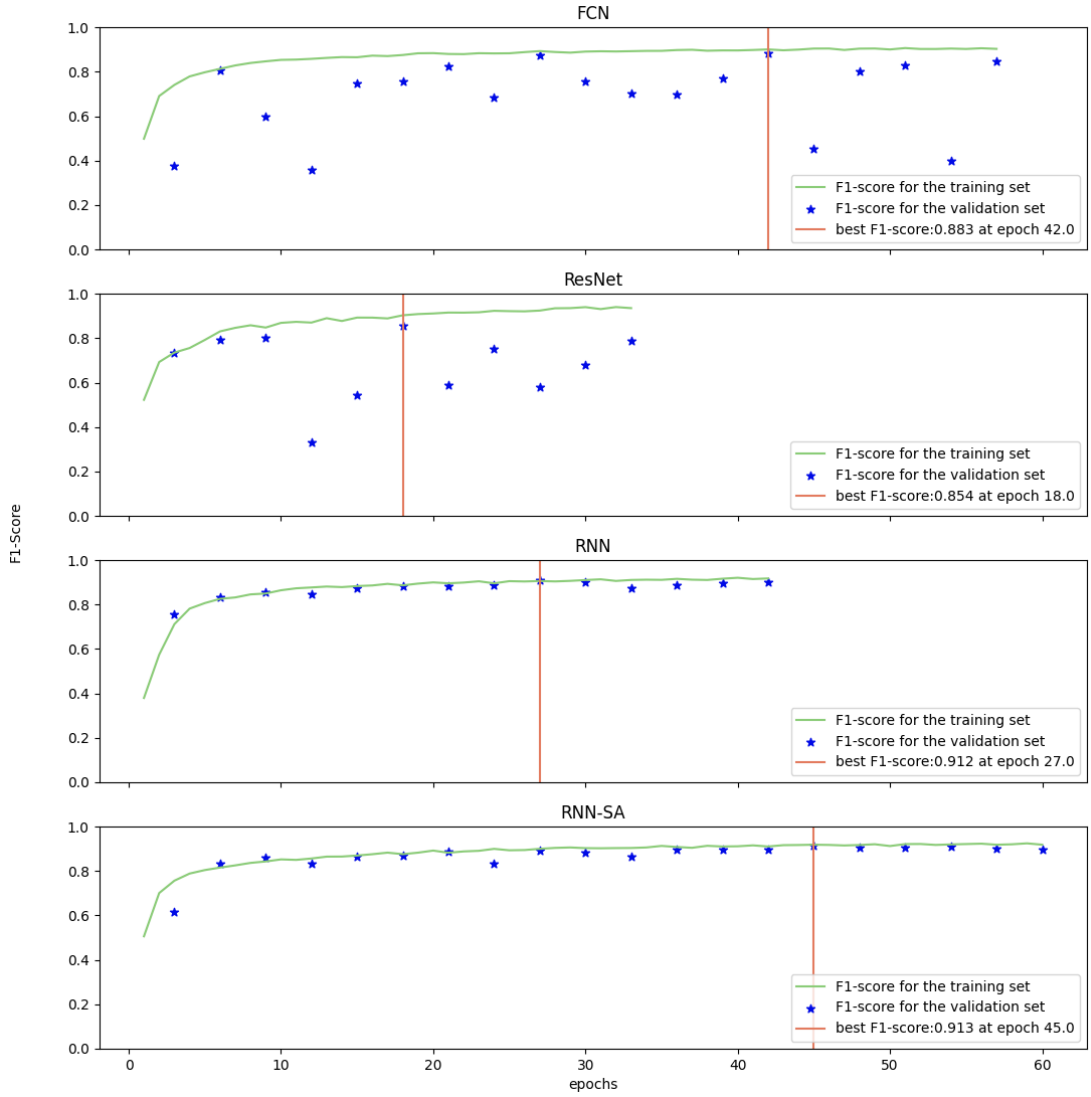


Figure 4.19 Confusion matrices for the results of all four models over a real ZTF data. The classifiers were trained with a simulated balanced dataset.

In contrast to the results over the simulated dataset, in Figure 4.20, we see that the classifiers get confused and try to label most samples as one specific class, specially the FCN. The Self-Attentive architecture performs visibly better, although it by no means obtains a good performance.

4.7.3 Experiments centered on earliness of classification

We saw in section 4.6.4 that the models unsurprisingly benefit from seeing beforehand light curves that have the same length as the light curves they are going to classify, however, the case where the models will only see light curves

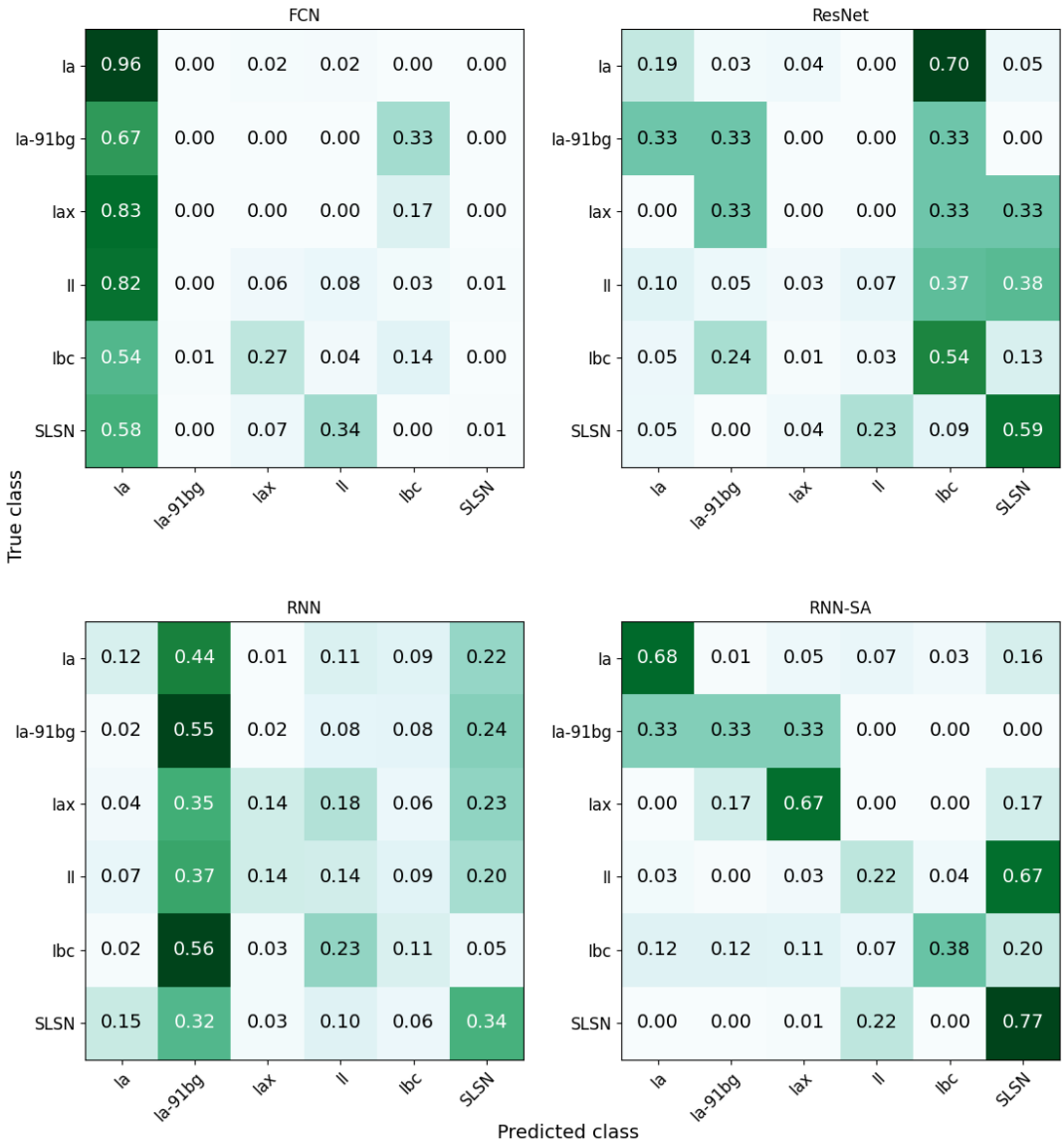


Figure 4.20 Confusion matrices for the results of all four models over a real ZTF data. The classifiers were trained with a simulated balanced dataset.

of a particular length would require using extra resources to train more models. Because of this, in this Section the models were trained with light curves of several different lengths: 100%, 50% and 25% of the light curves. 50% of the training set consisted of entire light curves, as to give the models as much data as possible. The remaining half was split into two. While this might cause the later part of the sequence to influence the network too much and thus worsen early classification performance (Charnock & Moss, 2017b), it is both a compromise on computational time and a more realistic scenario. I would be interesting, however, to study in the future how the length of the light curves given during

training affects the performance of the models and the effect it has when attention mechanisms are involved. Figure 4.21 shows the results of these experiments.

While the Self-Attentive architecture performs better than the other models, they all obtain similar F1-scores (and accuracies and losses) for the cases where the test set is made of entire or half light curves. When the test set contains only light curves truncated to 25% of their length, however, things change.

The models with the best performance are the FCN and the RNN-SA, the last one with slightly higher scores. I think this might be because both these architectures are supposed to extract feature maps that act as a representation of the relevant relationships between different sections of the light curves. The ResNet also does this, though with more steps, and indeed it shows very similar performance.

The RNN with GRUs and without attention is by far the one with the worst performance. Rather than it being a fault of the architecture itself, I think the reason for this are the practical details of its implementation. The models do not update their weights after they see a single sample, but after they ingest a batch of them. The RNN and RNN-SA both have an structure that focuses on temporal patterns, computing different states for each time-step in the sequences (light curves, in this case).

Ideally during training they would use only the existing parts of the sequences, but in practice, they use the same length for each sample in the same batch. The way I circumvented this issue was by simply using the maximum length in each batch for its entirety. This means that sometimes the RNNs will ingest parts of sequences that are empty (filled with zeros, in this particular implementation).

While the Self-Attentive model takes the outputs of the hidden states of all time-steps and gets a representation from them, the RNN uses the time-steps directly for the final classification. This means that the final fully-connected layer will ingest irrelevant parts of light curves that are likely detrimental to the overall performance of the model. It seems that adding an Attention layer to an existing RNN does help in early classification.

These results look promising, but things get complicated when we look at the classification of real light curves.

In Figure 4.22 we see that there is great confusion when the models try to classify real data. The FCN now label samples as three classes and ignores the less

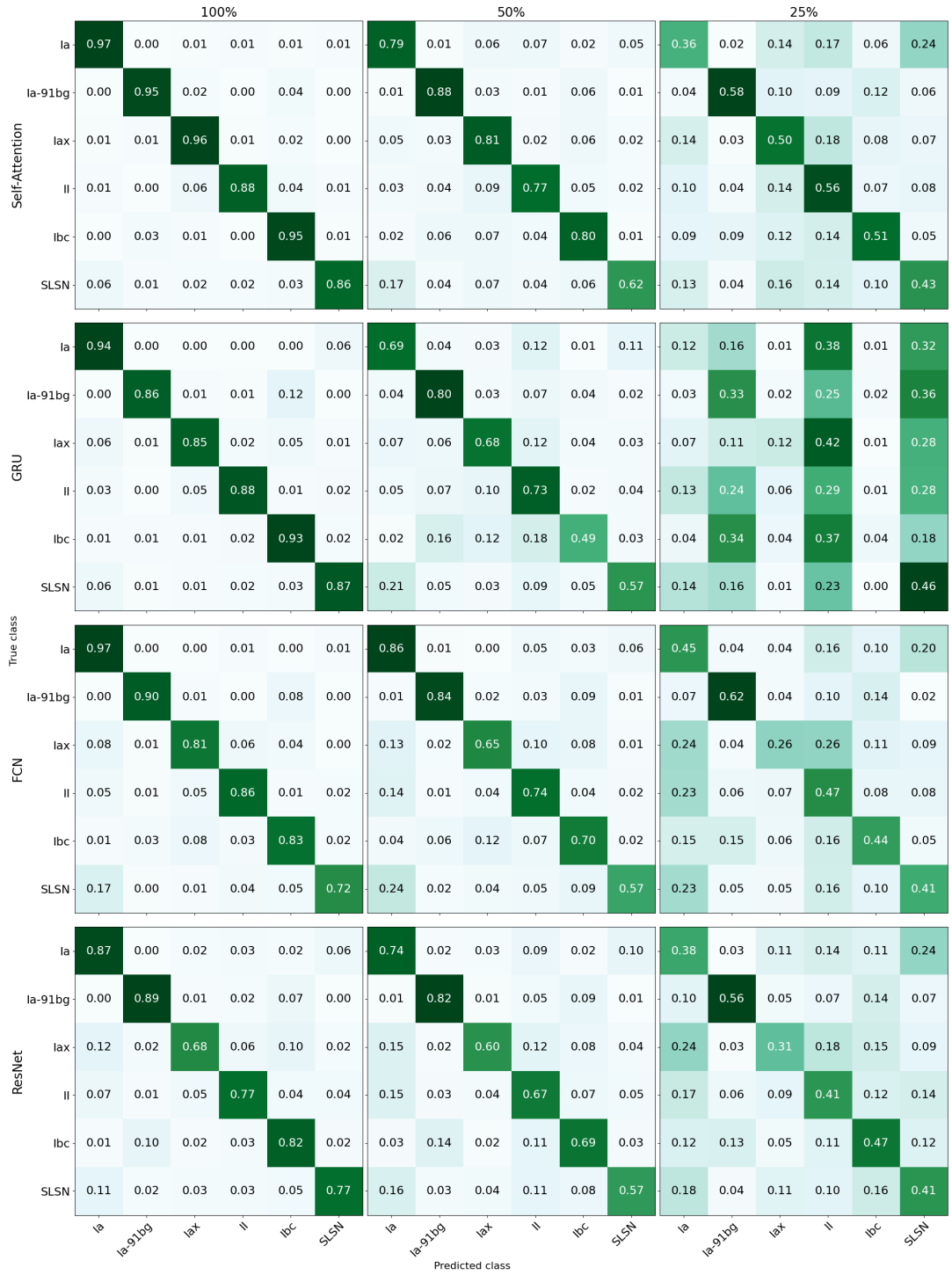


Figure 4.21 Confusion matrices for the results of all four models over a simulated test set. Each row represents a model and each column shows how much of the light curves was given to it for classification. The classifiers were trained with light curves of different sizes.

prominent ones. While the ResNet does manage to catch most SNe-Ia and about half of the SLSNe, it does not make sense of the other classes. The RNN is even

more confused than before. The Self-Attentive RNN, however, does surprisingly well. Although it mostly catches SNe-Ia and SLSNe (like the ResNet), its diagonal is much more defined. This architecture clearly benefited from seeing early light curves.

