Efficient Deep Learning for Real-time Classification of Astronomical Transients

Author: **Tarek Allam Jr.**

Primary Supervisor:
Prof. Jason McEwen
(Dept. of Space and Climate Physics)

Secondary Supervisor:
Dr Denise Gorse (Dept. of Computer Science)

A dissertation submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy
in
Astrophysics

University College London
Centre for Doctoral Training in Data Intensive Science

Branch: `master`
Version: `v2.0.3`
Most recent commit: `2453c34` (April 24, 2023)

Declaration

I, Tarek Allam Jr, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis. The work contains nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

April 2023

Copyright

Acknowledgements

"*He knows not where he's going, For the ocean will decide, It's not the destination, It's the glory of the ride*"

— **Edward Monkton**, *Zen Dog*

It has been quite the journey. One that began many years before I even knew what a Ph.D was. Not being one for "reading", I found my place with the equations and conceptual diagrams of physics and consumed anything and everything related to science. I did not know it at the time, but my affinity for numbers and pictures, while struggling with words was in large parts due to dyslexia. Always being behind on my writing and reading work made teachers consider me "slow". So I tried to compensate by racing ahead the best way I knew how; collecting science facts and trivia wherever possible, hoping to one day become a scientist myself. I found it fascinating that every scientist seemed to have something called a Ph.D, and wanting to do what they do, without knowing what it was, decided I would one day get a PhD too. At the impressionable age of 10, Bruce Willis' blockbuster movie Armageddon hit the screens. It was a scene from this movie that first made me realise that people with Ph.Ds, particularly in Astrophysics were considered smart. This to me cemented that I really did need to get a PhD in astrophysics. Not only would it help me become a scientist, but if I got one, *maybe people would't think I'm stupid?*[1]. And so began my journey.

With my eyes on the prize (the doctorate), there were of course many twists and turns along the way. Throughout all the tough times, it was the belief from others that kept me going, and helped

[1]Of course people tried to convince me that there was no reason to believe I was stupid anyway, but Oscar Gamble's quote captures the feeling perfectly; "They don't think it be like it is, but it do."

get me to this point. And while words won't do all their support justice, what follows is an unapologetically long list of thank yous to all those who believed in me. You helped me grow into a person who now truly believes in themselves and is excited for the next chapter in life, whatever that may be. I dedicate this work to you.

Thank you to Ms Paice and Ms Faure at Cranford Community College who noticed my mathematical potential and encouraged me to pursue it further. You were the reason I did not leave school at 16 and were the catalyst for me to stop dreaming and start believing that I could do this (PhD). I am forever grateful to Ms Pippa Moore at Royal Holloway University who diagnosed me with dyslexia in my 3rd year of unversity. Better late than never. Thank you for the ample support with not only examination preparation, but also tips on navigating life as someone with such a condition. Also at Royal Holloway, I have to thank my tutor Prof. Glen Cowen for his unbelievable patience with explaining concepts to me as an (often hungover) undergraduate. Similarly, I would like to thank Prof. Stewart Boogert, his passion for scientific computing and command-line wizardry showed me the power of programming, and ultimately inspired me to pursue Computer Science at UCL later on.

At UCL I had the pleasure of meeting incredible people and made some great friends along the way. I feel so lucky to have met Aaron Robertson, my command-line kindred spirit, who would indulge me with conversations on all things technology, and who turned me into the git and vim evangelist I am today. I want to thank Dr Denise Gorse for her mentorship, I am so grateful for all your time and advice. I thoroughly enjoyed our lengthy discussion on how best to improve the signal-to-noise of Otto the Dog. Of course countless thanks to Prof. Jason McEwen, my Masters and PhD supervisor. I'm so thankful for all the encouragement and championship over the years. Thank you for your patience, and thank you for your research guidance, helping me become a better researcher. Thank you Dr Kevin Bryson, who was there for me when times were particularly hard. When I had to interrupt my

studies to care for my mother, you helped me get through such a rough period of time. Another person I want to thank that helped me get through that tough period, but more indirectly, is Dr Dan Moore of Imperial College London. Thank you for allowing me to sneak *into* lectures and tutorials when all I wanted to do was learn, and perhaps even more so, to just have a distraction from it all – thank you for the punch card too. Thank you also to Prof. Daisuke Kawata for literally being that shoulder to cry on when I went through another difficult period in 2018.

I must thank Prof. James Hetherington for introducing me to the Software Sustainability Institute (SSI) and taking the time to sit with me way back in 2015 to map out a pathway to becoming a Research Software Engineer (RSE). Thank you to Dr Samin Ishtiaq who kindly encouraged me to keep to this pathway at the first RSE conference when I was tempted to sway. I met some wonderful people involved with the SSI, many of whom I had the chance to work with during my internship at the Alan Turing Institute (ATI). Thank you to the whole Research Engineering Group (REG) at the ATI who welcomed me with open arms from day one. In particular my NATS teammates, Dr Timothy Hobson, Dr Evelina Gabasova, Ruairidh MacLeod and Dr Radka Jersakova who through pure encouragement and believing in me eased my imposter syndrome. Furthermore, thank you to my desk buddy Dr Sarah Gibson for being that person I could share my PhD woes and laughter with in equal measure. Thank you to Dr Kirstie Whitaker for your guidance on research and community building. Thank you to Dr Rachael Ainsworth for believing in me and giving me a chance to represent the SSI as a fellow.

Thank you to all those at the CDT, but especially my fellow cohort; I have such fond memories of the CDT office in the first year. Thank you to Dr Patrick Roddy for listening to my ramblings of software engineering best practices and for entertaining my time-saving (but really time-wasting) automation ideas. Thank you Dr Davide Piras and Dr Gordon Yip for always being there to laugh with, and for being patient enough to help this very math-

ematically rusty student survive first year coursework. Thank Dr Charlie Donaldson for being the great friend you are, welcoming me to your home and helping to create such wodnerful memories outside of the PhD world along the way. Thank you to those in the Dark Energy Science Collaboration (DESC), in particular Dr Rahul Biswas, Prof. Renée Hložek, Dr Kara Ponder, Dr Gautham Narayan, Dr Alex Malz, who kindly welcomed this complete imposter to the world of cosmology and encouraged my contributions nonetheless. Thank you to Christian Setzer, may we share many more beers and deep discussions about the Universe again soon. Thank you to Catarina Alves, who bared my strong opinions about programming and made me practise what I preach. Thank you Dr David Pérez-Suárez for encouraging me to seek out Google Summer of Code (GSoC) opportunities, which eventually put me in touch with Dr Julien Peloton, someone who I cannot thank enough for his support and belief in me. Through Julien I was able to meet with the wider FINK team, Dr Emille Ishida and Dr Anais Möller who all supported my ideas and encouraged my efforts. You showed me what great research teams and communities can be like and inspired some of my best work.

Away from the professional world and a little closer to home, I want to thank my mum for her relentless support, who over the years always supported my dreams and helped with proofreading and countless applications. I want to thank all my friends, especially Kamaldeep Dhaliwal, Tejpal Jutla and Rami Khleif who put up with me when I was constantly deliberating on my career choices, you three were the voices of reason when I needed it, and the encouragement when I most doubted myself. Thank you Caragh Skipper for lending me your ear to talk about my research, in-between putting "jobs in the oven". I want to send a huge thank you to Donovan Newson for always being there to share my physics enthusiasm, and ensuring my understanding is sound. After more than a decade of saying we will get our doctorates one day, we are (almost) there.

Finally, my biggest thanks goes to my wife Stephanie, my best

friend, my cheerleader. Thank you for your patience, I know it's been a long 5 years. Thank you for your willingness to listen to my ideas, even when it meant I was drawing diagrams in the middle of the night. Your belief in me has been second to none, your relentless support gave me the strength to keep going on when times were hard, and God knows there were many of those. I could not have completed this journey without you by my side, and as such, I will forever be indebted to you for all your support and love along the way. *Thank you.*

Publications

The following articles are included as a whole or in parts in this thesis:

- **Allam, Jr., Tarek** and McEwen, Jason D (2021). *Paying Attention to Astronomical Transients: Photometric Classification with the Time-Series Transformer*. In: *arXiv preprint arXiv:2105.06178*. arXiv: 2105.06178 [astro-ph.IM]

- **Allam Jr, Tarek**, Peloton, Julien, and McEwen, Jason D (2023). *The Tiny Time-series Transformer: Low-latency High-throughput Classification of Astronomical Transients using Deep Model Compression*. In: *arXiv preprint arXiv:2303.08951*. arXiv: 2303.08951 [astro-ph.IM]

Furthermore, the following works were auxiliary PhD research but are not covered in this thesis:

- Alves, Catarina S, Peiris, Hiranya V, Lochner, Michelle, McEwen, Jason D, **Allam, Tarek**, Biswas, Rahul, et al. (2022b). *Considerations for Optimizing the Photometric Classification of Supernovae from the Rubin Observatory*. In: *The Astrophysical Journal Supplement Series* 258.2, p. 23

- Möller, Anais, Peloton, Julien, Ishida, Emille EO, Arnault, Chris, Bachelet, Etienne, Blaineau, Tristan, et al. (2021). *Fink, a new generation of broker for the LSST community*. In: *Monthly Notices of the Royal Astronomical Society* 501.3, pp. 3272–3288

- Hložek, R, Ponder, KA, Malz, AI, Dai, M, Narayan, G, Ishida, EEO, et al. (2020a). *Results of the Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC)*. In: *arXiv preprint arXiv:2012.12392*. arXiv: 2012 . 12392 [astro-ph.IM]

- Malz, AI, Hložek, Renée, **Allam, T**, Bahmanyar, Anita, Biswas, Rahul, Dai, Mi, et al. (2019a). *The Photometric LSST Astronomical Time-series Classification Challenge PLAsTiCC: Selection of a Performance Metric for Classification Probabilities Balancing Diverse Science Goals*. In: *The Astronomical Journal* 158.5, p. 171

- **T. Allam Jr**, Biswas, R., Hlǒzek, R., Lochner, M., J. D. McEwen, Peiris, H. V., et al. (2019). *Optimising the LSST observing strategy for Supernova light curve classification with machine learning*. In: Biomedical and Astronomical Signal Processing Frontiers (BASP)

- The PLAsTiCC team, **Allam Jr, Tarek**, Bahmanyar, Anita, Biswas, Rahul, Dai, Mi, Galbany, Lluís, et al. (2018a). *The photometric LSST astronomical time-series classification challenge (PLAsTiCC): Data set*. In: *arXiv preprint arXiv:1810.00001*

- Lochner, M., Scolnic, D. M., Awan, H., Regnault, N., Gris, P., Mandelbaum, R., et al. (2018a). *Optimizing the LSST Observing Strategy for Dark Energy Science: DESC Recommendations for the Wide-Fast-Deep Survey*. In: *arXiv*. eprint: arXiv:1812.00515

- Scolnic, D. M., Lochner, M., Gris, P., Regnault, N., Hlǒzek, R., Aldering, G., et al. (2018a). *Optimizing the LSST Observing Strategy for Dark Energy Science: DESC Recommendations for the Deep Drilling Fields and other Special Programs*. In: *arXiv*. eprint: arXiv:1812.00516

Abstract

A new golden age in astronomy is upon us, dominated by data. Large astronomical surveys are broadcasting unprecedented rates of information, demanding machine learning as a critical component in modern scientific pipelines to handle the deluge of data. The upcoming Legacy Survey of Space and Time (LSST) of the Vera C. Rubin Observatory will raise the big-data bar for time-domain astronomy, with an expected 10 million alerts per-night, and generating many petabytes of data over the lifetime of the survey. Fast and efficient classification algorithms that can operate in real-time, yet robustly and accurately, are needed for time-critical events where additional resources can be sought for follow-up analyses. In order to handle such data, state-of-the-art deep learning architectures coupled with tools that leverage modern hardware accelerators are essential.

The work contained in this thesis seeks to address the big-data challenges of LSST by proposing novel efficient deep learning architectures for multivariate time-series classification that can provide state-of-the-art classification of astronomical transients at a fraction of the computational costs of other deep learning approaches. This thesis introduces the depthwise-separable convolution and the notion of convolutional embeddings to the task of time-series classification for gains in classification performance that are achieved with far fewer model parameters than similar methods. It also introduces the attention mechanism to time-series classification that improves performance even further still, with significant improvement in computational efficiency, as well as further reduction in model size. Finally, this thesis pioneers the use of modern model compression techniques to the field of photometric classification for efficient deep learning deployment. These insights informed the final architecture which was deployed in a live production machine learning system, demonstrating the capability to operate efficiently and robustly in real-time, at LSST scale and beyond, ready for the new era of data intensive astronomy.