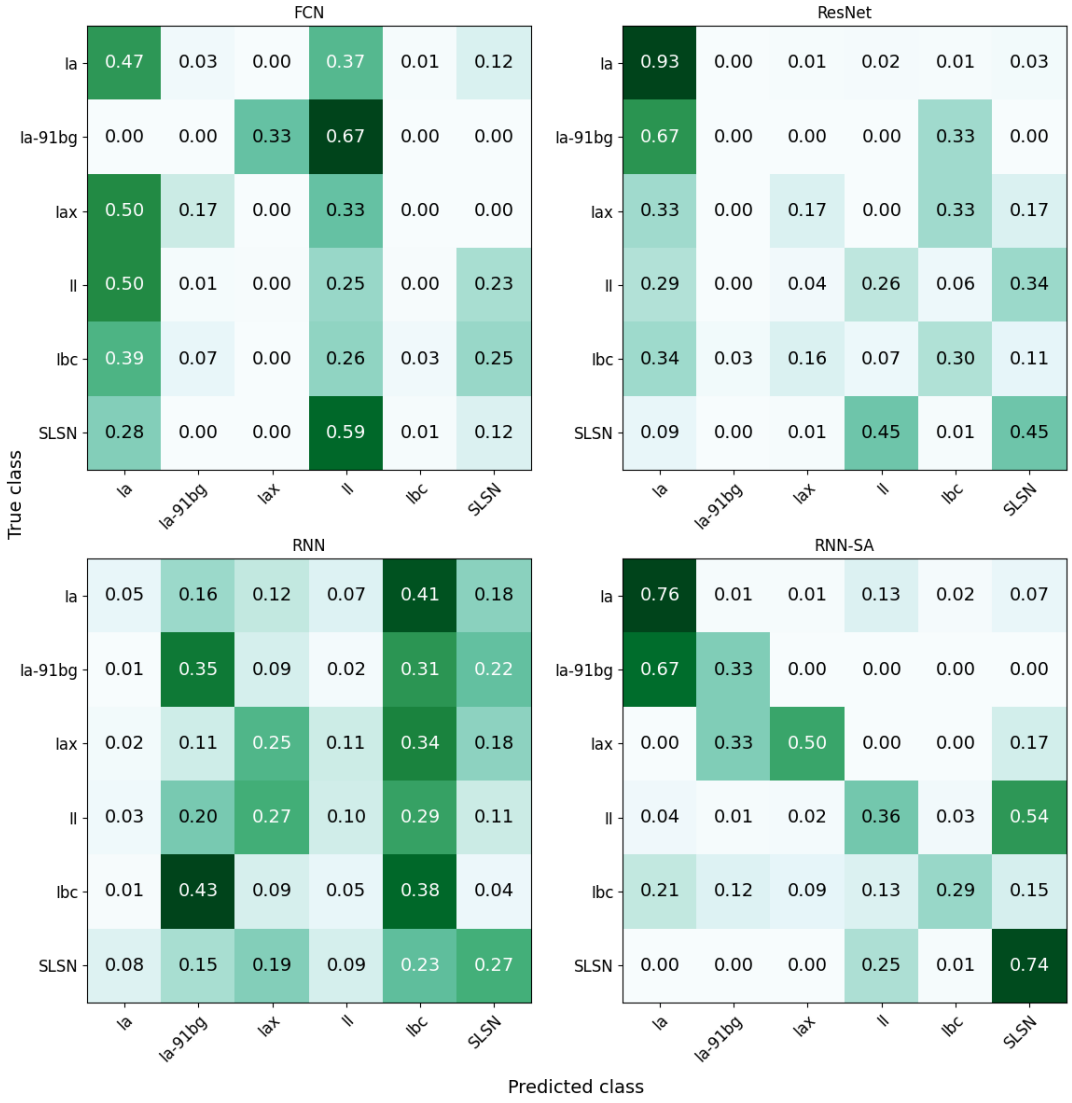


Figure 4.22 Confusion matrices for the results of all four models over real ZTF data. The classifiers were trained with light curves of different sizes.

4.8 Discussion

From the experiments performed and the results obtained we can get some insight as to what matters and what does not when trying to classify transients early on.

First, we saw that when using the simulated PLAsTiCC dataset, the self-attention mechanism seems to help in the classification of light curves when there is only a few observations, but does not really make a marked difference for the task of classifying entire light curves, where all models tried performed similarly well. When using a self-generated ZTF-like simulated dataset, all the classifiers perform well when presented with entire light curves. Although the confusion matrices in Figure 4.21 show very similar performance, it is important to mention that these show the best results for each experiment. Things look different when we consider several different runs of the same experiments, as we can see in Figure 4.23. The top Figure shows the histogram of 15 different experiment runs for the case of classifying entire light curves. It is clear that the RNNs have a more consistent performance and that the RNN-SA outperforms the other models. When trying to classify light curves early, we find that all classifiers show a better performance when they have previously encountered light curves of varied lengths during the training process.

When classifying real data, however, performance dramatically drops, as we can see at the bottom of Figure 4.23. Again the RNN-SA performs better, but its F1-scores are still low. Since we are calculating the F1-scores as the mean of the F1-score of each separate class, it is not surprising that they are low, since the model mostly catches SNe-Ia and SLSNe. Since the aim of Self-Attention is to encode relevant relations within a sequence, it may be that the model is able to make sense of prominent features of these two categories. SNe-Ia are known to have a luminosity-width relation, for example.

It seems to be that the task of light curve classification and even early light curve classification is not especially problematic provided there exists a training data set that resembles closely the type of data that is being classified. All the baseline models seems to perform similarly well when the data used for training and the data used as test set comes from the same data set, but fail otherwise. The only model that is able to generalize a bit better is the Self-Attentive one.

There are several possible reasons for this. The most obvious is that the simulated dataset is not representative enough of the real data. Perhaps I should have been more careful when degrading the data and selecting the transients to use. I purposefully did not want to correct dust extinction or z time dilation to see whether the models would be able to tell apart objects without that information. Since simulated objects with different z are correctly classified with high F1-scores, I think this might be possible and indeed it is a matter that has been

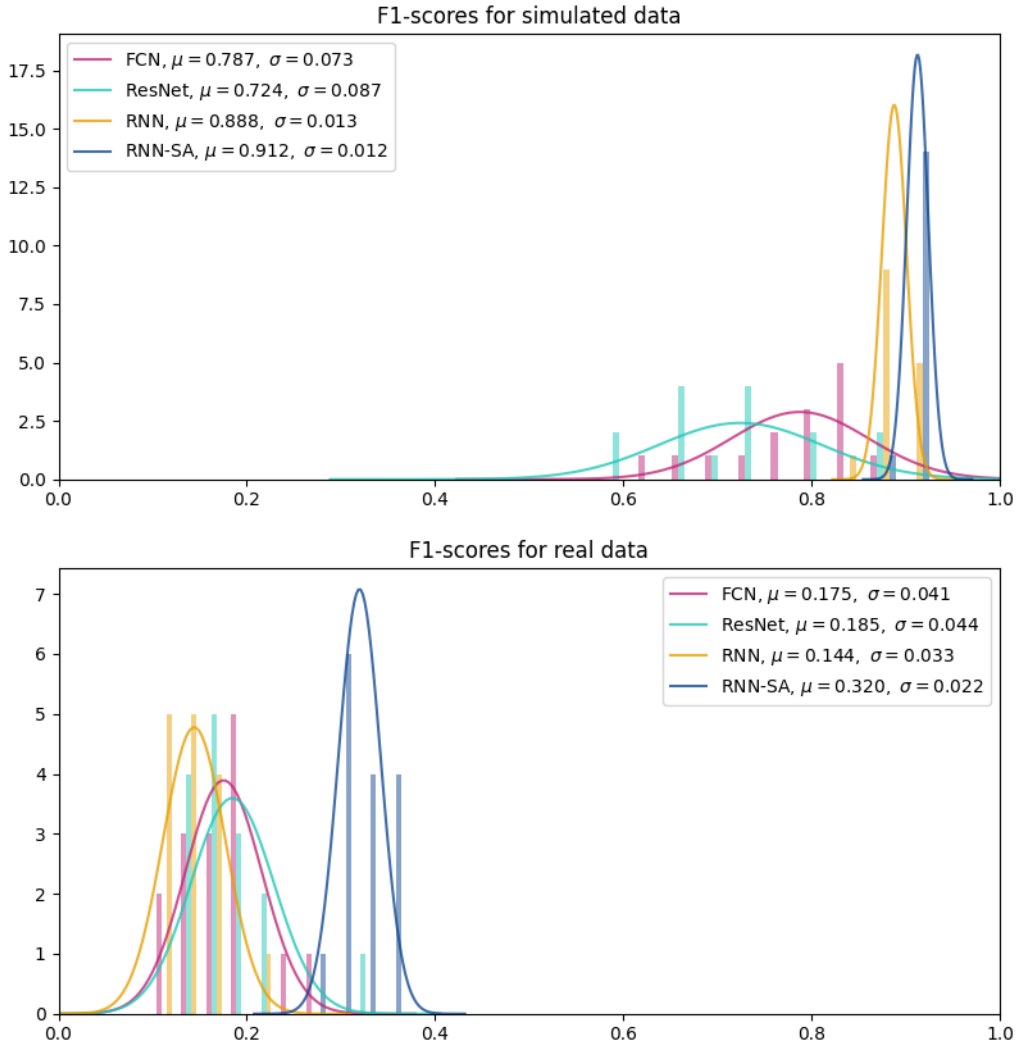


Figure 4.23 Histogram of the F1-scores obtained for the case of classifying entire simulated light curves (top) and real light curves (bottom).

debated.

I think one major issue was the small number of templates I used when simulating objects that are prominent in the real data test set, particularly SNe-Ia and SNe-II. While this would help the models to classify the simulations themselves, it is probably detrimental to the classification of real light curves. In a future version of these simulations, more templates should be considered instead of just using the built-in ones that SimSurvey and Sncosmo provide.

Another issue is the accuracy of the labels I had to begin with. While hopefully most of them are correct, sometimes the community differs on what these objects are and often their labels change as more observation points come in. As time

passes and the data volume increases one would expect that these labels will be increasingly more reliable.

There might be pre-processing mechanisms that can be applied to the incoming data to ensure that it is sufficiently similar to the simulated data for whatever classifier used to be able to perform well, but as we have seen, this is by no means a simple task: Indeed, if measuring similarities between time series were straight forward, the whole problem of light curve classification would be solved.

When I started this Chapter, I wanted to test whether Attention mechanisms could help with light curve classification, as they work so well in other fields. I particularly thought of early-light curve classification, not only because it seemed like a relevant problem, but also because of the way Attention works. While models like RNNs are recursive and tend to forget what they saw early on, Attention attempts to highlight relevant meaning in all of the sequence. I thought that by adding this to an architecture inherently temporal as an RNN, perhaps it would learn to recognize important features early on and be on the look out for them whenever the model got a short sequence.

While we saw that indeed Self-Attention can help in this way, I was more surprised that the Self-Attentive model managed to generalize between two different datasets when the other models failed. The Self-Attention architecture by no means got good scores, but it clearly caught something that the baselines missed. I think this is worth looking into.

Since this work was completed, there have been relevant developments in using both Deep Learning and specifically Attention mechanisms for light curve classification. In the next Chapter, we will briefly review what these are, examine in more detail what are the intra-relations within a light curve that the Self-Attention mechanism learnt and look further into the issue of bridging the gap between datasets that differ significantly in characteristics.

Chapter 5

Looking further into Attention

As we have seen thus far into the problem, the quality of a training set and how much it resembles the test set is crucial for accurate and robust classification of light curves. This is a non-trivial issue, given that the data sets produced by different surveys differ in characteristics such as passbands and cadence.

Another difficulty and a major bottleneck is that at the beginning of a survey there simply are not enough data samples to properly perform automatic classification, and even when those samples do come in, it takes a while for the scientific community to manually classify them so they can be used as a suitable training set for future samples, which makes the process of supervised learning difficult. One way to surmount this obstacle is to generate simulated light-curves, however, as we encountered in the previous Chapter, sometimes the simulations are not representative of the real data.

Since the work in the previous Chapter was finished, more research has been dedicated to the matter of using Attention mechanisms for light-curve classification. For the contents of this Chapter to make sense, it is first necessary to go over them and other relevant background. In Sections 5.2 and 5.2, we will briefly review said research and establish the required context. We will later go on to inspect the methods used in those works in order to design and build pertinent experiments, which will be divided into three: setting-up a baseline from the methods revised, checking whether the representations learnt through Attention can be enhanced by adding a decoder, and finally, framing the problem in a probabilistic manner. After, we will discuss the lessons that can be learnt from all of this.

5.1 The Transformer

While we saw in the previous Chapter that Self-Attention can be useful for early light-curve classification, so far we have only used a classic form of Self-Attention, that is, additive Attention as an extra layer to a model designed to process data in time-steps (RNNs). In one of the most influential research works in NLP (Vaswani et al., 2017), however, it was argued that *Attention is all you need* to effectively model sequences of words.

Although in the article where the model is proposed it does not have any particular name, later the community coined the term *Transformer*. It was initially designed for language translation, but later proved to be useful for several different tasks, such as: text summarization, text generation, image generation, sequence modeling, time series forecasting, etc.

An Auto-encoder is a sequence-to-sequence model made of an encoder and a decoder. In the case of the Transformer both the *encoder* and the *decoder* use *Multi-Headed Attention* to process sequence time steps in parallel. This differs from other approaches, such as an RNN, in that there is no recursion which makes them more efficient.

The encoder receives an embedding of a sequence and processes it to extract a representation from it through an alignment function f_a, as seen in Section 4.2. In the case of the Transformer, the alignment function used is called *scaled dot-product attention*:

$$f_a = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) \tag{5.1}$$

$$\text{Attention}(Q, K, V) = f_a V \tag{5.2}$$

Where, as in Section 4.2, Q, K and V are called query, key and value vectors. We obtain them by taking the sequence embedding and multiplying it by three matrices that are later learnt through backpropagation; $Q \in \mathbb{R}^{L \times d_q}$; $K \in \mathbb{R}^{L \times d_k}$ and $Q \in \mathbb{R}^{L \times d_v}$, where L is the length of the sequence and $d_q; d_k$ and d_v are the feature dimensions of $Q; K$ and V, respectively. The output of the Self-Attention layer A has dimension $L \times d_v$.

The Transformer also introduced the concept of *Multi-headed Attention* (shown in Figure 5.1) where we now have multiple sets of Q, K, V weight matrices in parallel, that we later concatenate and pass through a linear layer (multiply by another weight matrix W^O) to obtain:

$$\text{MultiHead}(Q, K, V) = A = \text{Concat}[A_0, .., A_h]W^O \tag{5.3}$$

Where A_i =Attention$(Q, K, V) \in \mathbb{R}^{L \times d_{v,i}}$ and $A \in \mathbb{R}^{L \times d}$. $W^O \in \mathbb{R}^{L \times d}$ is also learnt through back propagation.

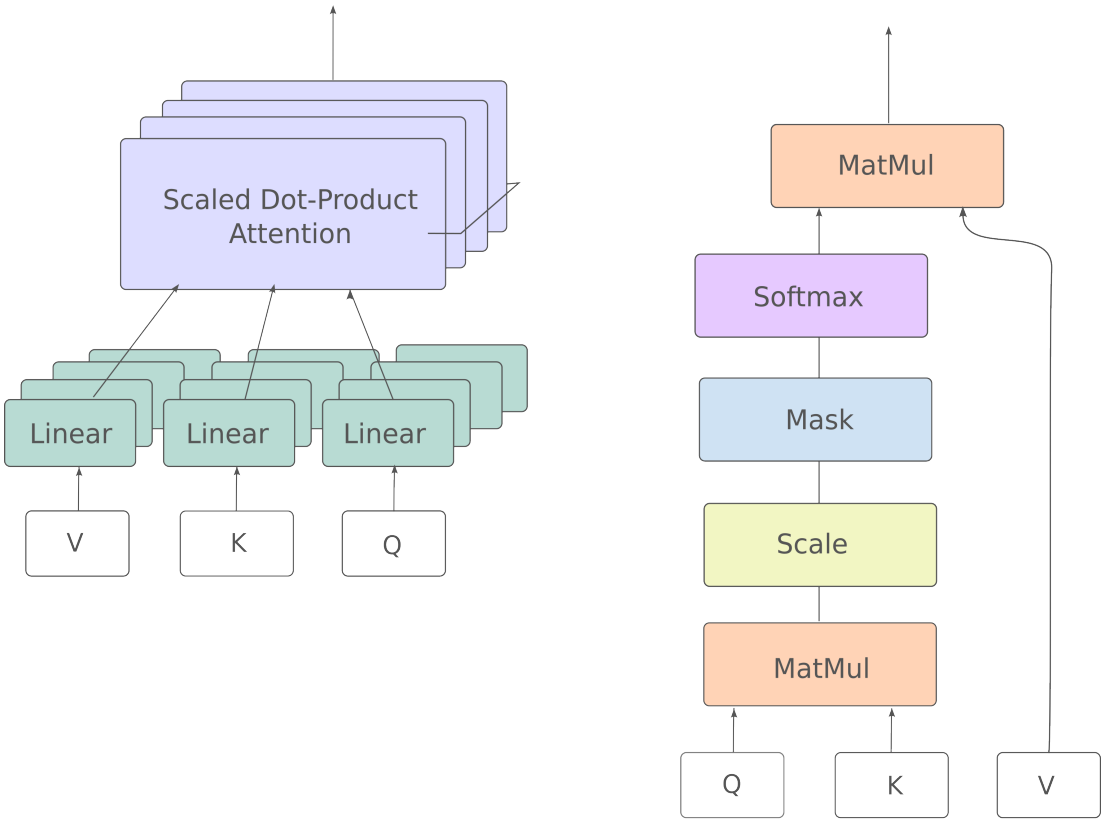


Figure 5.1 Multi-headed Attention found inside the Transformer. The diagram on the left represents the scaled dot-product attention heads, while the diagram on the right shows the functionality inside a scaled dot-product attention block.

The encoder is actually made up of several encoder blocks stacked together, where each of them has an attention layer, a normalization layer and a fully-connected layer. It is important to note that within each encoder block, there are shortcuts between the input of a layer and the following normalization layer (in the same way the ResNet described in Section 4.3).

The decoder, on the other hand, uses both self-attention and encoder-decoder attention. It is made of several decoder blocks stacked together. Each decoder block has a self-attention layer, and encoder-decoder attention layer and a fully-connected layer, with normalization layers in between. As the encoder, it also has shortcuts between each layer and the normalization layer that follows.

The self-attention layer works just as the self-attention layers in the encoder: using Q, K and V values obtained from the inputs of each block. The encoder-decoder attention layers are different, they have that name because they receive K and V matrices obtained from the encoder output. That is, each decoder block receives information coming from the last encoder block.

The Q values, on the other hand, come from the previous outputs of the decoder itself. In the original formulation of the Transformer, given a sequence $x_0, ..., xt$, the decoder generates a target sequence $y_1, ..., ym$, one step at a time. While the process is auto-regressive, it is not recursive. That is, while the output y_{i-1} is required to calculate y_i, all the previous time-steps $y1, ...yi - 1$ are processed in parallel, instead of sequentially. This differs from the RNN approach, were each time we want to generate y_i, all the previous time-steps must be sequentially processed again, in order to ensure that the sequence does not lose its order. Usually, during the training process a technique called *Teacher Forcing* is used, where the whole previous sequence is given to the decoder, instead of having it generate a new sequence from scratch. To ensure that the encoder is not peaking at the future time-steps, we use masking: everything that follows what the transformer is currently predicting is blocked so the decoder does not see it. Figure 5.2 shows the architecture adapted to the case of reconstructing a light-curve.

The encoder and decoder do not receive the sequences directly. Instead they receive sequences that are processed in two ways: embedding and *positional encoding*. Embedding just means projecting a sequence into a higher dimension, where the length of each sequence is also standardized.

Positional encoding means including temporal information in an embedding, since the transformer has no way to account for it otherwise. While in the original article describing the Transformer, positional encoding is obtained simply by using the function described in 5.4, later other approaches were proposed: learning a set of positional embeddings, relative positional encoding, and hybrid approaches.

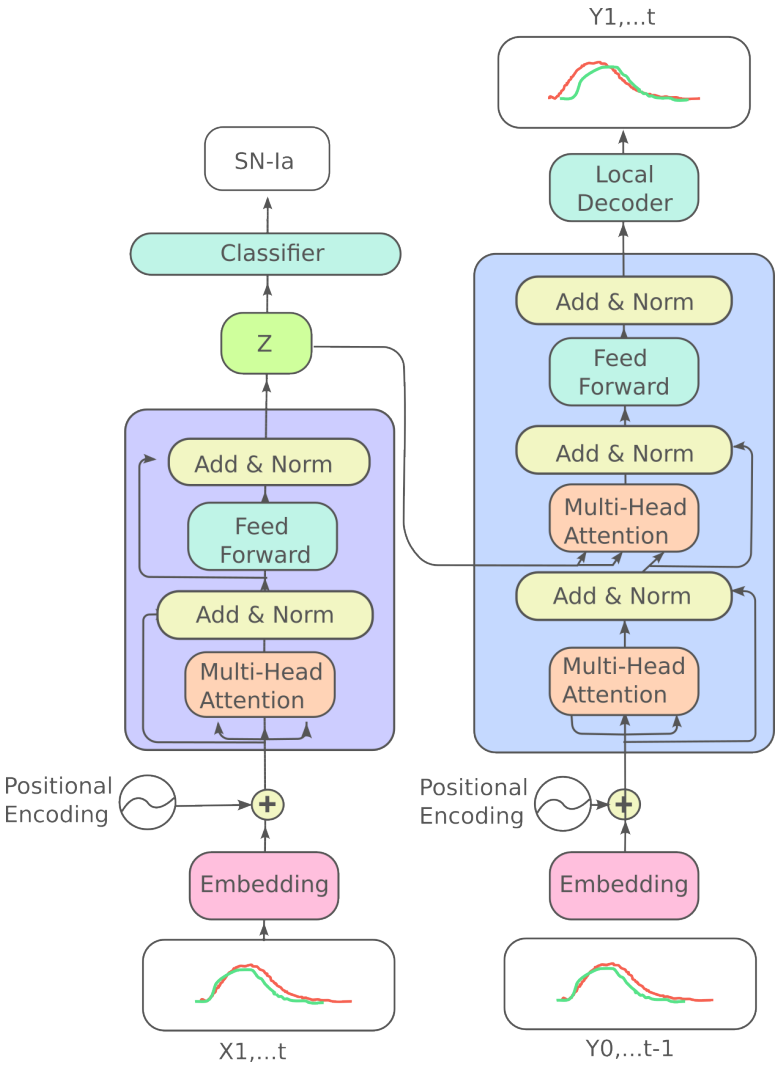


Figure 5.2 The transformer adapted for light-curve reconstruction.

5.1.1 Transformers in Time Series

Since Transformers achieved great success both in Natural Language Processing and Computer Vision (Khan et al., 2022) (CV), it is not surprising that they started to become popular in Time Series analysis. They have been shown to work for several different tasks: TS forecasting, anomaly detection in TS and TS classification.

Although TS resemble the kind of data used for both NLP (in that they are sequences that depend highly on temporal ordering) and CV (in that, in the case of Multivariate-TS, the spatial relations between different channels are akin to the relations of different channels in images), the changes that need to be made to transformers in order to tune them for this kind of data are still an open research

field. These modifications have been explored both at the level of modules (layers) and at the level of the entire architecture (Wen et al., 2022).

The first obvious change that needs to be accounted for is at the embedding/positional encoding stage. Several approaches have been tried: from using the same vanilla positional encoding presented originally (which did not seem to work too well), to going through the path of learning positional encodings (Devlin et al., 2018; Ke et al., 2020). For this latter option the community has investigated using LSTMs and CNNs (Lim et al., 2019), amongst others.

At the higher level of architectural changes, many ideas coming from other architectures have been added to the traditional transformer. Maxpooling operations have been considered between attention blocks in an attempt to downsample sequences (Zhou et al., 2020), convolutions have been used to generate queries and keys (instead of simple matrix multiplication) (Li et al., 2019), and general adversarial networks have been employed and shown to improve results in forecasting (Wu et al., 2020). Other changes have been applied specifically to attend to the periodic nature of certain signals, such as using Fourier decomposition as positional encoding mechanism in with the intent of using attention in the frequency domain (Zhou et al., 2022).

The most marked difference in architecture, however, is perhaps performed in the TS Classification task. Usually, in this case the decoder part of the architecture is discarded altogether (Wen et al., 2022; Liu et al., 2021; Rußwurm & Körner, 2020), and the encoder is used as a way of extracting features for the final classification (fully-connected or feed-forward) layer. For Multi-variate TS, attention has been applied both in the time dimension and in the feature (in the case of light-curves, passband) dimension. Whenever unlabelled data is available, a semi-supervised learning approach can also be tried, where the unlabelled data is first used for weight initialization (Zerveas et al., 2020; Yuan & Lin, 2021).

In the following section, we will see how some of these approaches have been adopted for light-curve classification specifically.

5.2 Transformers for light-curve analysis

Now that we have a deeper understanding of what a Transformer is and how it can be applied to Time Series analysis, let us review how it has been used when

treating light curves.

In the first relevant work we will explore (Allam & McEwen, 2021), an encoder alone was used as a way to extract relevant features for light-curve classification. In this work, only the PLAsTiCC test set was used, so the issue of working with different datasets was not relevant. They do not use any type of decoder, as they assume that an encoder is enough for the task, which seems to be a common assumption when using transformers for TSC (Wen et al., 2022; Liu et al., 2021; Yuan & Lin, 2021; Rußwurm & Körner, 2020; Shankaranarayana & Runje, 2021; Song et al., 2017).

I actually made a similar implicit assumption in the previous Chapter. In that case, my encoder was the RNN and Self-Attention Layer, which allowed for the extraction of a encoded representation of the original sequence. In the work discussed now (Allam & McEwen, 2021), the encoder part of the transformer is used to obtain that representation. They also add a Global Average Pooling (Lin et al., 2013a) operation and a Fully-Connected Layer for the classification. The GAP is there to reduce dimensionality, smooth out extracted features and aggregate the representations obtained in each time-step.

In Chapter 4, I used linear interpolation as a way to represent the light-curves as if they were evenly sampled. In the article discussed, they use GPs instead. While this technique seems to indeed produce good results for light-curve reconstruction, as we have seen it is computationally costly. We have yet to see whether the performance obtained is in fact worth the added cost.

They then project the data into a higher dimensionality by using *convolutional embeddings* (Lin et al., 2013a), which is akin to a time-distributed feed-forward NN. We mentioned before that, unlike an RNN like the one used in Chapter 4, a transformer encoder has no way of discerning the order of the different time-steps unless that information is explicitly added. In Allam & McEwen (2021) they use the positional encoding function from the original Transformer article (Vaswani et al., 2017).

While they achieve great results in the light-curve classification task, two questions arise from this work: would using a decoder really make no difference in the process?; and would this type of architecture be useful if the dataset were not as heterogeneous?

A later article (Pimentel et al., 2022) somewhat addressed this issues. While they

still used the Transformer encoder, they added a decoder that is meant to aid the classification task by hopefully improving the feature extraction capabilities of it. It is important to note that the decoder used in this work is not one based on the transformer, but an LSTM-RNN.

This changes the training process significantly: the idea is that they first train the model as an auto-encoder, that is, they teach it to take a light-curve apart and extract features from it using Self-Attention and then reconstruct it from those features. This is done to ensure that whatever is learnt in the Self-Attention representation, it is indeed relevant, and in fact necessary, to describe the light-curves. While this idea certainly makes a lot of sense, it is worth asking whether the same features required to successfully encode a light curve are helpful when it comes to the task of telling different types of SN apart. They address this issue with a second training step, where they use the output of the encoder as input of the classification layer. Additionally, they change the positional encoding mechanism to be a learnable set of functions based on a Fourier decomposition.

In a third work investigating the usefulness of transformers for light-curve analysis (Morvan et al., 2022), they are used for de-noising. Here they also use encoder-only architecture. The positional encoding tehcnique they use is the same one proposed in the original transformer (Vaswani et al., 2017), that is, fixed rather than learnt. Although in some TS related work they claim that this does not necessarily yield the best results (Lim et al., 2019; Zerveas et al., 2020), in this particular work they find that fixed positional encoding gives better results that the learnable options. While they only use the encoder part of the transformer, the sequence does need to be reconstructed in some way, and they do this with a linear (fully-connected) layer. In essence, this layer serves as the decoder, even if it is not directly addressed with that name.

While all these approaches give excellent results, it is difficult to compare them, since they use different data-sets and different classification taxonomy. Indeed this difficulty establishing a baseline seems to be recurring in the field of light-curve classification and leads one to the conclusion that any of these methods might achieve somewhat similar performance. In the following Section 5.3.1 we will start by establishing a baseline (if not fully due to time-constrains, at least partially) with the aim not only of comparing approaches, but of having a better idea of what could be useful to build upon.

Something else that stands out while inspecting the work already done on

transformers for light-curve analysis is the fact that the transformer decoder is never used. While I do not know why this is the case, I think it is worth exploring whether the whole transformer architecture, decoder included, might prove useful. We will look into it in Section 5.3.2.

5.3 Experiments

This Section is divided into three: First we will inspect the methods already proposed and establish a baseline for the following experiments, second we will see whether the transformer decoder is useful for light-curve classification and finally, we will extend the Transformer to work as a Variational Auto-Encoder, with the aim of giving the architecture a simple way of generalizing between datasets.

In all the following experiments we will be mainly using the simulated dataset built in Section 4.7.1.2, but will be reducing the taxonomy to three types of objects only, for simplicity: SNe-Ia, SNe-Ib/c, SNe-II and SLSNe. We will use this dataset for three reasons: Firstly, in case more data is needed at some stage, it will be easy to produce it. Secondly, since the simulated data is meant to emulated the ZTF dataset, the architectures will be able to process real ZTF light-curves that share the same number of features (passbands). The third and last reason is the underlying issue that has been present in all this thesis (and in most work dedicated to light-curve analysis using ML): the marked differences between simulated and real data. By using a dataset that differs from the real light-curves, we have the chance to periodically check how the performance of the experiments changes with the gap between the datasets.

5.3.1 Initial Experiments: Establishing a baseline

For simplicity, we will initially consider classification using an architecture without a decoder, since this seems to be a common case, and focus on the encoder, its inputs and outputs. We will explore different data representations, embeddings, positional encodings and operations to perform on the output of the encoder. An example of one of the architectures used is displayes in Figure 5.3.

While depending on the experimental set-up it may vary a little, its overall structure will remain the same.

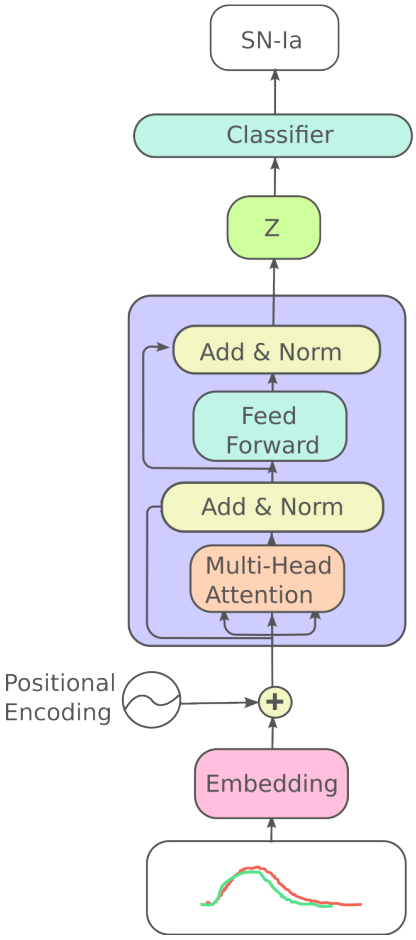


Figure 5.3 A classification model using the encoder part of a transformer to extract feature representations.

5.3.1.1 Data Representations

It is curious that although the literature on light-curve classification is extensive and many works use several different ways to represent said light-curves, there appears to be no consensus on what kind of representation is most beneficial for a given task. In a way it makes sense, since light-curves from different surveys present different characteristics and thus might need different structures.

It is often the case, however, that some sort of interpolation is used to account for the gaps in the light-curves where there are no observation points. In Chapter 4 I used linear interpolation, with added channels to describe distances to the nearest real points (as described in Section 4.5). GPs are another common way of doing

this: the winner of the PLAsTiCC challenged used them (Boone, 2019), and so does one of the articles reviewed in Section 5.2 (Allam & McEwen, 2021).

Another one of the works described in Section in 5.2 (Pimentel et al., 2022), however, uses no interpolation, since they present a positional encoding technique that is tailored to take care of the issue.

In the experiments in this Section, we will be using three types of data representation: light-curves that are linearly interpolated (as described in 4.5), light-curves that are reconstructed and then re-sampled using GPs (as presented in Allam & McEwen (2021)), and light-curves that are not interpolated (as described in Pimentel et al. (2022)).

5.3.1.2 Embeddings

We will use two types of sequence embeddings: convolutional embeddings (used in Allam & McEwen (2021)) and linear projections (used in Pimentel et al. (2022) and Morvan et al. (2022)). Both seem to be fairly common when using Transformers for TS.

A *linear projection* in Machine Learning terms is nothing more than a weighted sum of features (that is, a fully connected layer) that change the dimensionality of an input.

A *convolutional embedding*, on the other hand, is obtained after running convolutions through an input, with a sliding window of size one. In essence, this is the same as a series of time-distributed linear projections, that is, one linear layer per time-step.

The aim of both these methods is to change the dimensions of an input. In our case, each light-curve has two dimensions: the light-curve length and the necessary channels (in our case, at least the passbands g and r). Embedding a light-curve will expand the last (channel) dimension so it matches the size of the encoder model and thus can be processed.

5.3.1.3 Positional Encoding

We will also check two types of positional encoding techniques: the default positional encoding functions presented in Vaswani et al. (2017) and the positional

encoding described in Pimentel et al. (2022), which I will call 'Fourier based'.

The default positional encoding uses the following functions to add temporal information to the sequences:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}) \tag{5.4}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d}), \tag{5.5}$$

where pos is the position of a given time-step, i is a given dimension and d is the number of dimensions of the embeddings. Once these values have been calculated for all positions and dimensions, they are simply added to the embedded sequences.