Impact Statement

This thesis brings together various schools of deep learning to create fast and efficient architectures. While specifically designed for photometric classification of astronomical transients, these architectures are also applicable to multivariate time-series classification more generally. By innovatively drawing on methods previously deployed outside of physics, this research introduces several novel concepts to the field, thereby providing researchers with additional tools to develop efficient classifiers, ready for the new era of data intensive science.

Notably, by leveraging depthwise-separable convolutions from the field of computer vision and applying them to the task of time-series classification for the first time, computational efficiency is improved and parameter count significantly reduced when compared with similar methods. Furthermore, our innovative use of multi-head self-attention for transient classification brought into play a new methodology for efficient analysis of light curves, thereby opening up the potential of interpretability of classifiers that adopt this technique. Deployment of this mechanism was made viable through our convolutional embedding, which is a new way of projecting multivariate time-series into a vector-space representation. As the classic `word2vec` algorithm has become part-and-parcel of natural language processing (NLP), convolutional embedding is expected to feature in many new deep learning architectures applied to photometric classification, as well as general time-series classification more widely.

Primarily, we introduced the novel time-series transformer architecture which combines multihead self-attention with the convolutional embedding, and showcased the use of model compression methods for the first time in photometric classifiers. The use of our time-series transformer has already shown to be broadly useful to other domains, with recent real-world application to biomedical classification using radio signals (Brausch et al., 2022). The versatility to handle general multivariate time-series classification tasks

makes the use of our methods suitable to handle a wide scope of problems, in a variety of disciplines, ranging from medical diagnostics, to activity recognition, silicon wafer quality checks and beyond.

The deployment of our novel time-series transformer into the live production brokering system of FINK is the first real-world application of this kind within physics. It is currently running real-time classifications, on real streaming data. The deployment of our model far exceeds the FINK criteria for processing alerts in real-time at nightly terabyte scale, made possible by employing state-of-the-art compression methods. This allows low-latency and high-throughput processing, translating into time saved on the cluster, freeing computational resources for other science modules that require it.

By developing computationally efficient architectures abstracted from the hardware they sit on, we have created methods allowing other future users to benefit from the freedom to deploy these methods in a multitude of ways. This is in line with the major advances taking place at the moment in the computer architecture and distributed systems world. Additionally, this means that as more energy efficient and powerful hardware accelerators become available, paired with better networking capabilities, our models' runtime performances are set to continue improving.

Our pioneering use of the methods described above to the task of photometric classification, demonstrate significant time and resources saving capabilities, and thereby monetary savings which would otherwise be unnecessarily wasted. While this has appeal to a general audience interested in cost reduction and indirectly lowering environmental impacts through energy savings, the use of our methods will generate particular benefits for those interested in deploying deep learning models in-the-wild in resource constrained settings. The low code and memory footprint of the compressed time-series transformer suits real-time on-device inference environments where model size and ultra-low latency are key. Our marrying of efficient computer vision and natural language processing

research with dark energy science will have a lasting impact, by enabling those in the transient science community to build computationally and energy efficient systems that bring us closer to answering some of the most fundamental questions in the Universe.

Tarek Allam Jr.

Attending to the Stars: *Efficient Deep Learning for Real-time Classification of Astronomical Transients*

There is stardust in your veins. We are literally, ultimately children of the stars.

— Jocelyn Bell Burnell

Contents

List of Tables

List of Figures

Part I

Background

But the problem, you see, when you ask why something happens, how does a person answer why something happens?

— Richard P. Feynman

1

Motivation and the Road to First Light

"*These [holding toy cows] are small... but the ones out there [pointing to real cows outside] are far away. Small... far away...*"

— **Father Ted.**

For thousands of years, mankind have looked to the heavens and questioned their fundamental being and existence – *where did it all begin, and how will it all end?* In more recent times, the field of physical cosmology, a branch of astronomy that observes celestial objects to determine the chronology of the Universe, from the Big Bang to today and into the future, is continually shedding light on this story.

In the early 20th century, it was postulated by Alexander Friedmann (1922) that solutions to Albert Einstein's field equations show a dynamical Universe, and also independently by Georges Lemaître (1927) that an expanding universe was possible. Einstein knew of this possibility, but dismissed it as the consensus of the time was that the Universe was static. To achieve this, he introduced a *cosmological constant*, that counters the inevitable implosion if no such term exists in a matter only Universe. The dynamical nature of the Universe was confirmed observationally by Edwin Hubble (1929) with the famous linear relation of recession velocity of galaxies and their distance away from us. In 1998, it was discovered by two independent teams, that the universe is not only expanding but also accelerating (Perlmutter et al., 1999; Riess et al., 1998). The driving force of this acceleration has since been dubbed *dark energy*. As yet, there is no definitive theory that completely describes the physics behind dark energy, but it is hoped

that through improved cosmological observations, one can at least constrain models of the Universe that try.

This chapter provides a brief review of the underlying physics that motivates the search for dark energy, and the methods currently being used to probe its make-up. Section 1.1 introduces some of the key equations and concepts that govern the field of cosmology, where it will be shown how Einstein's efforts for mathematically describing a completely static Universe became his "biggest blunder". Then, in Section 1.2 an overview is given of the methods by which dark energy is probed, with focus on the use of Supernovae Type-Ia (SNIa). Finally, Section 1.3 discusses a particular upcoming observational experiment that hopes to investigate dark energy even further, among other science goals, in order to help us better understand our Universe.

1.1 The Expanding Universe

Modern cosmology is built upon a central assumption that the Universe is *homogeneous* and *isotropic*, meaning there is no special direction nor place in the Universe. This idea is known as the *cosmological principle*, and there exists strong observational evidence that satisfies these assumptions on large spatial scales with precise measurements of the Comsic Microwave Background (CMB) radiation (*e.g.* Akrami et al., 2020).

This section highlights how our Universe is mathematically described, and how solutions to Einstein's field equations, that relate matter, energy and curvature, not only dictate the eventual fate of the Universe, but gives insight into the history up to the present day.

1.1.1 *Spacetime* Geometry

In 3-dimensional space, the physical distance between two points separated by coordinate distances $\mathrm{d}x$, $\mathrm{d}y$ and $\mathrm{d}z$ is given by,

$$\mathrm{d}\ell^2 = \mathrm{d}x^2 + \mathrm{d}y^2 + \mathrm{d}z^2. \tag{1.1}$$

The physical distance, ℓ, is invariant under choice of coordinate system. However, as a consequence of special relativity and Lorentz contraction, the measured distance between two points differs when an observer is moving. The separation of two objects in homogeneous and isotropic space is described by the *spacetime* interval, $\mathrm{d}s$, that accounts for the distance in space and time in co-moving coordinates,

$$\mathrm{d}s^2 = c^2\,\mathrm{d}t^2 - a^2(t)\,\mathrm{d}\ell^2, \tag{1.2}$$

where t refers to time, c refers to the speed of light which we shall adopt natural units going forward by setting $c = 1$, and $a(t)$ denotes the mathematical quantity that describes the changing separation of two points as the Universe expands as a function of time, known as the scale factor.

In order to define a complete *metric*, which relates coordinate distances to physical distances, one must also account for the intrinsic curvature of three-dimensional space. By adding a further term for curvature, k, and transforming from Cartesian coordinates to Polar coordinates, one can substitute Equation 1.1, into Equation 1.2, to obtain the Robertson-Walker (RW) metric,

$$\mathrm{d}s^2 = \mathrm{d}t^2 - a^2(t)\left[\frac{\mathrm{d}r^2}{1-kr^2} + r^2\left(\mathrm{d}\theta^2 + \sin^2\theta\;\mathrm{d}\phi^2\right)\right], \tag{1.3}$$

where $k = 0$ corresponds to a spatially flat universe with Euclidean geometry ($\mathbb{E}^3$), $k > 0$ relates to spherical geometry ($\mathbb{S}^3$) and $k < 0$ for hyperbolic geometry ($\mathbb{H}^3$). The RW metric thus describes a homogeneous and isotropic Universe that can grow (or shrink) in relation to the scale factor.

For simplicity, one can define $\mathrm{d}\chi \equiv \mathrm{d}r/\sqrt{1-kr^2}$, and the identity $\mathrm{d}\Omega \equiv \mathrm{d}\theta^2 + \sin^2\theta\,\mathrm{d}\phi^2$ such that Equation 1.3 becomes,

$$\mathrm{d}s^2 = \mathrm{d}t^2 - a^2(t)\left[\mathrm{d}\chi^2 + S_k^2(\chi)\,\mathrm{d}\Omega^2\right], \tag{1.4}$$

with

$$S_k(\chi) = \begin{cases} \sin(\chi), & k > 0 \Rightarrow \mathbb{S}^3 \\ r, & k = 0 \Rightarrow \mathbb{E}^3 \\ \sinh(\chi), & k < 0 \Rightarrow \mathbb{H}^3 \end{cases} \tag{1.5}$$

where r, θ, and ϕ are the spherical polar co-moving coordinates, under the scale factor.

If the coordinates are rescaled with $a \to \lambda a$ or $r \to r/\lambda$, the metric remains unchanged. As such, we can use this to set the scale factor evaluated today t_0 to be $a_0 = a(t_0) = 1$. When dealing with co-moving objects under the scale factor, all relative distances remain the same, but the actual distances may have increased. Furthermore, by considering the distance along a spatial geodesic the angle (θ, ϕ) is constant, then Equation 1.4 reduces to $ds = a(t)r$. The physical distance between two points with one designated as the origin is then found by integrating over the radial co-moving coordinate r,

$$d_{\mathrm{phys}} = a(t)\int_0^r \frac{1}{\sqrt{1-kr'^2}}\,\mathrm{d}r = a(t)r. \tag{1.6}$$

Hence, if one considers a galaxy with an initial radius at time t_0, up to a time t_1 this will change from $R = r \to R = a(t_1)r$, *i.e.* as the distance between galaxies increases, the radius of curvature of the Universe $R(t) = a(t)R_0$ increases at the same rate (Ryden, 2017). Using this analogy, if we consider the same galaxy moving in co-moving coordinates, let its position over time be $\boldsymbol{x}(t)$, then by applying Equation 1.6, the physical coordinates of this trajectory becomes,

$$\boldsymbol{v}_{\mathrm{phys}}(t) = \frac{\mathrm{d}\boldsymbol{x}_{\mathrm{phys}}}{\mathrm{d}t} = \frac{\mathrm{d}a}{\mathrm{d}t}\boldsymbol{x} + a\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = H\boldsymbol{x}_{\mathrm{phys}} + \boldsymbol{v}_{\mathrm{pec}} \tag{1.7}$$

where $\boldsymbol{x}_{\text{phys}}$ is the physical position, $\boldsymbol{v}_{\text{pec}}$ is the peculiar velocity and $H(t)$ is the Hubble parameter that is directly related to the expansion of the Universe,

$$H(t) = \frac{\dot{a}}{a}. \tag{1.8}$$

The time derivative of the scale factor $\mathrm{d}a/\,\mathrm{d}t$ can be written as $\dot{a}$ where $\dot{a} > 0$ signifies an increasing distance and $\dot{a} < 0$ describes a contracting distance. The additional term, $\boldsymbol{v}_{\text{pec}}$, is the peculiar velocity, which describes the velocity of the galaxy relative to the Galactic rest frame. By observing a large sample of objects, at distances of $z > 0.01$, peculiar velocities, $\boldsymbol{v}_{\text{pec}}$, can be ignored[1]. Then, by evaluating the Hubble parameter relative to today, $H_0 \approx$ 70km/s/Mpc, a linear relation between velocity and distance can be established,

$$\boldsymbol{v}_{\text{phys}} = H_0 \boldsymbol{x}_{\text{phys}}. \tag{1.9}$$

Equation 1.9 is known as Hubble's law (Hubble, 1929). The velocity can be determined from shifts in spectral lines via the Doppler shift (assuming non-relativistic objects) from peculiar motion, rotation of galaxies and the *cosmological redshift* of expanding space. At large enough distances cosmological redshift dominates over peculiar velocities[2] allowing the cosmological redshift to be defined as,

$$z \equiv \frac{\lambda_{\text{obs}} - \lambda_{\text{em}}}{\lambda_{\text{em}}} \approx \frac{v}{c}. \tag{1.10}$$

where λ_{obs} refers to the wavelength of light that is observed and λ_{em} relating to the wavelength of light that was emitted from a

[1] Due to the random direction of the peculiar velocity, over a large sample this is averaged out to zero.