On the other hand, the positional encoding used in Pimentel et al. (2022) is described as:

$$\dot{x} = \gamma(t) \odot x(t + \epsilon) + \beta(t) \tag{5.6}$$

$$\gamma_d(t) = \sum_{m=1}^{M} a_{d,m}\sin(2\pi mt/T) + b_{d,m}\cos(2\pi mt/T) \tag{5.7}$$

$$\beta_d(t) = \sum_{m=1}^{M} v_{d,m}\sin(2\pi mt/T) + w_{d,m}\cos(2\pi mt/T), \tag{5.8}$$

where x is an embedded sequence, t is a time of observation in the sequence, $M = 12$ (the functions are based on a Fourier decomposition with M harmonic components), d is the dimensionality of the embedded sequence and $T = 1.5\max(t_{-1})$, where t_{-1} refers to the last observation time of a sequence.

5.3.1.4 Encoded representation

After the embedded sequence with positional information goes through the encoder, we obtain an encoded representation, which in our case will be used for classification. This representation Z will have the same dimensions as the encoder input: $l \times d$, where l is the length of the sequence and d is the number of dimensions. Each element $z_i \in Z$ encodes a representation for a different time-step i.

There are many different ways to deal with Z. One of them (used in Pimentel et al. (2022)) is using only the last representation z_l, which is obtained after the encoder went through all of the sequence. Another way is using some sort of operation to aggregate all z_i, for example, GAP, used in Allam & McEwen (2021). Although there exist several other ways (Shankaranarayana & Runje, 2021; Wen et al., 2022), we will focus on these two, which are commonly used.

5.3.1.5 Initial Results

I run several combinations of the options presented above. Hyper-parameters are presented in Table 5.1.

Hyper Parameter	Value
Attention Blocks	1
Attention heads	4
Size of embedding dimension	128
Size of linear layers	100
Learning rate	10^{-3}
Weight decay coefficient	10^{-2}
Batch size	64
Epochs	100
Validation Step	3
Patience	3

Table 5.1 Default values for hyper-parameters for the experiments.

Table 5.2 presents classification scores for a simulated test set and for a test set made of real light curves. The best confusion matrices for the simulated and the real test sets are presented in Figure 5.4. Let us look at these first.

First, we see that there is a marked improvement in results compared to the last Chapter. While this may be because of attention (given the results in the last Chapter I am inclined to think so), reducing the taxonomy from 6 to 4 classes probably played a role.

We also see that for the simulated test sets, GPs and linear interpolation have

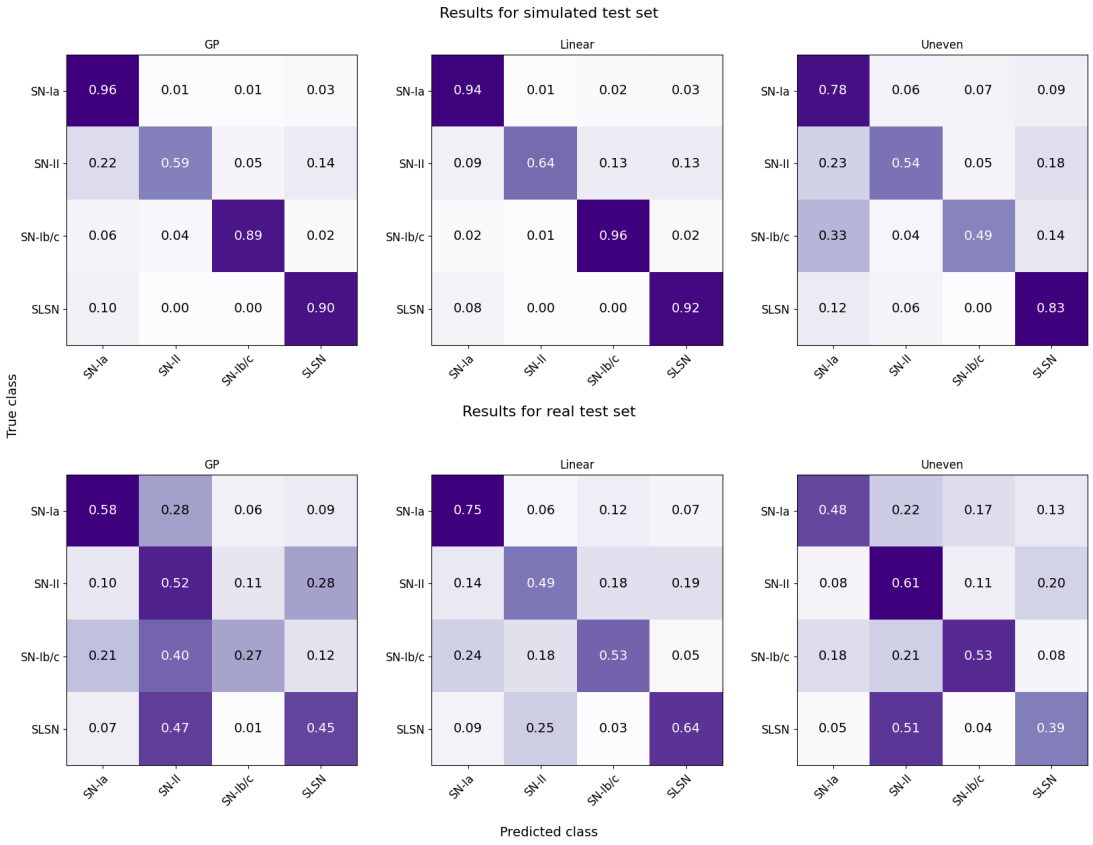


Figure 5.4 Confusion matrices for both the simulated and the test set using different data representations.

similar confusion matrices, where the most missed class is SNe-II. The confusion matrix for the uneven data representation, however, misses SNe-Ia and SLSNe more often and has trouble classifying SNe-Ibc. Looking at the confusion matrices for the real test set however, shows more marked differences. The GPs seem to be the worst performing, confusing many objects with SNe-II. The linear interpolations have the best results, classifying a significant portion of SNe-Ia and SLSNe.

It is interesting that in Chapter 4.7, while using simple additive attention combined with an RNN, the confusion matrix also showed that the model was able to catch SNe-Ia and SLSNe more often than the rest of the classes. At the time I though this was because Attention was picking up on some relation that could be generalized, but now we see that even when all models are using attention, the same occurs. In all cases, the class that more often is given a wrong label is SN-II, which are thought to be SLSNe. When looking at Table 5.2, the main thing that stands out from these experiments is how dramatically data representation affects results. If we look at the left side of Table 5.2 (where the test set is simulated),

we see that data representations that are evenly sampled (interpolated in some way) allow for much better classification scores than the data representation with no interpolation. On the other hand, if we look at the right side of Table 5.2, we see that this changes when dealing with real data. Now, in most cases, the light-curves reconstructed using GPs have scores that are much worse than the other two data representations used.

I think that the reason for this is that both the light-curves that are unevenly sampled and the light-curves that are linearly interpolated make the gaps in the light-curves explicit. In the case of the light-curves with no interpolation, the classifiers do not pay attention to the missing points simply because they are not there. In the case of the linearly interpolated light-curves, the classifiers can learn to pay attention to the points that are closest to actual observation points, since (as described in Section 4.5) they have extra channels with that information.

It is also noteworthy that the difference in scores between real and simulated test set is a lot less marked when we consider the unevenly-sampled light-curves. This tells us that even though having light-curves that are evenly sampled helps significantly in the learning process, it makes these light-curve classification models less transferable. In Table 5.2, the best scores for each experimental set-up are highlighted in blue. If we look at the losses for the real test set, we see that most of the time, the lower losses are achieved by models trained with unevenly sampled data. That is, when the classifiers were wrong at predicting a class, they were wrong with less certainty.

Another thing that is easy to spot is that most of the highest F1-scores are achieved when using the light-curves that are linearly interpolated. I think this is because they are both evenly sampled and maintain information about their gaps. Perhaps this would not be the case if the cadence were more spaced.

If we try to map layer configurations to classification scores, patterns are less clearly defined. It seems, however, that when considering real light-curves, results tend to be better when using linear projection as embedding. I think this is because this method simple has fewer weights than a convolutional embedding, and so there is less of a chance to overfit.

It is not clear to me how much better or worse a Fourier-based positional encoding method is overall, but it appears to be useful at least for linearly-inteprolated light-curves. Interestingly enough, it does not seem to work better or worse with unevenly sampled light curves.

		Sim. test set			Real test set		
Exp	DR	F1	Acc	Loss	F1	Acc	Loss
LP, D, LE	GP	0.588	0.771	0.527	0.354	0.443	1.728
	L	0.737	0.899	0.251	0.507	0.637	1.277
	U	0.432	0.733	0.748	0.441	0.613	1.131
LP, D, GAP	GP	0.681	0.872	0.360	0.249	0.211	4.139
	L	0.713	0.883	0.326	0.485	0.619	1.938
	U	0.565	0.778	0.661	0.481	0.637	1.385
LP, F, LE	GP	0.725	0.897	0.288	0.389	0.539	1.632
	L	0.724	0.901	0.282	0.516	0.685	1.494
	U	0.509	0.724	0.756	0.392	0.502	1.455
LP, FB, GAP	GP	0.727	0.882	0.313	0.221	0.229	3.554
	L	0.759	0.917	0.237	0.449	0.660	1.794
	U	0.477	0.654	0.894	0.355	0.436	2.218
C, D, LE	GP	0.702	0.873	0.268	0.268	0.310	2.350
	L	0.665	0.806	0.386	0.386	0.441	2.213
	U	0.489	0.734	0.431	0.431	0.600	1.113
C, D, GAP	GP	0.675	0.841	0.453	0.235	0.209	5.366
	L	0.713	0.871	0.351	0.428	0.516	1.990
	U	0.536	0.732	0.729	0.417	0.534	1.483
C, F, LE	GP	0.733	0.900	0.258	0.313	0.365	2.303
	L	0.691	0.857	0.367	0.443	0.558	1.785
	U	0.502	0.727	0.715	0.397	0.516	1.435
C, F, GAP	GP	0.766	0.903	0.252	0.230	0.237	2.764
	L	0.733	0.895	0.302	0.481	0.641	1.326
	U	0.571	0.801	0.586	0.437	0.611	1.476

Table 5.2 Results for both a simulated test set and a real test set.
The first column encodes the experimental set-up in letters:
LP: Linear projection embedding
C: Convolutional embedding
D: Default positional-encoding
F: Fourier-based positional encoding
LE: Last element of z is used for classification
GAP: Global average pooling is performed over z
The second column (DR) shows the different data representations:
GP: light-curves reconstructed using GPs
L: linearly interpolated light-curves
U: unevenly sampled light-curves
For each experiment block, the best scores are highlighted blue. The overall best F1-score is highlighted in orange.

As for the output z of the encoder, when considering one simulated dataset, GAP seems to be the best way to process it. When considering two different datasets though, just taking the last representation z_l appears to be the better option.

Moving forward, we will be using linearly-interpolated light curves, linear projections as embedding, Fourier-based positional encoding, and considering z_l.

5.3.2 Autoencoder using Attention: Adding a Transformer Decoder

While it is often assumed that the use of the encoder part of the transformer is enough to obtain good results, the question remains whether that is indeed so for light-curve classification. It is reasonable to expect that the features needed for encoding and reconstructing a light-curve would be relevant and perhaps needed for light-curve classification.

In order to test this, I decided to try something similar to what was done in Pimentel et al. (2022). That is, building an auto-encoder by adding a decoder architecture to our already implemented classifier. There are some significant differences, however, in the learning process.

Since we now have two tasks, reconstruction and later classification, the learning workflow will also be divided into two: first the weights of the network are initialized via the process of light-curve reconstruction and a fine-tunning process follows after, where the network learns to classify.

In Pimentel et al. (2022), the initial reconstruction process is done using simulated light-curves only. In the fine-tunning stage, however, only real light curves are used. Their idea is to alleviate the issue of the differences between datasets this way. In our case, however, I do not have enough light-curves that are labelled with sufficient certainty. Because of this, we will initially use simulated light-curves for the whole of the learning process (and will try something else in Section 5.3.3).

The learning process considers two different loss functions: a reconstruction loss and a classification loss. For the reconstruction task, we will be using weighted mean squared error (WMSE) and for the classification task, we will use cross-entropy, as in the rest of the thesis. While at the classification stage only cross-entropy will be considered, during the fine-tuning stage the loss function that will be minimized is actually made up of both functions:

$$\mathcal{L}_{\text{reconstruction}} = \frac{1}{N}\sum_{i=1}^{N} k_0 \mathcal{L}_{\text{rec}i} + k_1 \mathcal{L}_{\text{classification}i} \tag{5.9}$$

$$\mathcal{L}_{\text{fine-tuning}} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\text{classification}i} \tag{5.10}$$

Where $k_0 = 10^4$ and $k_1 = 1$ are initial hyper-parameters, we will change them later.

Although in Pimentel et al. (2022) the fine-tunnig process only modifies the weights of the classifier while the weights of the auto-encoder are frozen, this failed completely when I tried it. The network was only able to learn to classify at all when it could modify all of its weights during training, which is why in the following experiments the entire network is considered during the fine-tunning. We will come back to this in Section 5.3.3.

I decided to try three different types of decoders, the first one is a GRU of one layer; similar to the one used in Pimentel et al. (2022); the second one an MLP similar to what was used in Morvan et al. (2022) and the third one is a transformer decoder. Additionally, since it was not all that clear from the previous section whether the Fourier-based positional encoder provided significant benefits, I decided to try both that approach and the default positional encoding. I maintained the hyper-parameters from the previous experiments. Results are presented in Table 5.3, and the best confusion matrices for each decoder are presented in Figure 5.5

In the best case confusion matrices, all decoders work similarly well when tried over the simulated test set and results do not vary greatly when compared to those of the linerly inteprolated data representation in Figure 5.4. Over the real test set, however, only the transformer decoder shows equally better results than the version without a decoder. For the case of the GRU decoder and linear decoder, results actually get worse.

Let us now look at Table 5.3 and the case of the transformer decoder. This time the Fourier-based positional encoding does make a difference, where classification results for the real test set are markedly better. This positional encoding also improves reconstruction results for the simulated test set, but the opposite happens for the real test set. This might be because with the Fourier-based

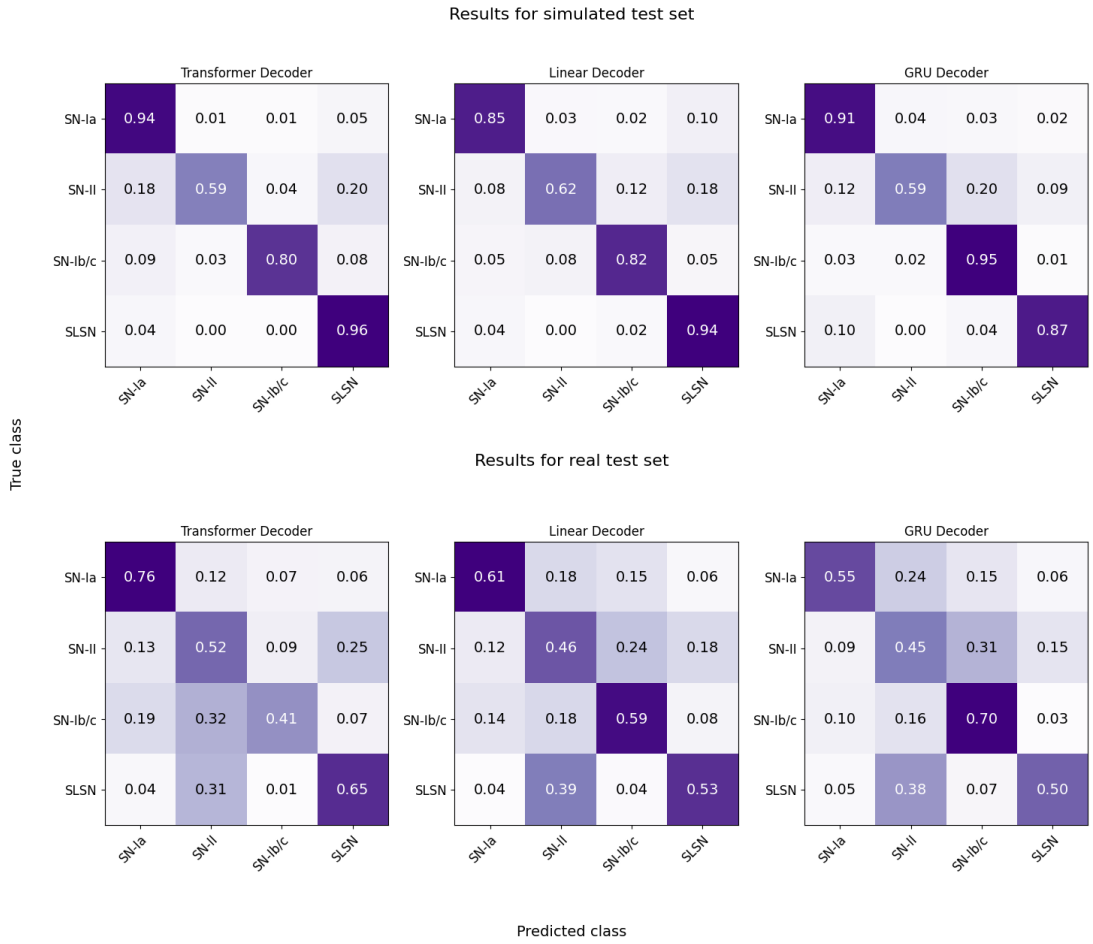


Figure 5.5 Confusion matrices for both the simulated and the test set using different decoders in the models trained.

positional encoding more weights are added to the architecture, making it easier for it to be customized for a particular dataset.

For the transformer decoder, the reconstruction loss is better for the simulated data when it is worse for the real data and vice versa. It makes sense that a better reconstruction for a certain dataset would mean a worse reconstruction for another one, as the auto-encoder would try to make the new dataset resemble the one it learnt from. A higher reconstruction loss for a given dataset would mean that the auto-encoder latent space (the representation vector that is the output of the encoder) would encode features specific to that dataset more accurately.

For the real data, classification results worsen as reconstruction loss improves. Having a higher loss (worse performance) for the reconstruction of the simulated dataset also impacts classification results negatively for the real test set.

This seems to suggest that when a Trasformer auto-encoder learns to reconstruct

		Sim. test set				Real test set			
Dec	PE	R.Loss	F1	Acc	Loss	R.Loss	F1	Acc	Loss
G	D	1.661	0.690	0.864	0.394	2.666	0.4319	0.540	1.470
G	F	0.076	0.566	0.800	0.501	1.861	0.379	0.444	2.081
L	D	0.034	0.633	0.818	0.405	0.802	0.446	0.580	1.487
L	F	0.037	0.681	0.861	0.357	0.802	0.443	0.556	1.510
T	D	0.370	0.646	0.841	0.430	0.673	0.342	0.357	2.162
T	F	0.051	0.696	0.875	0.321	1.373	0.486	0.688	1.368

Table 5.3 Results for the simulated and the real test set for the case where a decoder is used for reconstruction previous to fine-tunnig the network for classification. The first column shows the different decoders used:
G: GRU decoder
L: Linear decoder
T: Transformer decoder
The best results for each metric are highlighted in blue. R.Loss stands for reconstruction loss.

one SNe light-curve dataset, it still extracts features that can be helpful for the classification of another SNe light-curve dataset, even if not for its reconstruction. It might be that this is only possible because the dataset used in the initial reconstruction phase (the simulated one) contains light curves with characteristics that more obviously mark them as being from a certain class, since they were created from templates.

For the case of the GRU decoder, classification results are similar to those obtained when using a transformer decoder, only a bit better for the validation sets and a bit worse for both simulated and real test sets. In this case, default positional encoding seems to work a lot better than the Fourier-based one.

For the linear decoder, classification results are very similar to the other two options, but it is interesting that reconstruction losses were low at the same time. In this scenario the same features useful for reconstruction seem to be useful for classification.

Overall, however, when comparing all of these results with the ones in Table 5.2, there seems to be no clear advantage to using a decoder for solving a classification task, as it provides little to no improvement in classification scores while at the same time adding to computational costs. A decoder, however, might prove useful

in a different setting. We will explore this in the next Section.

5.3.3 Variational Auto-Encoder using Attention: a probabilistic approach to the Transformer

We saw how an auto-encoder is used to encode a sequence into a latent representation and then decode it into a similar sequence, with the aim of capturing in that representation relevant characteristics and relations present in the sequence. From this, one could think that if we slightly altered the encoded representation, z, new data similar to the input sequence could be generated. This, however, does not necessarily hold true, since we have no way of knowing whether the encoder produced a latent space that is regular enough for us to do so. If we think about the training process in our auto-encoder, it was taught only to minimize the reconstruction loss as much as possible, no matter how the latent space was organised.

This means that the initial states of the weights in the network, the order of the data that was used to train and the definition of the architecture itself all provide opportunities for the network to overfit. Regularization techniques somewhat mitigate this, but still not enough for us to be sure that whatever was encoded in the latent space represents features inherent to SNe in general instead of the dataset used during training specifically.

Variational auto-encoders (VAEs) (Kingma & Welling, 2013) attempt to solve this issue by encoding inputs as distributions instead of fixed points. Instead of the deterministic model of encoder and decoder we had previously, we will now have a probabilistic model of the same structures.

Let us say there is a probability model for samples x and its inherent latent variables z. The joint probability of the model is $p(x, z) = p(x|z)p(z)$. We can see x but want to infer the characteristics encoded in z. In other words, we want to compute $p(z|x)$. We now that:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{5.11}$$

However, $p(x)$ is intractable and would take forever to calculate, so we need to

approximate our posterior $p(z|x)$. Let us consider a different distributions q for our approximation.

In the context of probabilistic modeling, we can think of $p_\theta(x)$ and q_ϕ, where θ and ϕ are the values that parametrize p and q, respectively. In the context of VAEs, the probabilistic decoder will compute $p_\theta(x|z)$, and the probabilistic encoder will output $q_\phi(z|x)$.

In order to ensure that $q_\phi(z|x) \approx p_\theta(z|x)$ one commonly used metric is the Kullback-Leiber divergence, which is used to measure the distance between different probability distributions. It is defined as:

$$D_{\mathbb{KL}}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{z\sim q_\phi(\cdot|x)}[\ln\frac{q_\phi(z|x)}{p_\theta(z|q)}] \tag{5.12}$$

$$= \mathbb{E}_{z\sim q_\phi(\cdot|x)}[\ln\frac{q_\phi(z|x)p_\theta(x)}{p_\theta(x,z)}] \tag{5.13}$$

$$= \ln p_\theta(x) + \mathbb{E}_{z\sim q_\phi(\cdot|x)}[\ln\frac{q_\phi(z|x)}{p_\theta(x,z)}]. \tag{5.14}$$

We still cannot compute $p_\theta(x)$, but we can define:

$$L_{\theta,\phi}(x) := \mathbb{E}_{z\sim q_\phi(\cdot|x)}[\ln\frac{p_\theta(x,z)}{q_\phi(z|x)}] \tag{5.15}$$

$$= \ln p_\theta(x) - D_{\mathbb{KL}}(q_\phi(z|x)||p_\theta(z|x)). \tag{5.16}$$

Expanding the joint distribution into the prior and likelihood terms and using the product rule for logarithms we get for a data sample x_i:

$$-L_i(\theta,\phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(x_i|z)] - D_{\mathbb{KL}}(q_\phi(z|x_i)||p(z)). \tag{5.17}$$

Which is usually called ELBO or Evidence Lower Bound function. Maximizing the ELBO with respect to θ and ϕ is equivalent to maximizing $\ln p_\theta(x)$ and minimizing $D_{\mathbb{KL}}(q_\phi(z|x)||p_\theta(z|x))$ at the same time. And so this becomes an optimization problem, where the ELBO is our loss function. While maximizing the log-likelihood of the observed data is equivalent to minimizing a reconstruction loss, minimizing $D_{\mathbb{KL}}$ ensures that the z representations that the encoder and

decoder learn to input and output, are sampled from a regular latent space.

In the most simple case, the encoder $q_\phi(z|x)$ is modeled as a Gaussian probability density. Now the encoder will no longer output a representation z, it will instead learn to give a mean μ and a variance σ^2. We can then sample a noisy z, $\hat{z} = \mu + \sigma \times \epsilon$ (where $\epsilon \in \mathcal{N}(0, 1)$), which will be passed to the decoder $p_\theta(x|z)$. In this case p is defined as $p(z) = \mathcal{N}(0, 1)$, which means that if the encoder gives output representations $\hat{z}$ that are too different from a standard normal, $D_{\mathbb{KL}}$ will apply a penalty in the loss. Although this is the most common way of doing modeling a VAE, there exists extensive literature exploring different alternatives, even in the context of Transformers themselves (Jiang et al., 2020; Tang & Matteson, 2021; Movellan & Gabbur, 2020; Wang et al., 2013; Henderson & Fehr, 2022). For experiments in Section 5.3.3.1, I built a Transformer with a simple mechanism for it to be able to have an organized latent space, which can be seen in Figure 5.2.

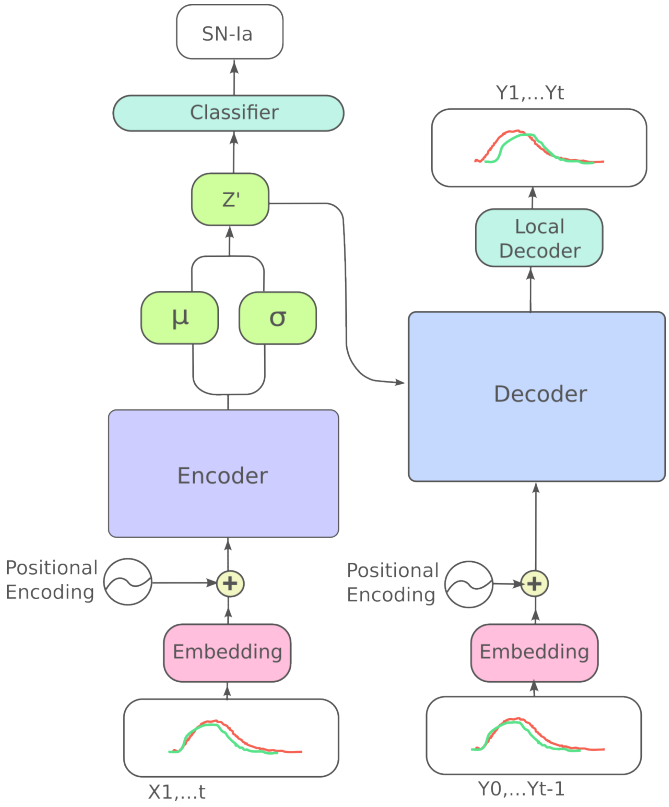


Figure 5.6 Transformer-VAE architecture. The encoder now outputs parameter for a Gaussian distribution, instead of a fixed vector representation z. The encoder and decoder are not drawn in detail, but they are the same as in Figure 5.2

It is also important to point out that the aim of VAEs is not reconstruction.

Rather, the aim is to learn representations that allows us to gain insight about our data and even generate new samples from it. This changes the task significantly, since minimizing the reconstruction loss function will no longer be top priority.

I mentioned before how, although the classification results of the real and simulated test set when using a decoders were generally fine, I was not able to reproduce the training phase as it was presented in Pimentel et al. (2022). Obviously part of it is that I do not have sufficient real light curves for the fine-tuning process. Still, freezing the layers of the network that did the reconstruction resulted in the network not being able to learn at all when it came to classification, even for the case of using only the simulated data from the same dataset.

While the experiments in Section 5.3.2 did consider the classification loss at the time of reconstruction training, as described in Equation 5.9, it did not add significantly to the overall loss because of the value of the parameter $k1$. I changed this during the reconstruction phase in the following experiments for two reasons: first, I wanted to see whether that would help the network learn while freezing the encoding-decoding layers, in order to maintain the reconstruction capabilities of it; second, since the loss now will include KL-Divergence ($D_{\mathbb{KL}}$), I did not want the classification loss to get neutralized by it. The new loss function for the reconstruction training phase is defined as follows:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(1-k_1)\mathcal{L}_{R_i} + k_1\mathcal{L}_{C_i} - \beta D_{\mathbb{KL}}. \tag{5.18}$$

Where N is the number of samples in a batch $\mathcal{L}_R$ is the reconstruction loss (in this case Mean Squared Error), $\mathcal{L}_C$ is the classification loss (cross-entropy) and, since $q_\phi(z|x)$ needs to be approximate a Gaussian and $p(z) = \mathcal{N}(0,1)$:

$$D_{\mathbb{KL}} = -\frac{1}{2}(\log(\sigma_i^2) - \sigma_i^2 - \mu_i^2 + 1). \tag{5.19}$$

To be able to regulate the contribution to the total loss of the regularization loss (in this case, $D_{\mathbb{KL}}$), I implemented a variant of $D_{\mathbb{KL}}$ known as $\beta D_{\mathbb{KL}}$. This variant introduces a hyper-parameter β that can be tweaked to adjust the balance between the components in the loss function, thus prioritizing either the accuracy

in reconstruction (if $\beta < 1$) or the disentanglement of the learnt latent space (if $\beta > 1$).

I also added a sigmoid cyclic schedule to the hyper-parameter β. This means that, during the reconstruction training phase, the parameter β be will periodically updated. I did this because, at the beginning of the training process, since the sampling of $\hat{z}$ tends to give out random numbers, the decoder sometimes learns to discard that representation altogether and instead focuses solely on the last time-step that it decoded. For all of these reasons, it does not make sense to compare reconstruction losses with the previous experiments. Classification metrics, however, remain the same.

5.3.3.1 Classification using VAEs

The training process of the VAE used in the following experiments is at first similar to the one used previously for the AEs: we first have a reconstruction phase where the loss function in Equation 5.18 is minimized, followed by a fine-tuning phase where we optimize cross-entropy. However, as we will discuss in Section 5.3.3.2, this process will eventually be changed. The learning rate for both training phases was initially set to be $\eta = 10^{-3}$, and since the learning algorithm used is Adam (Kingma & Welling, 2013), it is adjusted during the process. I did not manually fine-tune the learning rates due to computational time constrains, but it would definitely be interesting to explore different learning rate values and strategies to modify it during training.

Given that in the experiments of Section 5.3.2, during the fine-tuning phase, the loss functions failed to converge whenever I froze the encoder-decoder layers, I decided to start these experiments by trying out whether this was the case for a Trasformer-VAE architecture also. Since the architectures in Section 5.3.2 learnt encoded representations z as fixed points instead of probability distributions, it did not seem strange that by not allowing the networks to update their weights entirely, they would fail to generalize, specially from one dataset to another. I considered three scenarios: first I froze the encoder and decoder parts of the architectures during fine-tuning; then I froze only the decoder part, considering also the layers in charge of sampling $\hat{z}$; and finally, I allowed the networks to update all their weights.

Also, I decided to increase the importance of the classification loss during the

initial training process, thinking this would help the networks learn features during the reconstruction phase that would also be useful later for classification. While I tried several different k_1 values, it was eventually clear that the importance of the classification loss had to be in the same order of magnitude as the reconstruction loss for it to make a difference later during the fine-tuning phase.

As hyper-parameters, I decided to decrease the model size (the hidden dimension of the feed-forward layers within the attention heads) from 100 to 64 in order to mitigate over-fitting. I was also less patient with the networks, forcing them to stop early after three validation checks where they failed to learn (as before, I run a validation check every 3 learning epochs). The epoch budget for both reconstruction and classification learning was 50.

We can see in Table 5.4 that classification results improved somewhat for both the simulated and real tests sets compared to the previous Section, though not by much. Some results are now comparable to the ones obtained in Section 5.3.1, when we did not have a decoder. The first column indicates the layers that were frozen during fine-tuning (E for encoder and D for Decoder). In the case where both the Encoder and Decoder were frozen, a lower k_1 negatively impacts classification scores for both tests sets. This suggests that by freezing both those layers the network struggles to learn features that help in classification and so it is forced to rely on the features it learnt during reconstruction. Since lower k_1 value means that during reconstruction training little importance was given to the labels of the samples, the network fails to tell the light-curves apart.

In the case where no layer was frozen during the fine-tuning stage, classification scores for the real test set also drop with a lower k_1, though less significantly so. This does not happen for the classification scores of the simulated test set. I think this is because, by not freezing any layers, the network simply starts behaving as a regular auto-encoder and its latent space adapts to the simulated data. This means that, for the classification of the real test set, k_1 holds greater importance, since the network relies on whatever general features it learnt during reconstruction that might help with classification.

The case were only the Decoder is frozen during fine-tuning appears to be the most robust to fluctuations in k_1. I think this is because the network is able to keep its latent space organized while at the same time having the ability to learn to encode features specific for classification. Since a classification loss term was

considered in the loss used during the reconstruction phase, in a way the classifier part of the architecture also acts as the decoder, in the sense that it too learnt to receive encoded representations sampled from Gaussian distributions. It seems that as long as the latent space keeps its order, the classifier retains its ability to interpret those representations to an extent, even if the encoder has changed the meaning of those representations.