[2] At distances of $z < 0.01$, the Doppler shift due to peculiar motions of the galaxies relative to one another causes significant scatter when calculating Hubble's law.

light source. The cosmological redshift relation above arises from the intuitive notion that as space expands, the wavelength of photons will also be stretched. Subtly different from Doppler shift, cosmological redshift causes a redshift in light spectra if space is expanding, and blueshift if space is contracting. Let the wavelength of light observed today be denoted as a_0, and wavelength of light at some time in the past as a_1, then with our definition for a_0 as unity, Equation 1.10 can be reformulated as,

$$z = \frac{1 - a(t_1)}{a(t_1)} \implies 1 + z = \frac{1}{a(t_1)}. \tag{1.11}$$

The beauty here is that we have a quantity that is measurable. Using spectral absorption lines of different elements as a molecular fingerprint, we can look to the heavens and use our understanding of how these spectral lines would be affected under motion and expanding space. This ultimately gives us a strong indication of how the stars are behaving, and more fundamentally, if they are moving away, or towards us.

The question now moves towards how can we accurately determine distance, the other variable in Hubble's law. This is by no means a trivial task, and one must be careful how distance is defined and thought about[3].

1.1.2 Notions of Distance

With the measurable quantity of redshift, and therefore velocity, to hand, we would be able to reveal the nature of the scale factor, and hence the history of the Universe, if we can also accurately measure distances in the cosmos.

Recall in Equation 1.6 we defined *proper* distance, d_{phys}. The proper distance, although includes the scale factor, is not a quantity that we can directly measure.

Instead, we must look towards a more practical definition of

[3]As Father Dougal discovered when observing cows in a field, and cows on a table, were the same "size" (Father Ted - Series 2, Episode 1).

distance that takes the expanding Universe into consideration and uses light which we can *see*, allowing for the fact it takes a finite amount of time to reach us. Ignoring parallax for nearby distances[4], the use of *luminosity distance* is an indirect way of using our knowledge of how bright a celestial object should be and then comparing it to the apparent brightness that we measure.

Given the *intrinsic luminosity* L of an object, we would like to determine the apparent luminosity, or flux, F, given by the energy per unit time per unit area, seen by a distant observer, to infer the luminosity distance. Assuming isotropic emission, in a static Euclidean space, the observed flux at a distance d, the energy spread out over a sphere $\mathbb{S}^2$ of area $4\pi d^2$ would give,

$$F = \frac{L}{4\pi d^2}. \tag{1.12}$$

When working in an expanding spacetime the denominator of Equation 1.12 is replaced by $4\pi a^2(t_0)S_k(\chi)^2$, where t_0 is the time the radiation has been observed.

The rate of the photons that arrives from the source is reduced by a factor related to that shown in Equation 1.11, *i.e.* by $1/(1+z)$, and so reduces the observed flux by the same amount. Furthermore, with cosmological redshift, the energy of photons is also inversely proportional to the scale factor, and thus another $1/(1+z)$ is included. As such, in an expanding universe, the observed flux from a source with intrinsic luminosity L at a coordinate distance χ, and redshift z, is given by

$$F = \frac{L}{4\pi S_k(\chi)^2(1+z)^2}, \tag{1.13}$$

which if we refer back to Equation 1.12 where distance is given by d, we can yield an expression for *luminosity distance* $d_{L(\chi)}$ in terms

[4]Parallax, the change position in the sky when observing a source from two known points, is a highly precise procedure for measuring distances, but does not scale to cosmological scales.

of redshift z with,

$$d_L(\chi) = S_k(\chi)(1+z) \tag{1.14}$$

If L is known, we can then use these relations to measure luminosity distance d_L (Abbott et al., 2019; Tong, 2019).

To be able to use this methodology, cosmologists can then only use certain objects, known as *standard candles*[5], where the underlying physics of the light source is well understood *i.e.* intrinsic luminosity, and of course, that it is bright enough to see at large distances.

Cepheid Variable Stars

Cepheids are a type of star that pulsate with a relatively short period, with the length of the periodicity of the pulses correlated to the intrinsic brightness of the star. In the early 20th century Hubble was able to use these objects to construct a diagram that put forward evidence for an expanding Universe (Leavitt and Pickering, 1912) (see Figure 1.1). Although Cepheids allow for reasonably precise distance measures, their luminosity is simply not strong enough to probe distances beyond 50Mpc (Las Cumbres Observatory, 2022).

Supernovae Type-Ia

Supernovae, the explosive death of a star, are so bright that they can even outshine the light of the galaxy they reside in, and have been observed with the naked eye, with several particular events noted in human history. In 1572, Tycho Brahe observed one such "new star" in the sky in the constellation Cassiopeia (see Figure 1.2). The Dutch astronomer undertook a detailed analysis of this event, posthumously dubbed *Tycho's Supernova*, and tried to

[5]Perhaps the correct terminology is *standardisable* candles, since for the case of SNIa, it was not until work by Phillips, 1993, that these objects were actually useable as standard candles.

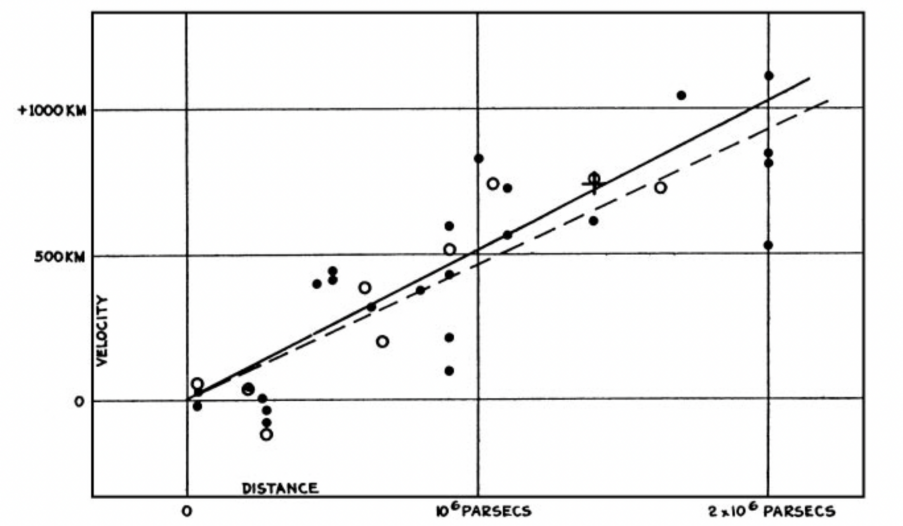


FIGURE 1.1: Original Hubble diagram that expresses the velocity–distance relation among extra-galactic nebulae. The velocity-distance measurements were extremely limited due to observing only nearby galaxies, and thus showed considerable scatter due to the significance of peculiar velocity, which led to an incorrect estimation for the age of the Universe. Reproduced in full from Hubble (1929)

convince[6] the other astronomers of the day that through his scientific analysis it must have originated far beyond the Earth, in contrast to the Aristotelian doctrine that the heavens never change.

Many years later this event was determined to be a SNIa, which is a particular type of supernova that happens when a white dwarf accretes mass from a companion star. When this goes beyond the Chandrasekhar limit of 1.4 $M_\odot$, the point at which electron degen-

[6]Fighting the millennia long Aristotelian mindset, in the preface of Tycho Brahe's book, *De Stella Nova in Pede Serpentarii*, he called upon those who doubted him: "Oh thick wits. Oh blind watchers of the sky", to look at his scientific analysis and see that the supernova could certainly not have occurred within the terrestrial sphere.

eracy pressure can no longer support the star against gravitational forces, the star implodes (Chandrasekhar, 1931). Since the Chandrasekhar limit is well understood, these are prime candidates for standard candles.

Kilonovae

It is hoped that in the future, *standard sirens* would also be able to be used for measuring distances in the Universe in addition to standard candles. Following from the recent discovery of gravitational waves (GW) (Abbott et al., 2017), and the associated electromagnetic counterpart, we can measure the GW amplitude to obtain a measurement for luminosity distance (Holz et al., 2018). Furthermore, we can use the redshift from the electromagnetic counterpart to independently verify estimates for the Hubble constant (Baumann, 2022).

1.1.3 *Spacetime* Dynamics

So far we have seen that we can describe an isotropic and homogeneous universe with the RW metric. It gives an avenue for us to explore the dynamical nature of the Universe by way of the time dependent scale factor.

To begin, we start with the idea of modelling the Universe as a *perfect fluid* through the cosmological principle. With an expanding fluid, one would expect the energy density to become diluted in relation to conservation of energy laws, and the first law of thermodynamics, which states that $\mathrm{d}E + p\,\mathrm{d}V = T\,\mathrm{d}S$, where $\mathrm{d}E$ is the change in energy, $\mathrm{d}V$ the change in volume, T the temperature, $\mathrm{d}S$ the change in entropy, and p the pressure. Considering $\mathrm{d}S = 0$ for an adiabatic process, and taking the time derivative, we have

$$\frac{\mathrm{d}E}{\mathrm{d}t} = -p\frac{\mathrm{d}V}{\mathrm{d}t}. \tag{1.15}$$

Then, if we consider a region of fluid in a co-moving volume V_0,

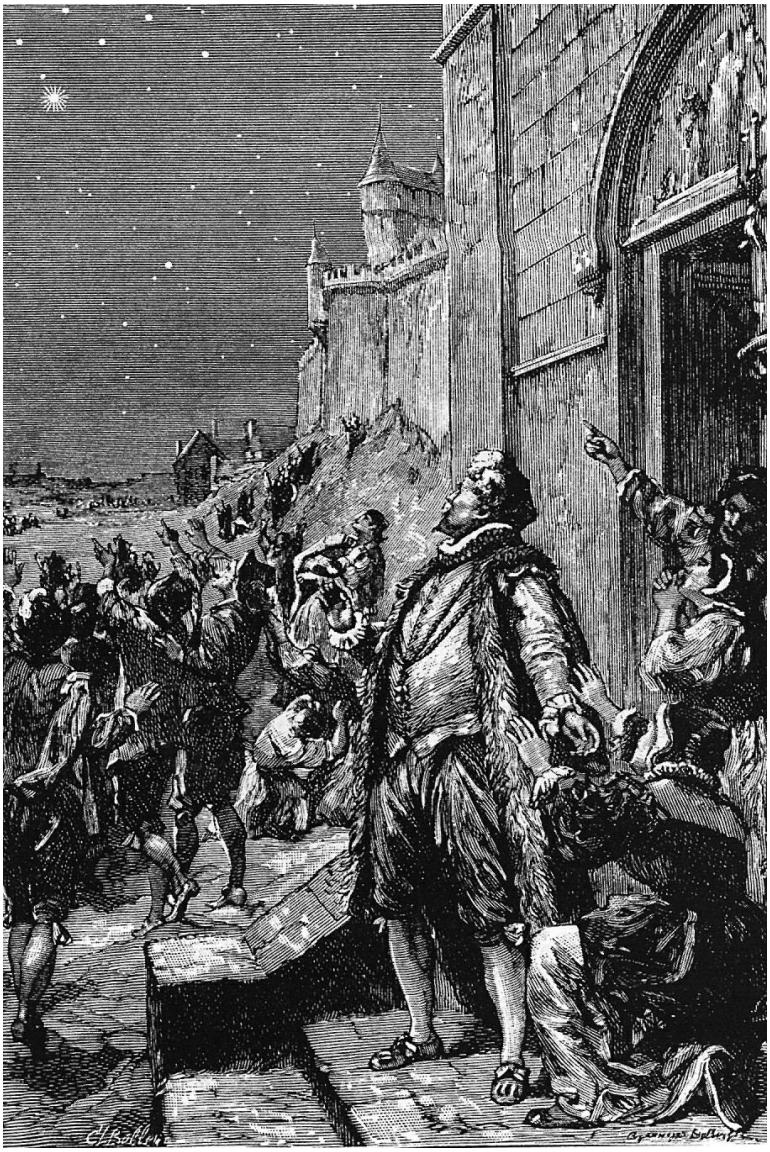


FIGURE 1.2: Tycho Brahe and the Supernova of 1572 (Flammarion, 1894)

the physical volume is $V(t) = a^3(t)V_0$ and hence,

$$\frac{\mathrm{d}V}{\mathrm{d}t} = 3a^2\dot{a}V_0. \tag{1.16}$$

Next, by noting the energy in such a volume being given by $E = \rho a^3 V_0$ we obtain,

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \dot{\rho}a^3V_0 + 3\rho a^2\dot{a}V_0. \tag{1.17}$$

Combining Equation 1.16 and Equation 1.17 into Equation 1.15 we arrive at an expression for energy conservation in a cosmological setting (Tong, 2019) known as the *continuity equation*,

$$\dot{\rho} = -3H(\rho + p). \tag{1.18}$$

By defining an *equation of state* as $p = w\rho$, which is to say that pressure in the system is proportional the density of the fluid in the system. It can be integrated in order to establish how the energy density would depend on the scale factor as a function of time. In doing so, we obtain,

$$\frac{\dot{\rho}}{\rho} = -3(1+w)\frac{\dot{a}}{a} \Rightarrow \log\left(\rho/\rho_0\right) = -3(1+w)\log a \tag{1.19}$$

$$\Rightarrow \rho(t) = \rho_0 a^{-3(1+w)} \tag{1.20}$$

with $\rho_0 = \rho(t_0)$ and using the definition of $a(t_0) = 1$ for today. If it is the case that there is a set of fluids in the system described by different equations of state, we must be sure to include all of these together such that $\rho_{\mathrm{tot}} = \sum_w \rho_w$. We also update the notation of Equation 1.20 to be more strictly defined as,

$$\rho_w(t) = \rho_{w,0}a^{-3(1+w)}, \tag{1.21}$$

where $\rho_{w,0} = \rho_w(t_0)$.

With an equation of state and the continuity equation, we are missing just one more piece of the puzzle to describe the Universe

dynamically. In order to determine the time evolution of the scale factor, $a(t)$, one must combine the RW metric, defined in 1.4 together with Einstein's field equations. The result yields one of the most significant equations of modern cosmology, the *Friedmann equation*,

$$H^2 = \left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3}\rho_{\text{tot}} - \frac{k}{a^2}, \tag{1.22}$$

where the curvature given by k and total density of the Universe by ρ_{tot}.

The Friedmann equation can be understood as defining how expansion of the Universe is driven by the components within it. Measuring the parameters of the Friedmann equation is a principle task in cosmology as this reveals the composition of our Universe as well as giving insight into its fate.

If we consider a flat Universe with $k = 0$, the Friedmann equation simplifies to

$$H^2 = \frac{8\pi G}{3}\rho_{\text{tot}}, \tag{1.23}$$

which when differentiated with respect to (w.r.t) time gives the *acceleration equation*,

$$\frac{\ddot{a}}{a} = -\frac{4\pi G}{3}\left(\rho_{\text{tot}} + 3p\right). \tag{1.24}$$

The acceleration equation implies that a Universe containing only matter will decelerate due to gravitational effects. Einstein first introduced the cosmological constant, Λ, in an attempt to counter this with a repulsive force and keep to a static cosmology; an act that would come to be known as his *biggest blunder* (Cohen-Tannoudji, 2018). It was shown that while achieving a static Universe, it was in unstable equilibrium. Any slight increase or decrease of the Universe through expansion or contraction would cause a

runaway effect (Ryden, 2017). A decade later, Edwin Hubble showcased empirical evidence for an expanding Universe, when he constructed his eponymous diagram, shown in Figure 1.1 (Hubble, 1929). Hubble built upon work and earlier measurements by Vesto Slipher (1917), to confirm Hubble's law, described in Equation 1.9. However, due to the inherent challenges of only being able to measure nearby galaxies, peculiar velocity effects led to an incorrect estimation of the linear relationship which was an order of magnitude off what could reasonably explain the age of the Universe[7]. Consequently, the cosmological constant was not dismissed, but actually leveraged by cosmologists to help explain this discrepancy. If Λ is sufficiently large in the acceleration equation to make $\ddot{a} > 0$, then $\dot{a}$ would be smaller in the past compared to today and hence the Universe would be older than what Hubble measured as H_0.

If one keeps the cosmological constant, the question remains, what physically is this repulsive entity? Georges Lamaître was one of the first to realise that the cosmological constant should be identified as *vacuum energy* (Luminet, 2015), or zero point energy to borrow a term from quantum field theory, with an associated energy density defined as,

$$\rho_{de} = \frac{\Lambda}{8\pi G}. \tag{1.25}$$

In this manner, the vacuum energy, or as it is called today — *dark energy*, is subsumed into the description of the total energy density $\rho_{\rm tot}$, along with other components that make up the cosmological fluid including matter, $\rho_{\rm m}$, and radiation, $\rho_{\rm r}$. Setting $k = 0$ for a flat Universe, the total energy density, ρ_{tot}, must sum to equal the Hubble constant squared together with the other factors

[7]Hubble's measurements suggested a best-line fit to the velocity-distance relation to be $\approx$ 500km/s/Mpc (Hubble, 1929) which would give an estimated age of the Universe to be around 2 billion years. This went against evidence of the day with geological Earth data and observations from Globular Clusters that suggested the Universe should be far older.

in Equation 1.23. Therefore, we can define a *critical density* as,

$$\rho_{\text{crit}} \equiv 3H^2/(8\pi G), \tag{1.26}$$

which describes the energy density required for the Universe to halt its expansion (in a zero curvature setting). This is helpful to allow us to define a dimensionless density parameter for each fluid component with,

$$\Omega_w = \frac{\rho_{w,0}}{\rho_{crit,0}}, \tag{1.27}$$

where the sum of all parameters is given by,

$$\sum_{i=m,r,de} \Omega_i = 1 + \frac{kc^2}{R^2 H_0^2}. \tag{1.28}$$

Thus, for a flat Universe, $\sum_w \Omega_w = 1$, whereas for a positively curved Universe with $k = +1$, $\sum_w \Omega_w > 1$. If it is the case that we live in a negatively curved space with $k = -1$, then $\sum_i \Omega_i < 1$.

Although not physically meaningful, it can be helpful to regard curvature in a similar way to other energy densities, and define it as,

$$\rho_k = -\frac{3kc^4}{8\pi G R^2 a^2} \implies \Omega_k = \frac{\rho_{k,0}}{\rho_{\text{crit},0}} = -\frac{kc^2}{R^2 H_0^2}. \tag{1.29}$$

Using the definitions from Equation 1.21, together with the Friedmann equation (1.22), yields an expression that describes the relationship between the energy density of the Universe with the scale factor.

$$\left(\frac{H}{H_0}\right)^2 = \frac{\Omega_r}{a^4} + \frac{\Omega_m}{a^3} + \frac{\Omega_k}{a^2} + \Omega_{de} \tag{1.30}$$

where normal matter is expectantly diluted by $1/a^3$ as the volume expands, and radiation is affected by the cosmological expansion

further still, causing the wavelength of photons to be stretched. Equivalently, by using our definition from Equation 1.11 this can also be described as,

$$H^2 = H_0^2 \Big[\Omega_m(1+z)^3 + \Omega_r(1+z)^4 + \Omega_{de}(1+z)^{3(1+w)} + \Omega_k(1+z)^2\Big] \ , \tag{1.31}$$

Thus, by determining the parameters in the above equation, we are able to gauge the dynamics of our Universe, and map its evolution over time. This is the central mission of cosmology, and we shall see in the following section how teams of scientists search for signs in the night sky that help to reveal this.

1.2 In Search of Dark Energy

Theoretically we have shown that an extra component, which behaves like a cosmological constant is supported under Friedmann's equations. Moreover, having a dark energy component also allows for an acceleration in the expansion of the Universe to help solve the problem that a matter only Universe creates, known as the age problem, where a matter only Universe predicts an age that is much smaller than observations of Globular Clusters would indicate (Valcin et al., 2020). The inclusion of the cosmological constant, and thus an additional form of dark energy along with matter helps to resolve this discrepancy. The striking evidence for the existence of such an entity came following the construction of the Hubble diagram using SNIa when the Universe was found to not only be expanding but to be accelerating. Two teams, the Supernova Cosmology Project (Perlmutter et al., 1999) and the High-Z Supernova Search Team (Riess et al., 1998), used SNIa as standard candles to independently discover there to be an epoch of acceleration starting around 5 billion years ago.

It was noted earlier how the use of standard candles are used to determine distance in the Universe, and with SNIa playing a pivotal role for constraining properties of dark energy. A more detailed breakdown of how they are used to measure luminosity distance is given here. In Section 1.1.2, we covered the overarching equations relating to luminosity distance. In the case of Supernovae (SNe), we should be reminded that they are not in fact pure standard candles, but can be standardised by way of the Phillips relation (Phillips, 1993). In this way, one can infer the distance to SNe from measurements of their apparent brightness. Practically, astronomers quantify light from a source by the *apparent magnitude*, given by

$$m \equiv -2.5 \log_{10} \left(f / f_x \right) , \tag{1.32}$$

where f_x is a reference flux, and *absolute magnitude*, which is the apparent magnitude that it would have if it were at a luminosity distance of $d_L = 10$ parsecs. As such, a light source with intrinsic luminosity, L, has absolute magnitude as,

$$M \equiv -2.5 \log_{10} \left(L / L_x \right) , \tag{1.33}$$

where again, L_x is simply a reference luminosity, which relates to f_x viewed at a distance of 10 parsecs (Ryden, 2017). Combining the two equations above, as well as Equation 1.12, allows us to express it as

$$M = m - 5 \log_{10} \left(\frac{d_L}{10\text{pc}} \right) . \tag{1.34}$$

However, it is more common among astronomers to use *distance modulus* when dealing with luminosity and flux of a light source, given by

$$\mu = m - M = 5 \log_{10} \left((1+z) \int_0^z \mathrm{d}z \frac{H_0}{H(z)} \right) , \tag{1.35}$$

which can inform us of the expansion history by plotting the observed μ versus z (De Putter, 2010).

Using these equations, both the Supernova Cosmology Project (Perlmutter et al., 1999) and the High-Z Supernova Search Team (Riess et al., 1998), were able analyse around 50 SNe at redshift below 1, and construct a Hubble diagram with distant modulus on the y-axis and redshift on the x-axis. The fainter the object that is observed, the farther away it is and hence the further back in time, allowing us to treat y-axis as a time-axis. In addition, since photons are stretched proportionally to the expansion[8], by having redshift as our x-axis, the Hubble diagram consequently gives an indication to the stretching of the Universe as a function of time. Looking back in time, approximately 5 billion years ago, deviations in the expansion rate caused by the cosmological parameters can be seen. These teams found that for distant SNe the observed magnitude was higher than expected (fainter than expected) for a given redshift, under the assumption of a matter-dominated decelerating Universe. The data indicated an accelerating Universe with a dark energy equation of state of $w \approx -1$ to explain the observation.

Other experiments have proceeded to confirm the observations of the two Nobel Prize winning teams, with one such example being the Dark Energy Survey (DES) (Abbott et al., 2019), whose efforts to construct a similar Hubble diagram can be seen in Figure 1.3.