		Sim. test set				Real test set			
Freeze	k_1	R.Loss	F1	Acc	Loss	R.Loss	F1	Acc	Loss
ED	0.1	0.230	0.397	0.697	0.880	1.327	0.287	0.376	1.487
ED	0.3	0.395	0.655	0.857	0.394	1.384	0.386	0.453	1.995
ED	0.5	0.386	0.654	0.828	0.481	1.256	0.518	0.709	1.153
D	0.1	0.308	0.646	0.854	0.376	1.262	0.507	0.662	1.182
D	0.3	0.384	0.653	0.845	0.423	1.259	0.457	0.573	1.580
D	0.5	0.346	0.718	0.892	0.307	1.137	0.494	0.642	1.249
-	0.1	0.334	0.602	0.7945	0.535	1.275	0.338	0.376	2.179
-	0.3	0.347	0.725	0.891	0.310	1.294	0.448	0.453	1.508
-	0.5	0.427	0.681	0.876	0.320	1.277	0.506	0.709	1.162

Table 5.4 Test results for the real and simulated test sets when classified by a Transformer-VAE architecture trained in different ways. The first column refers to the layers that were frozen during the fine-tuning phase, where D stands for 'Decoder', ED for 'Encoder-Decoder' and - means no layers were frozen. The second column (in orange) shows how the loss function was altered by changing its hyper-parameter $k1$, a higher value is presented in darker shades. F1-scores are shown in blue, where the best scores for each block are also presented in darker shades.

There are two questions that stem from these results: First, how would classification scores be altered if only the latent space (in the practical sense, only the layers that output μ, σ^2 and do the sampling) was frozen during fine-tuning? And perhaps more importantly, is fine-tuning even necessary in our case?

5.3.3.2 Classification using VAEs and semi-supervised learning

The idea of initializing the weights of a network by using smooth simulated data for reconstruction makes sense, as one would expect that at least some (if not most) of the information needed for reconstruction would also be necessary for labelling. As we have seen this is not necessarily the case for SNe light-curves. The aim of fine-tuning is to make sure that the network does indeed learn features that are useful specifically for the task at hand, in this case, classification. The issue, however, is that this also prompts the network to learn features specific to a dataset. This would not be an issue if the fine-tuning were done with light-curves coming from the same dataset as the test set we are trying to classify, but the dataset of real light-curves that are labeled with high certainty that I have at hand is much too small to allow for it.

Besides, we have seen that given enough reliable training samples, different architectures will show similar performance, so the most interesting question in my opinion is not how to fine-tune a model to a dataset, but rather whether it is possible for a model to generalize between different datasets.

Given that we are already considering a classification term within the reconstruction loss and that we have seen how changing its relevance alters classification results, in the next experiments we will see whether this term is enough for us to get rid of the fine-tuning phase we previously had. Clearly using only the simulated dataset will not be enough, so instead we will consider a different approach: semi-supervised learning.

While I do not have enough reliably labeled light-curves, I did put together a dataset of light-curves that are labeled with low probabilities or unlabeled. It is still not a big dataset (it has $\approx 5,000$ objects) and it is obviously not balanced, but it is still bigger than the one we are using now as real test set. The light-curves in this dataset were scraped from both Mars and Lasair brokers, and their low probability labels were obtained from the Transient Name Server. This is further described in Section 4.7.1.

In the next experiments the learning process will go as follows: first, we will reconstruct the simulated light curves, setting a high k_1, so the network focuses on features used for classification while still reconstructing. Then, in the second stage, we will give the network the real light-curves with unreliable labels, but we will set k_1 to a low value, so the model focuses on reconstruction. After that, we

will classify both the simulated and the real test set in two scenarios: one where we fine-tune the network with simulated data for only a few epochs and one were we run the tests sets through the classifier with no further classification training other than whatever was learnt during reconstruction.

Results are presented in Table 5.5, where we can see different scores for three different datasets: the simulated test set, the real test set, and the dataset with low probability labels (Low Prob), which was reconstructed as a second stage during training. Although I gave the reconstruction phase using this dataset a maximum budget of 50 epochs, in all cases the models stopped learning before reaching it, as shown by the third column (R.E for Reconstruction epoch). In contrast, during the reconstruction training using the simulated dataset, the same process reached ≈ 100 epochs for all cases.

Reconstructing the low probability dataset, however, did influence the reconstruction losses of the other two datasets. Compared with the reconstruction losses presented in all previous Sections, the reconstruction loss for the real data decreased significantly, while the reconstruction loss for the simulated dataset increased. We can see there is a trade-off: the model can learn to better reconstruct the real data only at the cost of getting worse at reconstructing the simulated one.

Something similar happens with the F1 classification scores. I considered two scenarios, one with a fine-tuning stage using simulated light-curves (F1-ft in the columns in Table 5.5 and one without it. The F1-scores obtained after fine-tuning for both the real and the simulated test sets are very similar to the ones from the previous Sections, that is, the model outputs better scores for the simulated test set, since it is more similar to the data it was trained with. Without fine-tuning however, the situation is reversed: the F1-scores for the real test set are higher.

One could argue that the second reconstruction stage, where the network got to see real light curves, could be considered fine-tuning in a way. However, fine-tuning means 'to adjust precisely' which I would argue is not at all what happened here. While the model did interact with real light-curves, it either did not know or, by design, did not much consider the labels that these light-curves had. Indeed, if we look at the second column in Table 5.5, the value $\hat{k}_1$ represents the k_1 value used during the second reconstruction stage. It is in all cases a lower value than the one used during the initial reconstruction ($k_1 = 0.5$) and it does not appear to impact the classification scores significantly. Besides, these light-curves with

lower or no classification probabilities were few, unbalanced, and not smooth like the ones from the simulated test set. I think the classification scores for the test set did improve because the model had a chance to learn how to deal with 'dirtier' data, but also, I think the F1-scores got better for the real data simply because the model was not tailored to a different dataset.

I did not expect the classification scores for the simulated test set to drop so significantly without fine-tuning. Even though the weights of the network were updated last by the reconstruction of the low probability real light-curves, most of the training time was spent learning from the simulated data. Both the reconstruction loss and the F1 score for the simulated test set improve with higher $\hat{k}_1$ values, which makes sense, since this means that the model focused during reconstruction on learning features that might be useful for classification later. Whether the layers in charge of computing μ and σ^2 were frozen or not during the second reconstruction stage does not seem to make much of a difference.

While overall the classification results for the real test are still not as good as those of the simulated data, this is to be expected given how the training process was set up. Still, they improved considerably. Figure 5.7 shows the confusion matrices for one of the runs, where the results over the real test set when there was no fine-tunning phase are actually better than those of the simulated test set.

5.3.4 Discussion

While at the beginning of this Chapter it seemed like an encoder was enough for classification, and indeed the literature seemed to support it, we have seen that a decoder can be helpful to extract relevant features and even improve the results, provided that the data and the architecture allow for it. Although all decoders seen in Section 5.3.2 performed similarly, the attention-based decoder obtained the best results.

When I started this Chapter I had planed to investigate whether I could use a sequence-to-sequence model to generate light-curves similar to the real ones and then use those as a training set. As I went along I realized how sampling from distributions that were learnt from other light curves is very similar to using a template. Since the templates that we would have obtained would have been learnt from simulated light curves, which were themselves generated from actual

		Low prob		Sim. test set			Real test set		
Freeze	$\hat{k_1}$	R.E	R.Loss	R.Loss	F1	F1-ft	R.Loss	F1	F1-ft
LS	0.01	24	0.651	0.730	0.451	0.701	0.464	0.586	0.485
LS	0.01	40	0.627	0.904	0.320	0.601	0.456	0.586	0.409
LS	0.01	26	0.637	0.586	0.517	0.679	0.430	0.618	0.469
LS	0.1	24	0.640	0.602	0.508	0.644	0.445	0.584	0.436
LS	0.1	38	0.627	1.121	0.219	0.651	0.446	0.608	0.438
LS	0.1	46	0.623	0.858	0.350	0.691	0.474	0.589	0.519
LS	0.3	26	0.647	0.669	0.505	0.654	0.446	0.595	0.431
LS	0.3	23	0.645	0.653	0.505	0.706	0.437	0.614	0.448
LS	0.3	29	0.641	0.644	0.468	0.629	0.449	0.585	0.462
-	0.01	24	0.634	0.793	0.367	0.630	0.451	0.557	0.361
-	0.01	42	0.635	0.821	0.346	0.620	0.453	0.612	0.386
-	0.01	30	0.637	0.707	0.456	0.673	0.509	0.564	0.497
-	0.1	27	0.640	0.871	0.381	0.656	0.455	0.599	0.481
-	0.1	32	0.637	0.806	0.415	0.672	0.449	0.624	0.411
-	0.1	32	0.635	0.665	0.492	0.718	0.472	0.530	0.462
-	0.3	21	0.648	0.680	0.494	0.611	0.475	0.599	0.469
-	0.3	32	0.631	0.630	0.485	0.710	0.431	0.601	0.459
-	0.3	37	0.622	0.756	0.442	0.653	0.504	0.498	0.450

Table 5.5 Reconstruction and classification results for the real test set, the simulated test set and a real dataset made of light-curves with uncertain labels (Low Probs). The first column encodes whether the latent space (LS) was frozen or not during the reconstruction of the low probability light-curves. The second column $\hat{k_1}$ shows the value that was used as weight for the classification term within the reconstruction loss, during the reconstruction of the low probability light-curves. R.E stands for reconstruction epoch and it shows how many epochs this process took before the network stopped learning. F1-scores for each row are marked in blue, with the best ones in darker shades.

templates, it did not seem it would have helped to generalize from one dataset to another. I then considered using the real light curves I already had to generate

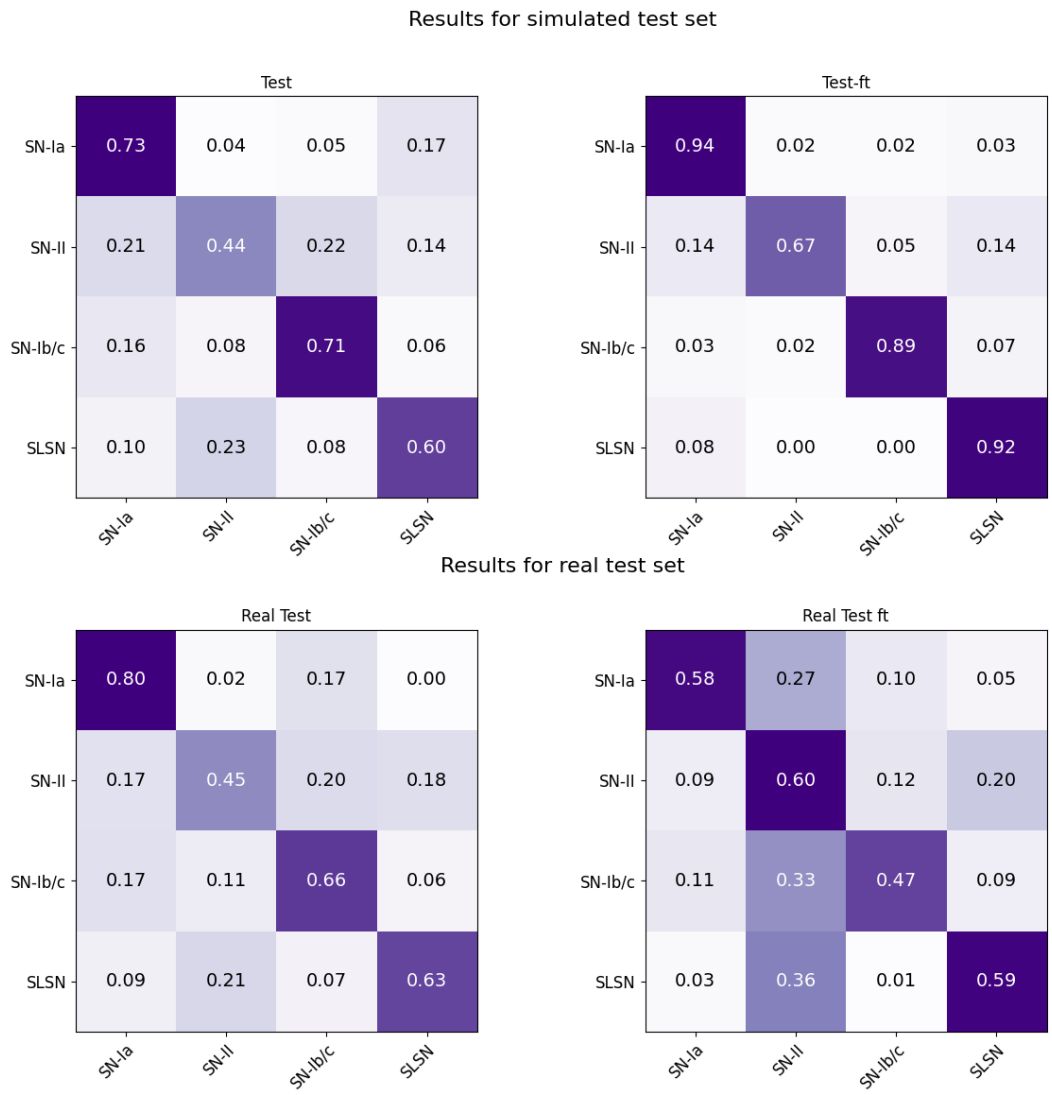


Figure 5.7 Confusion matrices of the TransformerVAE arhictecture over the real test set and the simulated test set.

more from them, by giving them to the decoder in the transformer VAE as targets while at the same time sampling a fake encoded representation $\hat{z} = \mu + \epsilon \times \sigma^2$. This works, but unfortunately the light curves produced are similar enough that using them would defeat the purpose of having a test set. Given enough data and a better prior, I think this would be possible, though perhaps the prior would have to be learnt instead of set to being a standard normal.

We have seen in the last two Chapters how attention based mechanisms can be useful for issues related to Supernova light-curve classification that we have encountered throughout this thesis. In the next Chapter, we will briefly summarize the findings we have gained from this thesis and discuss the directions

in which future research may go to further investigate how to accurately and efficiently classify Supernovae light curves.

Chapter 6

Conclusion

If I were to be completely truthful, I would have to admit that at the very beginning of what has been my research journey I did not anticipate ending up at the stage where I am at. I did not start with a clearly defined research question and I did not have an aim for what exactly it was that I wanted to achieve. All I knew was that I felt eager to learn and that the opportunity of undergoing these studies allowed me an entrance to two broad fields that I knew little about and that seemed to me to be immensely interesting: the way machines learn and how things explode in the sky. It seemed unreal that they could intersect.

Gradually, research questions did start to emerge, form and solidify. Throughout this thesis I looked into a very specific problem, Supernova light curve classification. Rather than focusing on solving the task, it seems to me that the real question is how can we extract relations from these light curves that are general enough for us to be able to use them when confronted with similar, yet very different, data. Classification techniques often work well with light curve simulations but then struggle against real light curves.

In Chapter 4, I took great care to build a simulated dataset that resembled ZTF light curves because I wanted to be able to test whatever model I built against real light curves. This allowed me to test three widely used architectures (FCNs, ResNets and GRUs) against one another, and see how they achieved similar performance. I then proposed a model that extended a GRU with a Self-Attention mechanism, hypothesizing that such a mechanism could perhaps obtain important information from a light curve early on. While this proved to be the case for simulations (part of the PLAsTiCC dataset and my own simulated

dataset), the main contribution from that Chapter was finding that attention could mitigate the issue of having two different datasets. While it by no means solved the problem, it definitely presented a possibility.

I decided to look into this further in Chapter 5. I first looked into three different types of data representations. Though there are several ways in which one could represent a multi-variate time series, from those experiments I concluded that for light-curve classification and for the transformer architecture in particular, it is of important to express where the gaps in the light curves are. In other words, cadence is an important feature of a light-curve, and it being even is an important feature for a neural network.

I also tried three sequence-to-sequence architectures in order to see whether features extracted during light-curve reconstruction could be also used for classification. We saw that this is not necessarily the case, and often, having a model that extracts data representations to pass directly to a classifier (such as an encoder) is enough. This changed with the introduction of a probabilistic model that also made use of Attention: a Transformer Variational Auto-Encoder.