The subtle difference in expected brightness for events at $z \approx 0.5$ indicates an accelerating Universe. For SNIa with low redshift of the order 0.1, a linear relationship between distance and redshift (velocity) holds, as described in Equation 1.9. However, at greater distances, due to the scale factor changing over time, the way distance relates to redshift is modified is described in Equation 1.11, a deviation from linearity would be expected for an accelerating or decelerating Universe. Using Equation 1.11, we can see that a SNIa observed at $z = 0.5$, suggests the Universe was 2/3 the present size when the event occurred. For the scenario of an accelerating-expanding Universe, $\ddot{a}$ from Friedmann's equations is greater than zero, and hence $\dot{a}$ would be larger today compared to the past. It

[8]Which is cosmological redshift as defined in Equation 1.11

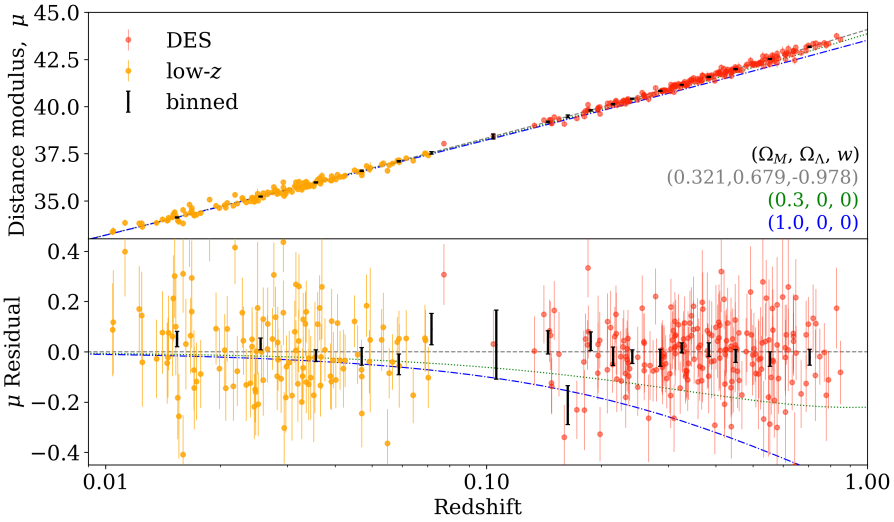


FIGURE 1.3: Hubble diagram depicting the linear relationship in log-space between distance and velocity (redshift), with best fit parameters for Equation 1.31. Reproduced in full from Abbott et al. (2019)

would give rise to larger distances and SNIa that are fainter than if it were the case for a constant $\dot{a}$ and a Hubble parameter that has the same value it has today. Corroborating with observations, and as shown in Figure 1.3, the deviation away from linearity tends towards an *accelerating* Universe.

While this evidence has since been supported by other experiments, such as DES mentioned above, and by way of other cosmological probes such as Baryon Acoustic Oscillations (BAO; Eisenstein et al., 2005) *etc.*, what is perhaps reassuring is confirmation of an epoch of deceleration in the matter-dominated era (Riess et

al., 2001, 2004). The current constraints on dark energy come from a multitude of experiments, with some of the most recent results showcased in Figure 1.4 and Figure 1.5.

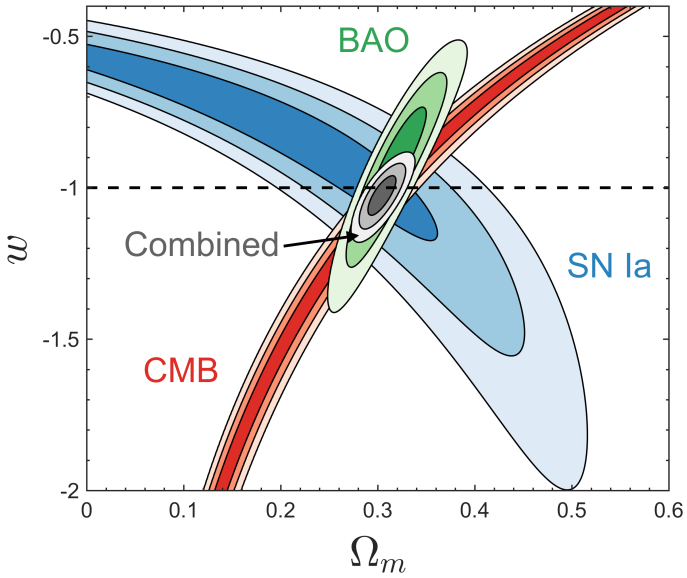


FIGURE 1.4: Assuming $k = 0$ flat Universe, the cosmological parameter constraints for Ω_m (JLA Betoule et al., 2014; blue), BAO (BOSS DR12 Alam et al., 2017; green), and CMB (*Planck* 2015 Ade et al., 2016; red), with credible region contours corresponding to 68.3%, 95.4%, and 99.7%. Reproduced in full from Huterer and Shafer (2017)

The goal now is to further constrain the cosmological parameters, and get tighter bounds for the equation of state, w and Ω_m, thereby refining our best-fit values for Ω_{de} and our understanding of dark energy. Note that w in particular can shed light on the form of dark energy. If through further investigation and improved precision measurements it is shown that previous estimates are wrong and the parameter w is not -1 then this will fundamentally change what we think we know about the Universe and force a rethink of the current model of cosmology. Hence, it is of utmost importance

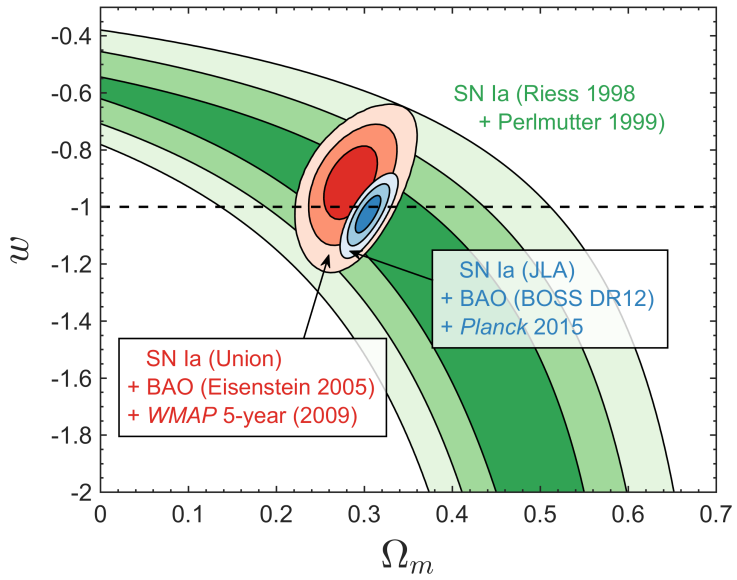


FIGURE 1.5: Historically improved measurements of cosmological constraints on Ω_m and the equation of state parameter w, assuming a flat Universe with $k = 0$. Reproduced in full from Huterer and Shafer (2017)

to continue with the endeavour for better measurements and to further constrain the estimates for w. The recent precision measurements from the Plank satellite (Aghanim et al., 2020) combined with joint constraint measurements with BAO, the Universe can be considered flat with $\Omega_k = 0.001 \pm 0.002$. Furthermore, since Ω_r has been directly measured to be very small, it is often considered to be negligible (Zyla et al., 2020) when determining $\sum_i \Omega_i$.

In terms of the future of SNe cosmology, the aim is to observe a far greater sample of SNe in general to reduce statistical errors, and to observe a large sample of SNIa at even higher redshift than before. Future planned surveys, such as the Legacy Survey of Space and Time (LSST) (Ivezić et al., 2019) at the Vera C. Rubin Observatory, will do just that, and it is the focus of this thesis to develop

tools and techniques that enable improved SNe cosmology for this survey.

1.3 The Legacy Survey of Space and Time

The Vera C. Rubin Observatory, currently under construction at El Peñón peak of Cerro Pachón in northern Chile, will carry out the upcoming Legacy Survey of Space and Time (LSST). Set to be the largest ground-based optical survey, it will focus on four key science drivers; accounting for objects in our solar system, mapping our galaxy, investigating transient events, and probing dark energy and dark matter (Ivezić et al., 2019). The telescope at the Vera C. Rubin Observatory uses a 3.2-gigapixel camera that will pan the sky in the southern hemisphere for 10 years, observing in 6 photometric filters, *u, g, r, i, z, y*, in the wavelength range of 320–1050 nm. Most of its operations will be devoted to the wide-fast-deep (WFD) mode which will observe approximately 18,000 deg^2 of the sky repeatedly. The other survey mode of deep-drilling-fields (DDF) will focus on special observations that look far back in time through long exposures and at higher *cadence*, *i.e.* greater sampling across each passband. The gains that LSST will bring to the transient science community are unprecedented and it is expected that the telescope will witness 10 million transient events per night, with the final database to contain around 32 trillion observations of 40 billion stars and galaxies (Ivezić et al., 2019). Of these observations, approximately 250,000 SNIa events *per-year* are to be expected, a considerably larger sample than all SNIa observed before by two orders of magnitude. The hope is to leverage this large number of SNIa to strengthen SNe cosmology, and improve the constraints on the dark energy equation of state, w, and Ω_m (and by that constrain Ω_{de} to better than 5% with just even 1/10th of the expected sample size (Abell et al., 2009)). Notwith-

standing, one needs to be able to distinguish between the millions of transient events that will be observed, where SNIa will only be a small fraction.

1.3.1 Transient Classification

Typically SNe, which are observed over a period of a few days to a few weeks, are classified by the presence of particular absorption lines in their spectra. Specifically, SNIa are distinguished by the absence of hydrogen lines and the presence of Si II λ 6150 absorption (Filippenko, 1997). However, spectroscopic classification of transient events is a costly process. By using broad photometric passbands, LSST will be able to "see" far more events than ever before, or that could be possible with spectroscopic equipment. The problem then arises: how can one accurately identify different transient photometrically using only passband information? In contrast to spectroscopic classification, photometric classification is far more challenging, and one is more susceptible to cross-contamination from other events such as core-collapse SNe (SNe Ib/c and SNe II) which share a similar profile to SNIa when observed photometrically (see B for examples). Consequently, studies have been done by similar photometric surveys to determine the acceptable level of cross-contamination from such events that would still allow for robust cosmological analysis of the dark energy equation of state. This range has been reported to be between 8% (DES; Vincenzi et al., 2021) and 5% (Pan-STARRS; Kaiser et al., 2002). It is expected LSST will require a high SNIa purity (described in more detail in Chapter 2) and cross-contamination rate to be at least within this range, if not lower. It should be noted that these levels are in the context of full phase light curves, and so one may expect a higher level of cross-contamination in the early phase of the events, where only partial information is available for identification.

The filter response for LSST can be seen in Figure 1.6 with wavelength given in units of angstroms, Å.

When making observations photometrically, the flux measure-

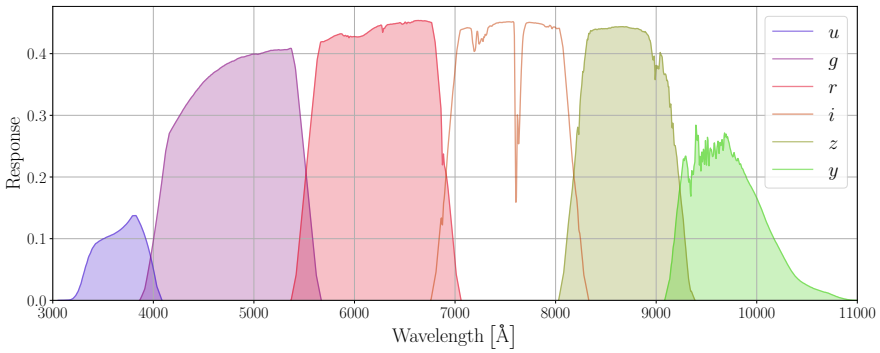


FIGURE 1.6: LSST filter response as a function of wavelength. Throughput values as of tag 12 from `github.com/lsst/throughputs` and produced using `speclite-v.015` (Robitaille et al., 2013)

ment corresponding to a given passband is obtained by collecting all light that is received through that particular filter. Multi-band photometry allows for more information to be retrieved to help determine properties of the light source, such as temperature, but this is of course not as rich as observing spectroscopically. However, if one collects multi-band photometry over a period of time, of the same source, a *light curve* can be constructed, which tells us more about what kind of a transitive event this may be. An example light curve of a SNe event can be seen in Figure 1.7, which shows flux measurements across four passbands of a single object, viewed over the course of roughly 100 days. There is significant variance in how long-lived a light curve may be depending on the light source. Some objects are intrinsically variable and so the light curve will peak and trough periodically over time. These may include Cepheid variables, RR-Lyrae variables and so on. Other non-periodic objects such as SNe will exbibit a single rise and fall of the light curve akin to that shown in Figure 1.7. In addition to SNe, these may also include objects such as Kilonovae, M-dwarf Flares whose light

curve is expected to be on the order of days, compared to other non-periodic objects of Pair Instability Supernovae (PISN) or Superluminous Supernovae (SLSN) which are on the order of 200 to 400 days respectively (De Cia et al., 2018; Gilmer et al., 2017).

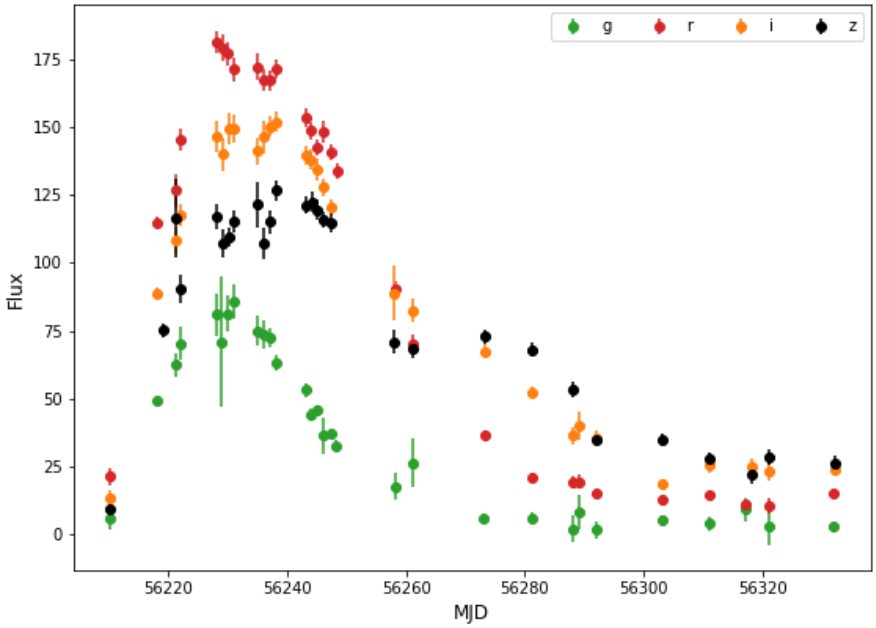


FIGURE 1.7: Example light curve of a transient event taken from the SPCC dataset (Kessler et al., 2010a) viewed across 4 passbands, *g, r, i, z* shown in different colours, over a period of $\sim$ 100 days, where time-axis is given in modified Julian date range. Heteroskedastic error bars are the result of varying weather and telescope conditions at the time of observation.

The image processing method of *difference imaging* measures differential photometry by matching the pointing and point-spread function(s) between image frames, typically for the detection time-varying celestial objects (Wang et al., 2017). A new image is compared with an aligned reference image, where the difference between

the two images is determined by calculating the difference between each pixel of each image, and forming a *difference image* from the result. An example of using difference imaging to detect a SNe can be seen in Figure 1.8. A detection occurs when the difference image is above a certain signal-to-noise threshold. When this threshold is reached, a transient event *alert* is triggered, with data streamed to brokers around the world for follow-up analysis.

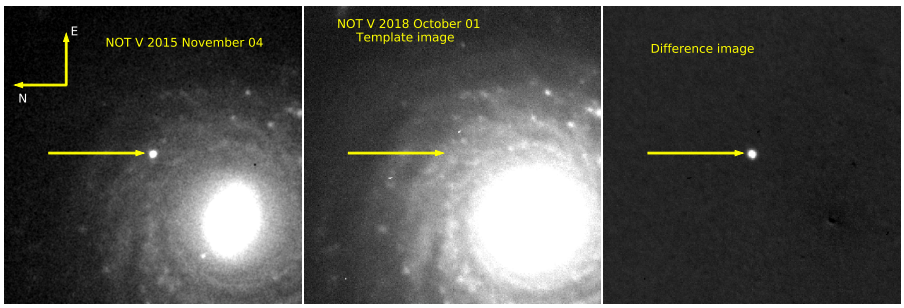


FIGURE 1.8: Difference imaging example. The image on the left shows a region of interest with a transient event occurring in a host galaxy. This was confirmed by comparing it to a reference image of the same patch of sky on a different date. The difference between the two is calculated by subtraction of one from the other to give the difference image shown on the far-right panel, revealing just the transient object. Reproduced in full from Singh et al. (2021).

1.3.2 Alert Brokers

It is expected that once LSST is fully operational, the 10 million transient alerts per night will produce around 1TB of raw data, and 3PB over the lifetime of the survey (Ivezić et al., 2019). With transient alerts being issued worldwide within 60 seconds, LSST will certainly usher in a new era of astronomy, where data-driven methods and machine learning will be critical to handle such a deluge of data. Alert brokering systems will be the first port-of-call for the real-time global distribution of alert stream data. A typical broker

will ingest the full alert stream and enrich the data through cross-matching with archival catalogues, identification and prioritisation of objects for follow-up analysis and spectroscopic observations, and photometric classification (Jurić et al., 2022).

With first light scheduled for the Spring of 2023[9], there are several broker groups already in place that are gearing up to handle the alert stream. Of the nine proposals for LSST brokering systems, seven teams were selected to receive the full alert stream: The Automatic Learning for the Rapid Classification of Events (ALerCE) (Förster et al., 2021), AMPEL (Nordin et al., 2019), Arizona-NOAO Temporal Analysis and Response to Events System (ANTARES) (Matheson et al., 2021), BABAMUL (Duev and Graham, 2022), FINK (Möller et al., 2021), Lasair (Smith, 2019) and Pitt-Google (Wood-Vasey et al., 2022).

In an effort to prepare, many teams are using alerts coming from current surveys, such as the Zwicky Transient Facility (ZTF) (Bellm, 2014), to fine-tune their data ingestion and classification pipelines. These brokers will need to operate in real-time and provide low-latency classification scores such that follow-up analysis can be triggered swiftly.

The work in this thesis focuses on the development of classification methods that are suitable for use with the photometric data that will be collected by LSST. Our transient classification estimates will help identify possible SNe candidates, as well as other transients of interest, to the scientific community. For greatest impact, we develop fast and efficient architectures that are worthy of deployment into alert brokers that will add value by way of classification scores to the alert stream in real-time. The ultimate goal is for this value added information to be harnessed for constraining cosmological parameters even further, and to hopefully shed more light onto the mysterious entity driving the acceleration of our Universe.

[9]Milestone updated as of April 5, 2022 (www.lsst.org/about/project-status)

2

A Brief History of Time-Series Classification

"I am rooting for the machines! I've always been on the machines' side"

— **Claude Shannon.**

Time-series data is ubiquitous across the sciences, and has seen a resurgence of research efforts over the last couple years as new machine learning methods are coming to dominate the landscape. A recent in-depth review of current state-of-the-art by Fawaz et al. (2019) highlighted the new trend in time-series classification which was to leverage deep learning for best results. However, previous methods are not without their merits, and it is important to review the traditional approaches to better understand their limitations yet determine how they could be extended.

This chapter gives scope to the underlying task at hand by first describing photometric classification of SNIa in terms of the general problem of multivariate time-series classification in Section 2.1. This is followed by a discussion of the evaluation metrics used for this task in Section 3.5. Then, in Section 2.3 previous methods are reviewed with discussions on their respective limitations. Further details are given in 2.3.1 on a particular method that was applied in earlier doctoral work for studies in telescope cadence optimisation. Next, in Section 2.4, a brief recap is presented of the recent success story of deep learning, especially as it has been applied to time-series classification.

The chapter closes in Section 2.5 with a summary of the doctoral

research that has been completed over the course of study, and then outlines the individual contributions that feature in this thesis in Section 2.5.2.

2.1 Photometric Classification: A Multivariate Time-Series Classification Problem

To better understand the problem, it will be useful to review the kind of data one is dealing with and to make some definitions (adapted from Fawaz et al., 2019) with regards to the task of astronomical transient classification. In general, the data that one observes can be viewed as an *irregular multivariate time-series* signal:

Definition 1 *A* univariate time-series *signal* $\mathbf{x} = [x_1, x_2, \ldots, x_T]$ *consists of an ordered set of* T *real values with* $\mathbf{x} \in \mathbb{R}^T$.

Definition 2 *An* M*-dimensional* multivariate time-series *signal,* $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M]$ *consists of* M *univariate time-series with* $\mathbf{X} \in \mathbb{R}^{T \times M}$.

Definition 3 *An* irregular time-series *is a ordered sequence of observation time and value pairs* (t_n, x_n) *where the space between observation times is* not *constant.*

Definition 4 *A dataset* $\mathcal{D} = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\}$ *is a collection of pairs* (X_i, Y_i) *where* X_i *could either be a univariate or multivariate time-series with* Y_i *as its corresponding one-hot label vector. For a dataset containing* C *classes, the one-hot label vector* Y_i *is a vector of length* C *where each element is equal to* 1 *for the index corresponding to the class of* X_i *and* 0 *otherwise.*

The goal for general time-series classification consists of training a classifier on a dataset $\mathcal{D}$ in order to map from the space of

possible inputs to a probability distribution over the class variable labels. Recall from Section 1.3.1 that for photometric classification of astronomical transients, the light that is observed is collected through different filters over a period of time to form a light curve. However, the light curve is irregularly sampled in time and across passbands which further complicates the task.

2.2 Evaluating Classifiers

In order to develop and evaluate models one must consider what metrics are most suitable for the task at hand. This section presents the key evaluation metrics that were used to measure the performance of our classifier, and also describes the motivation for such metrics in relation to the photometric astronomical transient classification problem that is considered in this thesis.

2.2.1 Performance Metrics

Choice of evaluation metrics is of high importance when considering the performance of a classifier. This is compounded when dealing with imbalanced datasets since most metrics consider the setting of an even distribution of samples among the classes. One must be careful when considering which metrics to evaluate a model's performance since relatively robust procedures can be unreliable and misleading when dealing with imbalanced data (Branco et al., 2015; Malz et al., 2019b).

Typically, threshold metrics are used which consider the rate or fraction of correct or incorrect predictions. Threshold metrics are formulated by combinations of the four possible outcomes a classifier could have with regards to predicting the correct class:

- True Positive (TP): prediction of a given class and indeed it being that class.

- False Positive (FP): prediction of a given class but it does *not* belong to that class.
- True Negative (TN): prediction that an object is *not* a particular class and it is indeed *not* that class.
- False Negative (FN): prediction that an object is *not* a particular class but it is in fact that class.

From these outcomes common threshold metrics can be formulated, with perhaps the most common threshold metric being *accuracy*, which is the number of correctly classified samples over the total number of predictions. However, for imbalanced data results on accuracy alone can be misleading as a model can achieve high accuracy by simply classifying the majority class. More robust metrics for imbalanced data are precision and recall since their focus is on a particular class:

$$\text{Precision/Purity} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.1}$$

Precision gives the fraction of samples predicted as a particular class that indeed belong to the particular class, while recall, also known as the true positive rate, indicates how well a particular class was predicted. In cosmology, precision is often referred to as *purity*. The purity of SNe compared to other classes is a key metric that ultimately determines the usefulness of a classifiers results for cosmological analyses.

Confusion Matrix

One way to visually inspect the performance of a classifier with regards to threshold metrics is by the confusion matrix. The confusion matrix provides more insight into the performance of the model and reveals which classes are being predicted correctly or incorrectly. Often these tables are normalised across the rows to give probabilities in order to provide a more intuitive understanding. A perfect classifier across all classes would therefore be equivalent

to the identity matrix with all ones along the diagonal and zero elsewhere.

Receiver Operating Characteristic

An important point to note is that threshold metrics alone assume the class imbalance present in the training set is of the same distribution as that of the test set (He and Ma, 2013). On the other hand, a set of metrics built from the same fundamental components as threshold metrics, called rank metrics, do not make any assumptions about class distributions and therefore are a useful tool for evaluating classifiers based on how effective they are at distinguishing between classes (Brownlee, 2020).

Rank metrics require that a classifier predicts a probability of belonging to a certain class. From this, different thresholds can be applied to test the effectiveness of classifiers. Those models that maintain a strong probability of being a certain class across a range of thresholds will have good class separation and thus will be ranked higher.

The most common of this type of metric is the receiver-operating-characteristic (ROC) curve, which plots the true positive rate verses the false positive rate to estimate the behaviour of the model under different thresholds. The ROC curve is then used as a diagnostic tool to evaluate the model's performance, with every point on graph representing a given threshold. Interpolating between these points forms a curve, with the area under the curve (AUC) quantifying performance. A classifier is effectively random if the AUC is 0.5 and, conversely, is a perfect classifier if the AUC is equal to 1.0. It is common when reporting the AUC for multi-class classification to give the *macro*- and *micro*-averaged score. A macro-averaged score is calculated by considering the metric independently for each class and then taking an average. In this way, all classes are treated equally which is troublesome if one has highly imbalanced data. A micro-averaged score on the other hand aggregates the contributions of all classes in order to calculate the

metric. Therefore, it is advisable to consider micro-average scores when dealing with imbalanced datasets.