Framing the transformer architecture as a probabilistic model allowed for data reconstruction in a way that is more akin to generation, since by learning feature distributions we can later sample from them. I think it is this characteristic that allowed for better generalization. Having a regular latent space allowed for the model to learn representations from simulations and then, after seeing only a few samples of unbalanced, unlabeled, sparse noisy light curves, it was able to obtain decent classification scores. On their own, those light-curves would not have been useful for training. The simulations on their own were not representative enough of the real data to be effective for generalization. But being able to combine both different datasets into one latent space is what I think not only improved classification scores for the test set but also made the models more robust to changes in hyper-parameters.

It is not only about the probabilistic framework. Indeed, there has been work (Pasquet et al., 2019) done using VAEs for light-curve analysis and seeing whether feature representations can be generalized across datasets. The authors, however, admit that in the case of real photometric datasets, their model does not generalize well. On the other hand, I implemented a naive version of a VAE and embedded it into the Transformer, and it immediately improved performance. I think it is the combination of both Attention and a probabilistic framework,

which in the recent years has been more discussed (Henderson & Fehr, 2022; Wang & Wan, 2019; Movellan & Gabbur, 2020; Tang & Matteson, 2021; Dehaene & Brossard, 2021), that might prove useful for tasks that require a higher degree of generalization, such as transfer learning between different (and in some ways similar) astronomical datasets.

The approach proposed in Section 5.3.3 also has potential to be used for synthetic light curve generation for data augmentation. In order to explore this possibility further, it would be interesting to inspect alternative regularization losses and different ways of organizing the probabilistic latent space of a βVAE. These possibilities are already being investigated in different research fields, both for univariate (Kavran et al., 2022) and multivariate (Desai et al., 2021; Iglesias et al., 2023) time series. VAEs could also be useful for reconstructing noisy and sparse light curves, as they have also been used for data imputation (Fortuin et al., 2020).

I do think the field of research of light curve analysis and classification could benefit in the future from looking further both into Attention and probabilistic approaches. Given how close we are as a community to be getting huge amounts of data from new surveys, it is of paramount importance that we look at the data we already have and consider this issue carefully. While we do not know yet what we will learn about the universe from this new surveys, we can be sure that many interesting, expected and unexpected challenges will be gifted to us. It will definitely be an experience, one worth being alive for.

Bibliography

AMD 2008, Radeon R3xx 3D Register Reference Guide, https://www.x.org/docs/AMD/old/R3xx_3D_Registers.pdf

Agnes P., et al., 2021, Journal of Cosmology and Astroparticle Physics, 2021, 043

Albawi S., Abed Mohammed T., ALZAWI S., 2017. , doi:10.1109/ICEngTechnol.2017.8308186

Allam T., McEwen J. D., 2021, Paying Attention to Astronomical Transients: Photometric Classification with the Time-Series Transformer, doi:10.48550/ARXIV.2105.06178, https://arxiv.org/abs/2105.06178

Allam Jr T., et al., 2018, arXiv preprint arXiv:1810.00001

Antoniou A., Słowik A., Crowley E. J., Storkey A., 2018, Dilated DenseNets for Relational Reasoning (arXiv:1811.00410)

Bagnall A., Lines J., Hills J., Bostrom A., 2015, IEEE Transactions on Knowledge and Data Engineering, 27, 2522

Bagnall A., Bostrom A., Large J., Lines J., 2016, The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version (arXiv:1602.01711)

Bagnall A., Lines J., Bostrom A., Large J., Keogh E., 2017, Data Mining and Knowledge Discovery, 31

Bahdanau D., Cho K., Bengio Y., 2015, arXiv e-prints, p. arXiv:1409.0473

Bassi S., Sharma K., Gomekar A., 2021, Frontiers in Astronomy and Space Sciences, 8

Bellm E. C., 2014, The Zwicky Transient Facility (arXiv:1410.8185)

Bellm E. C., et al., 2018, Publications of the Astronomical Society of the Pacific, 131, 018002

Belokurov V., Evans N. W., Du Y. L., 2003, Monthly Notices of the Royal Astronomical Society, 341, 1373

Bhandari S., et al., 2020, The Astrophysical Journal, 895, L37

Bhatnagar S., et al., 2018, in Müller V. C., ed., Philosophy and Theory of Artificial Intelligence 2017. Springer International Publishing, Cham, pp 117–135

Bloom J. S., et al., 2012, Publications of the Astronomical Society of the Pacific, 124, 1175

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

Bridle J. S., 1989, in NATO Neurocomputing.

Brown T. B., et al., 2020, Language Models are Few-Shot Learners (arXiv:2005.14165)

Bruenn S. W., et al., 2020, The Astrophysical Journal Supplement Series, 248, 11

Brunel A., Pasquet J., Pasquet J., Rodriguez N., Comby F., Fouchez D., Chaumont M., 2019, A CNN adapted to time series for the classification of Supernovae (arXiv:1901.00461)

Burhanudin U. F., et al., 2021, Monthly Notices of the Royal Astronomical Society, 505, 4345

Burrows A., Vartanyan D., 2021, Core-Collapse Supernova Explosion Theory (arXiv:2009.14157)

Cahuantzi R., Chen X., Güttel S., 2021, A comparison of LSTM and GRU networks for learning symbolic sequences (arXiv:2107.02248)

Cardelli J. A., Clayton G. C., Mathis J. S., 1989, , 345, 245

Carrick J. E., Hook I. M., Swann E., Boone K., Frohmaier C., Kim A. G., Sullivan M., 2021, Monthly Notices of the Royal Astronomical Society, 508, 1–18

Chan M. L., Heng I. S., Messenger C., 2020, Physical Review D, 102

Chandrasekhar S., 1931, , 74, 81

Charnock T., Moss A., 2017a, The Astrophysical Journal Letters

Charnock T., Moss A., 2017b, The Astrophysical Journal, 837, L28

Chen K.-J., 2021, International Journal of Modern Physics D, 30, 2130001

Cheng J., Dong L., Lapata M., 2016, CoRR, abs/1601.06733

Cho K., van Merrienboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y., 2014, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation (arXiv:1406.1078)

Churazov E., et al., 2014, Nature, 512, 406–408

Ciresan D., Meier U., Masci J., Gambardella L. M., Schmidhuber J., 2011. pp 1237–1242, doi:10.5591/978-1-57735-516-8/IJCAI11-210

Coelho R. C. V., Calvão M. O., Reis R. R. R., Siffert B. B., 2014, European Journal of Physics, 36, 015007

Collaboration: D. E. S., et al., 2016, Monthly Notices of the Royal Astronomical Society, 460, 1270

Collins C., Dennehy D., Conboy K., Mikalef P., 2021, International Journal of Information Management, 60, 102383

Cover T. M., Thomas J. A., 2006, Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing). Wiley-Interscience, USA

Crampton D., Simard L., Silva D., 2009, in Moorwood A., ed., Science with the VLT in the ELT Era. Springer Netherlands, Dordrecht, pp 279–285

Dai J.-P., Yang Y., Xia J.-Q., 2018, The Astrophysical Journal, 857, 9

Dau H. A., Bagnall A., Kamgar K., Yeh C.-C. M., Zhu Y., Gharghabi S., Ratanamahatana C. A., Keogh E., 2019, The UCR Time Series Archive (arXiv:1810.07758)

De Cia A., et al., 2018, , 860, 100

Dehaene D., Brossard R., 2021, Re-parameterizing VAEs for stability, doi:10.48550/ARXIV.2106.13739, https://arxiv.org/abs/2106.13739

Desai A., Freeman C., Wang Z., Beaver I., 2021, TimeVAE: A Variational Auto-Encoder for Multivariate Time Series Generation (arXiv:2111.08095)

Devlin J., Chang M.-W., Lee K., Toutanova K., 2018, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, doi:10.48550/ARXIV.1810.04805, https://arxiv.org/abs/1810.04805

Dexter J., Kasen D., 2013, , 772, 30

Dhar Gupta K., Pampana R., Vilalta R., Ishida E., de Souza R., 2016. pp 1–8, doi:10.1109/SSCI.2016.7849951

Dilday B., et al., 2008, The Astrophysical Journal, 682, 262

Djorgovski S., et al., 2011a, arXiv:1102.5004

Djorgovski S., et al., 2011b, arXiv:1102.5004

D'Isanto A., Cavuoti S., Brescia M., Donalek C., Longo G., Riccio G., Djorgovski S. G., 2016, Monthly Notices of the Royal Astronomical Society, 457, 3119–3132

Fawaz H. I., 2020, Deep learning for time series classification (arXiv:2010.00567)

Feindt U., Nordin J., Rigault M., Brinnel V., Dhawan S., Goobar A., Kowalski M., 2019, Journal of Cosmology and Astroparticle Physics, 2019, 005

Fitch F. B., 1944, Journal of Symbolic Logic, p. 49–50

Foglizzo T., Scheck L., Janka H., 2005, The Astrophysical Journal, 652

Foley R. J., et al., 2013, The Astrophysical Journal, 767, 57

Fortuin V., Baranchuk D., Rätsch G., Mandt S., 2020, GP-VAE: Deep Probabilistic Time Series Imputation (arXiv:1907.04155)

Fremling U. C., et al., 2019, The Zwicky Transient Facility Bright Transient Survey I: Spectroscopic Classification and the Redshift Completeness of Local Galaxy Catalogs, doi:10.48550/ARXIV.1910.12973, https://arxiv.org/abs/1910.12973

Fukushima K., 1980, Biol Cybern

Förster F., et al., 2021, The Astronomical Journal, 161, 242

Gabruseva T., Zlobin S., Wang P., 2019, Photometric light curves classification with machine learning (arXiv:1909.05032)

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

Gamboa J. C. B., 2017, Deep Learning for Time-Series Analysis (arXiv:1701.01887)

Gamezo V. N., Khokhlov A. M., Oran E. S., Chtchelkanova A. Y., Rosenberg R. O., 2003, Science, 299, 77–81

Gamma E., Helm R., Johnson R., Vlissides J., 1995, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co., Inc., USA

Gilliland R. L., Nugent P. E., Phillips M. M., 1999, , 521, 30

Glorot X., Bordes A., Bengio Y., 2011, in Gordon G., Dunson D., Dudík M., eds, Proceedings of Machine Learning Research Vol. 15, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. PMLR, Fort Lauderdale, FL, USA, pp 315–323, http://proceedings.mlr.press/v15/glorot11a.html

Gold O., Sharir M., 2018, ACM Trans. Algorithms, 14

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

Graves A., Schmidhuber J., 2009, in Koller D., Schuurmans D., Bengio Y., Bottou L., eds, Vol. 21, Advances in Neural Information Processing Systems. Curran Associates, Inc., https://proceedings.neurips.cc/paper/2008/file/66368270ffd51418ec58bd793f2d9b1b-Paper.pdf

Guiglion G., et al., 2019, 4MOST Survey Strategy Plan, doi:10.18727/0722-6691/5119

Guy J., et al., 2007, Astronomy &amp$$ Astrophysics, 466, 11

Hachisu I., Kato M., Saio H., Nomoto K., 2011, The Astrophysical Journal, 744, 69

Hamacher D. W., 2014, Are Supernovae Recorded in Indigenous Astronomical Traditions? (arXiv:1404.3253)

Hamuy M., Phillips M. M., Suntzeff N. B., Schommer R. A., Maza J., Aviles R., 1996, The Astronomical Journal, 112, 2391

He K., Zhang X., Ren S., Sun J., 2015, Deep Residual Learning for Image Recognition (arXiv:1512.03385)

Heger A., Fryer C. L., Woosley S. E., Langer N., Hartmann D. H., 2003, The Astrophysical Journal, 591, 288–300

Helder E. A., et al., 2009, Science, 325, 719–722

Henderson J., Fehr F., 2022, A Variational AutoEncoder for Transformers with Nonparametric Variational Information Bottleneck, doi:10.48550/ARXIV.2207.13529, https://arxiv.org/abs/2207.13529

Hillebrandt W., Niemeyer J. C., 2000, Annual Review of Astronomy and Astrophysics, 38, 191–230

Hillebrandt W., Kromer M., Röpke F. K., Ruiter A. J., 2013, Towards an understanding of Type Ia supernovae from a synthesis of theory and observations (arXiv:1302.6420)

Hinton G. E., Sejnowski T. J., 1999.

Hložek R., et al., 2020, Results of the Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC), doi:10.48550/ARXIV.2012.12392, https://arxiv.org/abs/2012.12392

Ho T. K., 1995, in Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1. ICDAR '95. IEEE Computer Society, USA, p. 278

Hsiao E. Y., Conley A., Howell D. A., Sullivan M., Pritchet C. J., Carlberg R. G., Nugent P. E., Phillips M. M., 2007, , 663, 1187

Huynh M., Lazio J., 2013, An Overview of the Square Kilometre Array (arXiv:1311.4288)

Hüsken M., Stagge P., 2003, Neurocomputing, 50, 223

Iben Icko J., Tutukov A. V., 1991, , 370, 615

Ibsen A., Mann B., 2020, in Pizzo R., Deul E. R., Mol J. D., de Plaa J., Verkouter H., eds, Astronomical Society of the Pacific Conference Series Vol. 527, Astronomical Data Analysis Software and Systems XXIX. p. 167

Iglesias G., Talavera E., González-Prieto Á., Mozo A., Gómez-Canaval S., 2023, Neural Computing and Applications, 35, 10123

Informatik F., Bengio Y., Frasconi P., Schmidhuber J., 2003, A Field Guide to Dynamical Recurrent Neural Networks

Inserra C., Prajs S., Gutierrez C. P., Angus C., Smith M., Sullivan M., 2018, The Astrophysical Journal, 854, 175

Ishida E. E. O., de Souza R. S., 2013, Monthly Notices of the Royal Astronomical Society, 430, 509

Ishida E. E. O., et al., 2018, Monthly Notices of the Royal Astronomical Society, 483, 2–18

Ismail Fawaz H., Forestier G., Weber J., Idoumghar L., Muller P.-A., 2019, Data Mining and Knowledge Discovery, 33, 917–963

Ivezić , et al., 2019a, The Astrophysical Journal, 873, 111

Ivezić , et al., 2019b, The Astrophysical Journal, 873, 111

Janka H.-T., 2001, Astronomy &amp$$ Astrophysics, 368, 527

Janocha K., Czarnecki W. M., 2017, On Loss Functions for Deep Neural Networks in Classification (arXiv:1702.05659)

Jerkstrand A., 2017, in , Handbook of Supernovae. Springer International Publishing, pp 795–842, doi:10.1007/978-3-319-21846-5_29, https://doi.org/10.1007%2F978-3-319-21846-5_29

Jha S. W., 2017, in , Handbook of Supernovae. Springer International Publishing, pp 375–401, doi:10.1007/978-3-319-21846-5_42, https://doi.org/10.1007%2F978-3-319-21846-5_42

Jiang J., Xia G. G., Carlton D. B., Anderson C. N., Miyakawa R. H., 2020, in ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp 516–520, doi:10.1109/ICASSP40776.2020.9054554

Joiner I. A., 2018, in Joiner I. A., ed., Chandos Information Professional Series, Emerging Library Technologies. Chandos Publishing, pp 1–22, doi:https://doi.org/10.1016/B978-0-08-102253-5.00002-2, https://www.sciencedirect.com/science/article/pii/B9780081022535000022

Jones R., 2021, What is DLSS and which games are supported?, https://www.trustedreviews.com/news/what-is-dlss-4110546

Karpenka N. V., Feroz F., Hobson M. P., 2012, Monthly Notices of the Royal Astronomical Society, 429, 1278–1285

Kasen D., Woosley S. E., Heger A., 2011a, The Astrophysical Journal, 734, 102

Kasen D., Woosley S. E., Heger A., 2011b, The Astrophysical Journal, 734, 102

Kavran D., Žalik B., Lukač N., 2022, in Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,. SciTePress, pp 15–23, doi:10.5220/0010749200003116

Ke G., He D., Liu T.-Y., 2020,] 10.48550/ARXIV.2006.15595

Kelley H. J., 1960, Ars Journal, 30, 947

Kessler R., et al., 2010, Publications of the Astronomical Society of the Pacific, 122, 1415

Kessler R., et al., 2019, , 131, 094501

Khan S., Naseer M., Hayat M., Zamir S. W., Khan F. S., Shah M., 2022, ACM Computing Surveys, 54, 1

Kingma D. P., Ba J., 2014, CoRR, abs/1412.6980

Kingma D. P., Welling M., 2013, Auto-Encoding Variational Bayes, doi:10.48550/ARXIV.1312.6114, https://arxiv.org/abs/1312.6114

Koppel A., Pradhan H., Rajawat K., 2020, Consistent Online Gaussian Process Regression Without the Sample Complexity Bottleneck, doi:10.48550/ARXIV.2004.11094, https://arxiv.org/abs/2004.11094

Langer N., Yoon S. C., Wellstein S., Scheithauer S., 2002, in Gänsicke B. T., Beuermann K., Reinsch K., eds, Astronomical Society of the Pacific Conference Series Vol. 261, The Physics of Cataclysmic Variables and Related Objects. p. 252

Lecun Y., 1992, A theoretical framework for back-propagation. IEEE Computer Society Press

Lecun Y., Bengio Y., 1995.