Precision-Recall Trade-Off

An alternative diagnostic plot to the ROC curve is the precision-recall (PR) trade-off curve. This is used in a similar way to the ROC curve but instead focuses on the performance of the classifier to the minority class, and hence is more useful for imbalanced classification problems (Brownlee, 2020). Much like the ROC curve, points on the curve represent different classification thresholds with a random classifier resulting in an AUC equal to 0.5 and a perfect classifier resulting in an AUC of 1.0. In addition, macro- and micro averaged scores can also be computed for PR curves, and the preference for using micro-averaged scores in the imbalanced setting remains.

2.3 Traditional Machine Learning Approaches

This section gives a high-level overview of machine learning and its applications to astrophysics in the context of SNIa classification. Machine learning is a sub-field of artificial intelligence that typically considers two distinct sections, *supervised* and *unsupervised* learning. Supervised machine learning involves using algorithms that have been trained with labelled data. Unsupervised, on the contrary, aims to learn from data directly without explicitly being shown what the true labels are. The approaches that follow are supervised learning techniques.

As mentioned, with supervised learning one trains an algorithm to learn from examples of true labels that distinguish between classes, using what is called a training set. A training set is a collection of data that has the associated true labels attached. After the algorithm has been shown *enough* examples, and it is able to

learn a mapping from input data to labels, it is then evaluated using unseen data called the test set. Often a third set of data, called the validation set, is derived from the training set and completely separate from the test set to monitor overfitting and help with selecting tunable model parameters (hyperparameters). It is the performance on the test set that is indicative of how the model may perform in the real-world.

Over the last decade a plethora of photometric classification algorithms have been developed. These stemmed from the fruitful Supernova Photometric Classification Challenge (Kessler et al., 2010b, SNPhotCC) in 2010 that focused on photometric classification of Supernovae only; and more recently the Photometric LSST Astronomical Time-Series Classification Challenge (Hložek et al., 2020b, PLAsTiCC) in 2018, which included a variety of different astronomical transient events among its classes.

Several challenges arise when observing photometrically; SNPhotCC and PLAsTiCC tried to simulate such conditions in terms of photometric sampling linked to the telescope cadence, as well as the distribution of classes one expects to observe. When creating such a simulated dataset, realistic distribution of classes is of great importance as often the training data available to astronomers is not of the same distribution one would observe through a real survey. This is due to Malmquist Bias (Butkevich et al., 2005), which is caused by the inherent bias towards observing brighter and closer objects when observing the night sky. As a consequence, training datasets are skewed to have more objects that are closer in distance, lower in redshift, and brighter in luminosity. In addition, the usefulness of observations of Type Ia Supernova has induced a bias towards spectroscopic follow-up of these events, resulting in vastly imbalanced training datasets that have a large number of Type Ia Supernova samples compared to other objects. The resulting training sets are therefore typically imbalanced and non-representative of the test sets that one might observe. These issues present a major challenge when developing classifiers. Sev-

eral methods have been proposed to address the problems of non-representativity and class imbalance.

Early attempts that applied machine learning methods to the SNPhotCC dataset can be found in Karpenka et al. (2013) using neural networks (neural networks are discussed in more detail in Section 2.4), in Ishida and Souza (2013) using kernel PCA with nearest neighbours, as well as methods found in Lochner et al. (2016) which compared a variety of techniques with impressive results on representative training data. Another successful approach can be found in Boone (2019) which was able to specifically extend the boosted-decision-tree (BDT) method in Lochner et al. (2016) by achieving good performance even in the non-representative training set domain. This work used BDTs coupled with data augmentation using Gaussian processes to achieve a weighted logarithmic loss (Malz et al., 2019b) of 0.68 in the PLAsTiCC competition (Hložek et al., 2020b) and 0.649 in a revised model following the close of the competition. This compares to a score of 0.0 for a perfect classifier and a score of 2.71 for a classifier that predicts all classes to be equally likely. However, one drawback with many of these methods is the reliance of the *human-in-the-loop*, where well crafted feature engineering plays an important role in achieving excellent scores. With few exceptions, such as the approaches of Lochner et al. (2016) and Varughese et al. (2015) that used wavelet features, many traditional machine learning approaches for photometric classification are model dependent, relying on prior domain specific information about the light curves.

2.3.1 Signal Processing with Wavelets

The work by Lochner et al. (2016) (`snmachine.v1`) is described in further detail here as it has been used in some earlier doctoral work (see Section 2.5.1 for details) as well as some elements of the data pre-processing steps being foundational to the new methods that are described in later chapters.

With the desire to move away from hand crafted features,

a model-independent signal processing approach was devised by Lochner et al. (2016). Their overall procedure was to first interpolate light curves onto a regular grid, and then perform wavelet decomposition. Then a dimensionality reduction is done to yield the feature set. The wavelet method and associated pipeline in Lochner et al. (2016) automatically acquires specialised features that can be passed through a variety of standard classifier algorithms. In this case, a wavelet analysis is applied to decompose the signal into a set of wavelet coefficients. Synonymous with Fourier decomposition that break a periodic signal down into a linear combination of sine and cosine coefficients, wavelet decomposition yields a similar set of coefficients without the loss of time and frequency information (Mallat, 2008).

However, to decompose a signal using wavelet analysis, the signal itself needs to be evaluated on a regular grid. As described in 2.1, the data that is observed is irregular in both passband and time. Thus it is necessary to interpolate the light curve such that a standard spacing of points in time can be evaluated.

A powerful tool for regression is the use of Gaussian processes, which one can use as an interpolation method that is able to capture the uncertainty contained in the data points. A Gaussian process can be fully specified by a mean function $\mathbb{E}[f(x)] = m(x)$ and covariance between two sampled observations x, x' as $\mathrm{Cov}(f(x, x')) = \mathbf{K}_f(x, x')$, where $\mathbf{K}_f(\cdot, \cdot)$ is a kernel. Of the wide variety of kernels one could choose, for simplicity and ease of use, the kernel used in `snmachine.v1` is the 1-dimensional squared-exponential kernel,

$$\mathbf{K}_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{|x - x'|^2}{2\ell^2}\right), \tag{2.2}$$

where the length scale ℓ determines the length of the fluctuations in the function, and σ^2 determines the function's average distance from the mean (Duvenaud, 2014).

Therefore, after using Gaussian processes to interpolate the light curve, the next step of wavelet decomposition is possible.

As outlined in Narayan et al. (2018) and Lochner et al. (2016) a suitable family of wavelets for transient classification are *symlets*, which are a form of stationary wavelet transforms that are able to capture the properties of the light curve and also have the benefit of being translation invariant.

After applying the wavelet decomposition to the regularly sampled light curve the resulting coefficients are features are highly redundant. While thresholding of the wavelet coefficients can be used to compress the data to some degree, this needs to be consistently applied across all scales. An alternative that allows for more flexibility with dimensionality reduction is principle component analysis (PCA), which is carried out following the wavelet decomposition step to reduce dimensionality in accordance to the size of dataset.

2.4 Neural Networks and the Deep Learning Revolution

As mentioned in the previous Section 2.3 neural networks (NNs) have been in use within astronomy for some time with the work of Karpenka et al. (2013) for photometric classification, as well as many other applications in astrophysics including work by Lahav (1996) that used NNs for galaxy classification over 20 years ago. A neural network, shown in Figure 2.1, consists of an input layer, one or more hidden layers and a final output layer. Each circle in Figure 2.1 represents a *neuron*, or activation a, that is simply a real number. The input layer of neurons is built from the features one provides to the network, whereas the following layers of neurons are determined by a linear combination of the weights, w, *i.e.* the lines between each neuron, and the previous layers activations. These are put through an activation function $f_{\text{act}}(\cdot)$ along with a bias term, b that helps guide the activation function output. The operation

between input and the first hidden layer can been seen in Figure 2.2, with similar operations taking place for each layer thereafter.

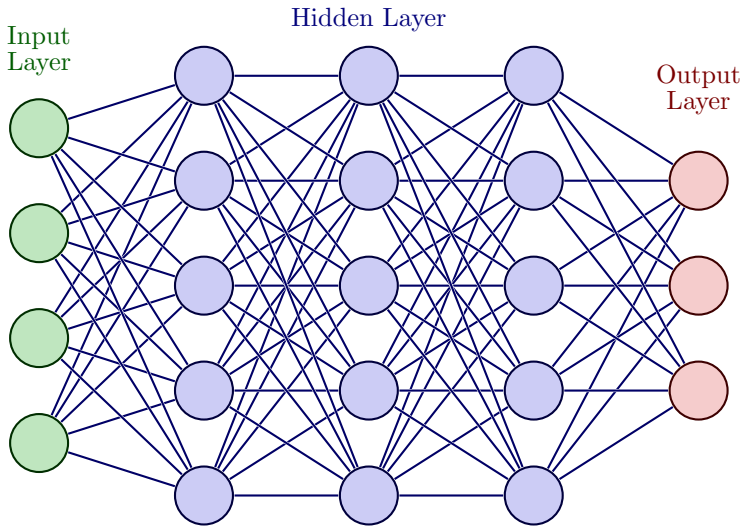


FIGURE 2.1: Basic neural network structure. Real-valued inputs are passed through the input layer of neurons, which are then passed through one or more hidden layers of neurons. Each connection between neurons is associated with a weight corresponding to the strength of that connection. A non-linear function is applied to the linear combination of the weight values and the previous layer's neurons. This is fed into the output layer that can yield a single real-value or many real-values. If the task at hand is classification, it is common to pass the set of values in the final layer through a softmax activation function to yield a probability score for each class.

The shape of the output layer is informed by the task at hand and the loss function and unlike all other layers, the output layer neurons typically do not have a non-linearity activation function. For regression tasks the output is real-valued and can be a single neuron or a neuron per target (*e.g.* a real-value pixel value per-

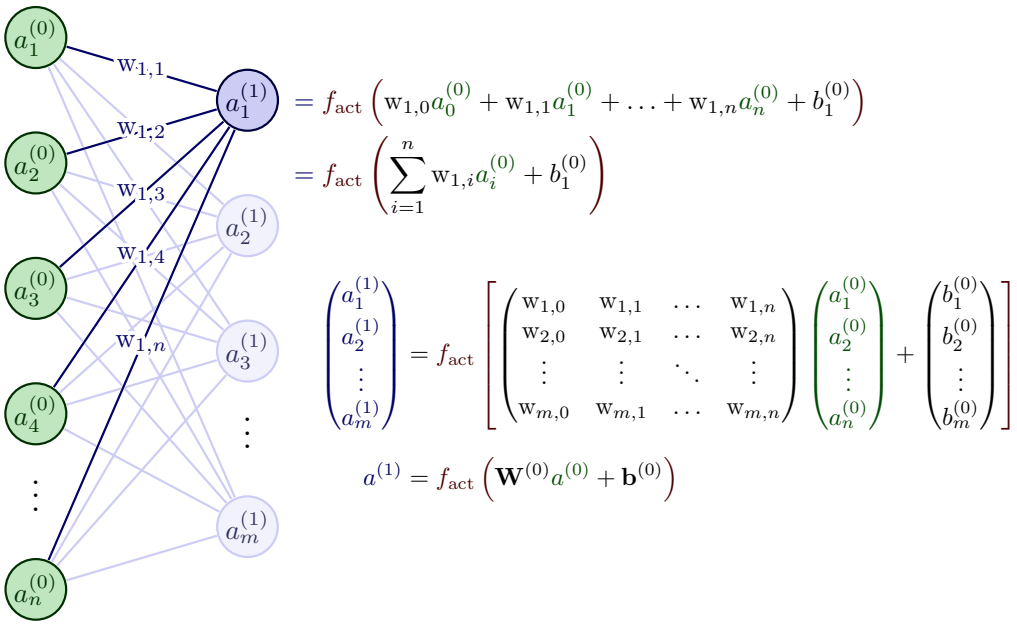


FIGURE 2.2: Neural network internals. Each neuron is determined from the linear combination of the previous layers weights and biases, which is then passed through a function, called an activation function, to induce non-linearity and aid with learning. This diagram shows the input layer in green and a hidden layer in blue. For each connection of the network there exists an associated weight, w, that describes the strength of that connection, and for each neuron there exists a bias term that relatively adjust the level of activation. The weights and biases together are known as the parameters of the network and is determined above by number of neurons from a previous layer × number of neurons in current layer for the weights, added with number of neurons in the current layer for the biases.

target pixel for the task of image inpainting). For classification tasks there is usually an output neuron for each class representing class scores that is also real-valued. A softmax function is often

then used to normalise the output to a probability distribution over predicted classes.