Lecun Y., Bottou L., Bengio Y., Haffner P., 1998, Proceedings of the IEEE, 86, 2278

Legg S., Hutter M., 2007, CoRR, abs/0706.3639

Levan A., et al., 2005, , 624, 880

Li S., Jin X., Xuan Y., Zhou X., Chen W., Wang Y.-X., Yan X., 2019, Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting, doi:10.48550/ARXIV.1907.00235, https://arxiv.org/abs/1907.00235

Lim B., Arik S. O., Loeff N., Pfister T., 2019, Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting, doi:10.48550/ARXIV.1912.09363, https://arxiv.org/abs/1912.09363

Lin M., Chen Q., Yan S., 2013a, CoRR, abs/1312.4400

Lin M., Chen Q., Yan S., 2013b, CoRR, abs/1312.4400

Lin Z., Feng M., dos Santos C. N., Yu M., Xiang B., Zhou B., Bengio Y., 2017, CoRR, abs/1703.03130

Lines J., Taylor S., Bagnall A., 2016, in 2016 IEEE 16th International Conference on Data Mining (ICDM). pp 1041–1046, doi:10.1109/ICDM.2016.0133

Liu D., Wang B., Han Z., 2017, Monthly Notices of the Royal Astronomical Society, 473, 5352

Liu M., Ren S., Ma S., Jiao J., Chen Y., Wang Z., Song W., 2021, Gated Transformer Networks for Multivariate Time Series Classification, doi:10.48550/ARXIV.2103.14438, https://arxiv.org/abs/2103.14438

Lochner M., McEwen J. D., Peiris H. V., Lahav O., Winter M. K., 2016, The Astrophysical Journal Supplement Series, 225, 31

Long J., Shelhamer E., Darrell T., 2015, Fully Convolutional Networks for Semantic Segmentation (arXiv:1411.4038)

Loshchilov I., Hutter F., 2017, CoRR, abs/1711.05101

Lunnan R., et al., 2018, A UV Resonance Line Echo from a Shell Around a Hydrogen-Poor Superluminous Supernova, doi:10.48550/ARXIV.1808.04887, https://arxiv.org/abs/1808.04887

Luong M.-T., Pham H., Manning C. D., 2015, Effective Approaches to Attention-based Neural Machine Translation (arXiv:1508.04025)

Mabanta Q. A., Murphy J. W., 2018, The Astrophysical Journal, 856, 22

Madau P., Dickinson M., 2014, Annual Review of Astronomy and Astrophysics, 52, 415

Mahabal A., et al., 2008, AIP Conference Proceedings

Mahabal A., Sheth K., Gieseke F., Pai A., Djorgovski S. G., Drake A. J., Graham M. J., 2017, 2017 IEEE Symposium Series on Computational Intelligence (SSCI)

Malhotra P., TV V., Vig L., Agarwal P., Shroff G., 2017, TimeNet: Pre-trained deep recurrent neural network for time series classification (arXiv:1706.08838)

Malz A. I., et al., 2019, The Astronomical Journal, 158, 171

Matheson T., et al., 2021, The Astronomical Journal, 161

Mazzali P. A., Ropke F. K., Benetti S., Hillebrandt W., 2007, Science, 315, 825–828

McCully C., et al., 2014, Nature, 512, 54–56

Mies M. . S. V., 1993, Ecofeminism

Mislis D., Bachelet E., Alsubai K. A., Bramich D. M., Parley N., 2015, Monthly Notices of the Royal Astronomical Society, 455, 626–633

Mitchell T. M., 1997, Machine Learning. McGraw-Hill, New York

Morgan R., et al., 2021, DeepZipper: A Novel Deep Learning Architecture for Lensed Supernovae Identification (arXiv:2112.01541)

Morozova V., Radice D., Burrows A., Vartanyan D., 2018, The Astrophysical Journal, 861, 10

Morvan M., Nikolaou N., Yip K. H., Waldmann I., 2022, Don't Pay Attention to the Noise: Learning Self-supervised Representations of Light Curves with a Denoising Time Series Transformer, doi:10.48550/ARXIV.2207.02777, https://arxiv.org/abs/2207.02777

Movellan J. R., Gabbur P., 2020, Probabilistic Transformers, doi:10.48550/ARXIV.2010.15583, https://arxiv.org/abs/2010.15583

Murphy K. P., 2012, Machine Learning: A Probabilistic Perspective. The MIT Press

Murtagh F., 1991, Neurocomputing, 2, 183

Muthukrishna D., Narayan G., Mandel K. S., Biswas R., Hložek R., 2019, Publications of the Astronomical Society of the Pacific, 131, 118002

Möller A., de Boissière T., 2019, Monthly Notices of the Royal Astronomical Society, 491, 4277–4293

Möller A., et al., 2016, Journal of Cosmology and Astroparticle Physics, 2016, 008–008

Müller M., 2007, Information Retrieval for Music and Motion, 2, 69

Müller B., Melson T., Heger A., Janka H.-T., 2017, Monthly Notices of the Royal Astronomical Society, 472, 491

NVIDIA 2021, ACCELERATED COMPUTING AND THE DEMOCRATIZATION OF SUPERCOMPUTING, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/sc18-tesla-democratization-tech-overview-r4-web.pdf

Nair V., Hinton G. E., 2010, in Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10. Omnipress, USA, pp 807–814, http://dl.acm.org/citation.cfm?id=3104322.3104425

Naul B., Bloom J. S., Pérez F., van der Walt S., 2017, Nature Astronomy, 2, 151–155

Neuhäuser R., Ehrig-Eggert C., Kunitzsch P., 2017, Astronomische Nachrichten, 338, 19–25

Nicholl M., 2021, Astronomy &amp$$ Geophysics, 62, 5.34

Nicholl M., et al., 2016, , 826, 39

Nicholl M., Berger E., Margutti R., Blanchard P. K., Guillochon J., Leja J., Chornock R., 2017, The Astrophysical Journal, 845, L8

Nugent P., Kim A., Perlmutter S., 2002, , 114, 803

Nun I., Protopapas P., Sim B., Zhu M., Dave R., Castro N., Pichara K., 2015, FATS: Feature Analysis for Time Series, doi:10.48550/ARXIV.1506.00010, https://arxiv.org/abs/1506.00010

O'Connor E., et al., 2018, Journal of Physics G: Nuclear and Particle Physics, 45, 104001

O'Dea M. Lagisz M. D. J. S. N., 2018, Nature Communications

Orr G., Müller K., 2003, Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, Springer Berlin Heidelberg, https://books.google.cl/books?id=VCKqCAAAQBAJ

Pascanu R., Mikolov T., Bengio Y., 2013, On the difficulty of training Recurrent Neural Networks (arXiv:1211.5063)

Pasquet J., Pasquet J., Chaumont M., Fouchez D., 2019, Astronomy & Astrophysics, 627, A21

Paulus R., Xiong C., Socher R., 2017, CoRR, abs/1705.04304

Pejcha O., 2020, in , Reviews in Frontiers of Modern Astrophysics. Springer International Publishing, pp 189–211, doi:10.1007/978-3-030-38509-5_7, https://doi.org/10.1007%2F978-3-030-38509-5_7

Phillips M. M., 1993, , 413, L105

Pichara K., Protopapas P., 2013, The Astrophysical Journal, 777, 83

Pimentel Ó ., Estévez P. A., Förster F., 2022, The Astronomical Journal, 165, 18

Piro A. L., Thompson T. A., Kochanek C. S., 2014, Monthly Notices of the Royal Astronomical Society, 438, 3456–3464

Poole D., Mackworth A., Goebel R., 1998, Computational Intelligence: A Logical Approach

Postnov K. A., Yungelson L. R., 2006, Living Reviews in Relativity, 9

Pu F. A.-C., 2008, Nvidia's CUDA: The End of the CPU?, https://www.tomshardware.com/reviews/nvidia-cuda-gpu,1954.html

Qu H., Sako M., Möller A., Doux C., 2021, The Astronomical Journal, 162, 67

Quimby R. M., et al., 2018, , 855, 2

Rada W., Neuhäuser R., 2015, Astronomische Nachrichten, 336, 249–257

Raina R., Madhavan A., Ng A. Y., 2009, in Proceedings of the 26th Annual International Conference on Machine Learning. ICML '09. Association for Computing Machinery, New York, NY, USA, p. 873–880, doi:10.1145/1553374.1553486, https://doi.org/10.1145/1553374.1553486

Ramachandran P., Parmar N., Vaswani A., Bello I., Levskaya A., Shlens J., 2019, CoRR, abs/1906.05909

Ranzato M., Huang F. J., Boureau Y., LeCun Y., 2007, in 2007 IEEE Conference on Computer Vision and Pattern Recognition. pp 1–8, doi:10.1109/CVPR.2007.383157

Richards J. W., Homrighausen D., Freeman P. E., Schafer C. M., Poznanski D., 2011, Monthly Notices of the Royal Astronomical Society, 419, 1121–1135

Riess A. G., et al., 1998, The Astronomical Journal, 116, 1009–1038

Robbins H., Monro S., 1951, The Annals of Mathematical Statistics, 22, 400

Rosenblatt F., 1958, Psychological Review, pp 65–386

Ruder S., 2016, CoRR, abs/1609.04747

Ruiz A., Flynn M., Large J., Middlehurst M., Bagnall A., 2021, Data Mining and Knowledge Discovery, 35, 1

Rumelhart H. . W., 1986, Nature 323

Russakovsky O., et al., 2015, ImageNet Large Scale Visual Recognition Challenge (arXiv:1409.0575)

Russell S., Norvig P., 2009, Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, USA

Rußwurm M., Körner M., 2020, ISPRS Journal of Photogrammetry and Remote Sensing, 169, 421

Ryder S. D., Sadler E. M., Subrahmanyan R., Weiler K. W., Panagia N., Stockdale C., 2004, Monthly Notices of the Royal Astronomical Society, 349, 1093–1100

Saio H., Nomoto K., 1985, , 150, L21

Samuel A. L., 1959, IBM Journal of Research and Development, 3, 210

Schlegel D. J., Finkbeiner D. P., Davis M., 1998, The Astrophysical Journal, 500, 525

Schmidt B. P., et al., 1998, The Astrophysical Journal, 507, 46

Seadrow S., Burrows A., Vartanyan D., Radice D., Skinner M. A., 2018, Monthly Notices of the Royal Astronomical Society, 480, 4710

Shankaranarayana S. M., Runje D., 2021, Attention Augmented Convolutional Transformer for Tabular Time-series, doi:10.48550/ARXIV.2110.01825, `https://arxiv.org/abs/2110.01825`

Shannon C. E., 1948, The Bell System Technical Journal, 27, 379

Shen K. J., Bildsten L., Kasen D., Quataert E., 2012, , 748, 35

Sherstinsky A., 2020, Physica D: Nonlinear Phenomena, 404, 132306

Smith K. W., et al., 2019, Research Notes of the AAS, 3, 26

Solheim J. E., Yungelson L. R., 2005, in Koester D., Moehler S., eds, Astronomical Society of the Pacific Conference Series Vol. 334, 14th European Workshop on White Dwarfs. p. 387 (`arXiv:astro-ph/0411053`)

Solin A., Hensman J., Turner R. E., 2018, Infinite-Horizon Gaussian Processes, doi:10.48550/ARXIV.1811.06588, `https://arxiv.org/abs/1811.06588`

Song H., Rajan D., Thiagarajan J. J., Spanias A., 2017, Attend and Diagnose: Clinical Time Series Analysis using Attention Models, doi:10.48550/ARXIV.1711.03905, `https://arxiv.org/abs/1711.03905`

Springenberg J. T., Dosovitskiy A., Brox T., Riedmiller M., 2015, Striving for Simplicity: The All Convolutional Net (`arXiv:1412.6806`)

Strobl C., Malley J., Tutz G., 2009, Psychological methods, 14, 323

Strolger L.-G., et al., 2015, , 813, 93

Sun S., Cao Z., Zhu H., Zhao J., 2019, A Survey of Optimization Methods from a Machine Learning Perspective (`arXiv:1906.06821`)

Sutton R. S., Barto A. G., 2018, Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA

Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna Z., 2015, Rethinking the Inception Architecture for Computer Vision (arXiv:1512.00567)

Sánchez-Sáez P., et al., 2020, Alert Classification for the ALeRCE Broker System: The Light Curve Classifier (arXiv:2008.03311)

Sánchez-Sáez P., et al., 2021, The Astronomical Journal, 161, 141

Tang B., Matteson D. S., 2021, in Beygelzimer A., Dauphin Y., Liang P., Vaughan J. W., eds, Advances in Neural Information Processing Systems. https://openreview.net/forum?id=HfpNVDg3ExA

Tomaney A. B., Crotts A. P. S., 1996, The Astronomical Journal, 112, 2872

Turatto M., 2003, Lecture Notes in Physics, p. 21–36

Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I., 2017, CoRR, abs/1706.03762

Wang T., Wan X., 2019, in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, pp 5233–5239, doi:10.24963/ijcai.2019/727, https://doi.org/10.24963/ijcai.2019/727

Wang B., Justham S., Han Z., 2013, Double-detonation explosions as progenitors of type Iax supernovae (arXiv:1301.1047)

Wang Z., Yan W., Oates T., 2016a, Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline (arXiv:1611.06455)

Wang Z., Yan W., Oates T., 2016b, CoRR, abs/1611.06455

WayBackMachine 2011, IBM 8514 description, https://web.archive.org/web/20140714141930/http://theodor.lauppert.ws/computer/ps2/8514a.html

Webbink R. F., 1984, , 277, 355

Wen Q., Zhou T., Zhang C., Chen W., Ma Z., Yan J., Sun L., 2022, Transformers in Time Series: A Survey, doi:10.48550/ARXIV.2202.07125, https://arxiv.org/abs/2202.07125

Werbos P., 1975, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University, https://books.google.co.uk/books?id=z81XmgEACAAJ

West J., Ventura D., Warnick S., 2007, A Theoretical Foundation for Inductive Transfer, Spring Research Presentation

Whelan J., Iben Icko J., 1973, , 186, 1007

Witheiler 2001, NVIDIA GeForce3 Roundup, https://www.anandtech.com/show/802

Woosley S. E., 2017, The Astrophysical Journal, 836, 244

Woosley S., Janka T., 2005, Nature Physics, 1, 147–154

Wu X., et al., 2007, Knowledge and Information Systems, 14

Wu S., Xiao X., Ding Q., Zhao P., Wei Y., Huang J., 2020, in Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS'20. Curran Associates Inc., Red Hook, NY, USA

Xu K., Ba J., Kiros R., Cho K., Courville A., Salakhutdinov R., Zemel R., Bengio Y., 2015, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, doi:10.48550/ARXIV.1502.03044, https://arxiv.org/abs/1502.03044

Ye L., Keogh E., 2009, in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '09. p. 947–956, doi:10.1145/1557019.1557122, https://doi.org/10.1145/1557019.1557122

Yu C., et al., 2021, WIREs Data Mining and Knowledge Discovery, 11, e1425

Yuan Y., Lin L., 2021, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 14, 474

Zerveas G., Jayaraman S., Patel D., Bhamidipaty A., Eickhoff C., 2020, A Transformer-based Framework for Multivariate Time Series Representation Learning, doi:10.48550/ARXIV.2010.02803, https://arxiv.org/abs/2010.02803

Zhang M. M., Dumitrascu B., Williamson S. A., Engelhardt B. E., 2019, Sequential Gaussian Processes for Online Learning of Nonstationary Functions, doi:10.48550/ARXIV.1905.10003, https://arxiv.org/abs/1905.10003

Zhou H., Zhang S., Peng J., Zhang S., Li J., Xiong H., Zhang W., 2020, Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting, doi:10.48550/ARXIV.2012.07436, https://arxiv.org/abs/2012.07436

Zhou T., Ma Z., Wen Q., Wang X., Sun L., Jin R., 2022, FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting, doi:10.48550/ARXIV.2201.12740, https://arxiv.org/abs/2201.12740