The loss function is a differentiable function that is used to determine how well aligned the weights of the network are such that the desired output is achieved. A simple example loss function typically used in regression tasks may be the Euclidean distance between a ground truth label and predicted outputs. For each forward pass of the network one compares the ground truth with the predictions and computes the loss. As the goal is to reduce the distance between ground truth and prediction (*i.e.* minimise the loss) we can frame this as an optimisation problem by taking the derivative of the loss function with respect to the weights of the network. Then for each batch of data[1] the weights are updated using the canonical Stochastic Gradient Decent (SGD) algorithm shown here,

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}')}{\partial \mathbf{w}}, \tag{2.3}$$

where $\mathcal{L}(\boldsymbol{x}, \boldsymbol{x}')$ is the network loss function between the input batch of data $\boldsymbol{x}$ and the predictions for that batch of samples $\boldsymbol{x}'$, $\mathbf{w}$ is the weight vector, and η is the learning rate or step size of each update. To help with convergence, one typically uses a learning rate *schedule* that reduces the learning rate over time. There are many options one could choose in terms of loss function, and this in itself it a complicated task. The loss function used in this thesis that focuses on classification is described in more detail in Chapter 3.

As signal through the network is determined by prior computations of the weights and biases, the final output of the network is thus controlled by this set of weights and biases. Hence, these are known as the parameters of the network. There are other factors that ultimately determine the output of the network, such as the depth and width of the hidden layers, as well as the learning rate

[1]Traditionally batch refers to all samples in the training set, but it has become commonplace to refer to batch as a disjoint set of the training data samples. This is also called minibatch, and can be used interchangeably

and schedule mentioned above, but as these are set by the practitioner rather than the learning algorithm itself, they are referred to as hyperparameters. Even for shallow networks with only a couple of hidden layers, the number of weights and biases (parameters) one needs to account for becomes very large.

2.4.1 Convolutional Neural Networks

Of late, there has been a resurgence of using neural networks and an explosion of *deep* learning methods applied to many areas in the physical sciences. Deep learning in this context is broadly related to that described above for neural networks, but with many hidden layers. Much of this is attributed to the success of Krizhevsky et al. (2012, `AlexNet`) when applied to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Deng et al., 2009). Their profound improvement in the state-of-the-art spawned significant efforts that applied similarly deep neural networks, with particular focus on convolutional neural networks (CNNs) to problems in the image domain and recurrent neural networks (RNNs) (discussed in the next section) for sequence modelling. Similar to the classical fully connected neural network architecture shown in Figure 2.1, the general structure of a CNN can be seen in Figure 2.3. The immediate difference is in the convolutional layers, shown in orange, is that not all neurons are connected. This is due to the weight sharing nature of the convolution operator across the receptive fields and is at the heart of convolutional neural networks. CNNs, initially proposed by LeCun et al. (1989a) for handwritten digit recognition, had their revival following the breakthrough work of Krizhevsky et al. (2012) mainly due to the abundance of data with which one can train deep neural networks, and perhaps more importantly, due to the progress made in hardware accelerators such as Graphical Processing Units (GPUs) for machine learning tasks, which now feature in all winning solutions to the competition.

A normal convolution operation (other variants discussed in later chapters) seeks to find cross channel information and spatial

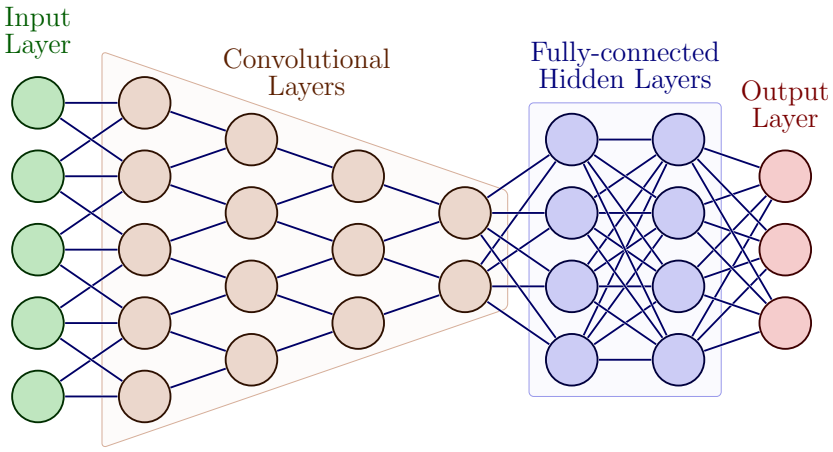


FIGURE 2.3: Typical convolutional neural network architecture with an input layer shown in green, a set of convolutional layers, followed by a fully connected neural network and a final output layer. Note the difference in number of connections between the convolutional layers and the fully-connected layers, due to weight sharing within convolutional layers.

information at once. For 2-dimensional input such as an image, channels could refer to R, G, B filters of an image and spatial referring to the height and width of an image. For 1-dimensional input such as time-series, channels may refer to the number of features in a multivariate time-series, M, and spatial actually referring to the temporal axis.

The operation itself slides a filter, also called a kernel over the input. The step size for which the filter is passed along the input is called the stride length. At each location the Hadamard product is computed which results in a single value in the output *feature map*. A full feature map is formed from the repeated application of the filter to the entire input. The replication of the filter across the input results in a shared parametrisation of the weights and biases (also referred to as weight-sharing) for each resulting feature

map, a key attribute of convolutional layers. For the first layer, The kernel size relative to the input is the *receptive field* and for images would have size height and width, whereas if one is dealing with time-series, we say the kernel is operating with a receptive field of window size, w. As we go deeper, the size of the receptive field can change and more of the signal, or output feature map is influenced by the kernel. Note the difference in notation between weights, w and kernel window size, w. This should also be dissociated to w described in Chapter 1 which described the dark energy equations of state parameter. An example of this operation for time-series of M-channels by L-length input is shown in Figure 2.4.

The operation can be chained together and constructed into its own type of neural network architecture.

2.4.2 Recurrent Neural Networks

With the two major milestones of improved compute power and an abundance of data, other *older* deep learning methods for sequential modelling such as RNNs (Jordan, 1997), were also revitalised. RNNs are essentially neural networks with loops that allow for information to be persisted as different inputs are passed in. This can be better understood in Figure 2.5 that on the left-hand side depicts a block of hidden layers, $\boldsymbol{h}$, which takes in x_i as input and produces y_t output after some number of recurrent steps. On the right-hand side of the figure is the unrolled version that shows how a sequence can be processed. As such, RNNs lend themselves well to sequence modelling tasks such as machine translation, time-series forecasting, and relevant to us, time-series classification.

Recurrent neural networks consist of repeating modules of hidden layers of neurons, where in the most basic form the hidden layer may be a single neuron activation. The combination of all neurons in the hidden layers $h_i \dots h_L$ form the *hidden state* (the representation of all previous inputs). Although the weights of a RNN are shared across these hidden layers, as the sequence of inputs grows, propagating meaningful updates to the weights be-

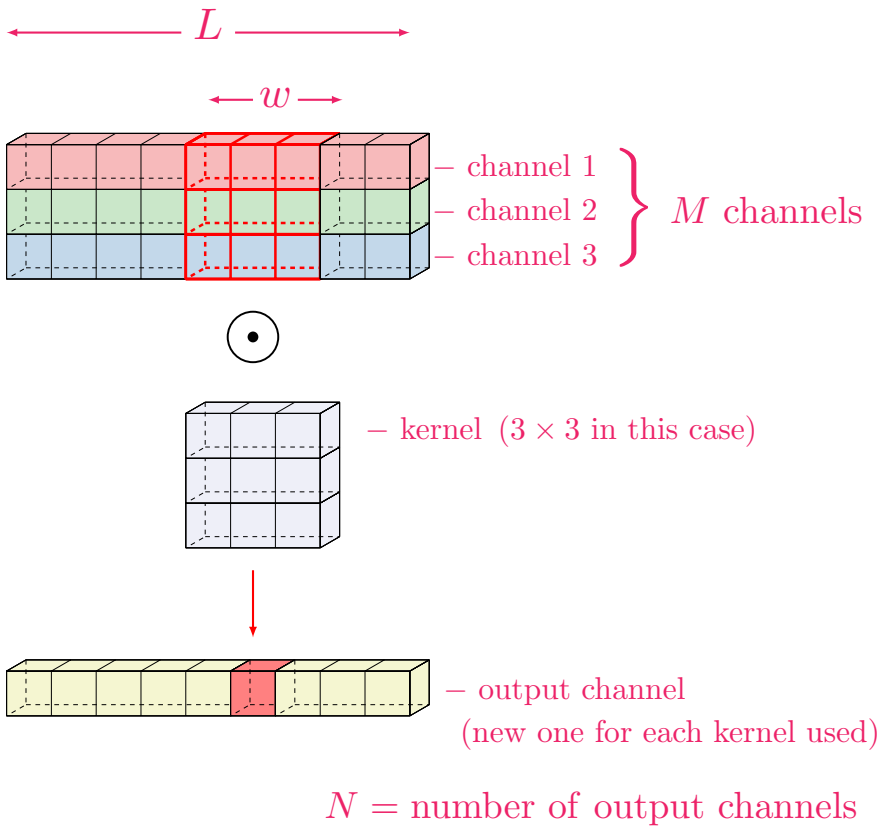


FIGURE 2.4: Normal convolutional operation. Spatial and cross-channel correlations are obtained in one step by sliding a kernel of width w, over the time-series signal (whilst three channels, red, green and blue, are depicted here, the normal convolution is not limited to the number of channels that can be operated on). The resulting output from the normal convolution shown in this figure can be seen as the red square below. Should another kernel or filter be used on the same signal, a further resulting output channel is created *i.e.* there is a resulting output channel for every kernel that is applied.

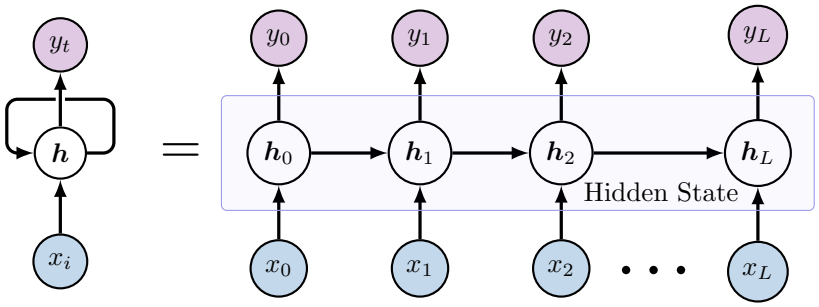


FIGURE 2.5: Basic structure of recurrent neural networks, which is similar to that of feed-forward neural networks, with the main difference being the looping pathway within the hidden state. This allows for information to persist and be retrained by the network as more inputs are ingested.

comes more challenging. Often referred to as the unstable gradients problem (Hochreiter et al., 2001), if weights are large, then through application of the chain rule to compute $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$ in Equation 2.3 will cause the gradient value to *explode*. Similarly, if the weights are small, the gradient can *vanish* towards zero. A variant of RNNs that has two separate paths for passing long-term information and short-term information through the network called Long-Short-Term-Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) has shown to mitigate this problem but consequently need to keep track of an increasing amount of information as an input sequence grows.

Perhaps the most popular construction of RNNs for sequence-to-sequence (Seq2Seq (Sutskever et al., 2014)) tasks are in the form of *encoder-decoder* architecture, shown in Figure 2.6. The inputs $\boldsymbol{x} = [x_0, x_1, \ldots, x_L]$ are processed in the encoder to create a hidden state $\boldsymbol{h} = [h_0, h_1, \ldots, h_L]$. The outputs of the encoder are discarded but the hidden state is combined with any additional information to form a context vector, $\boldsymbol{c}$. This additional information could be cell states *i.e.* long term memory information in the case of LSTMs,

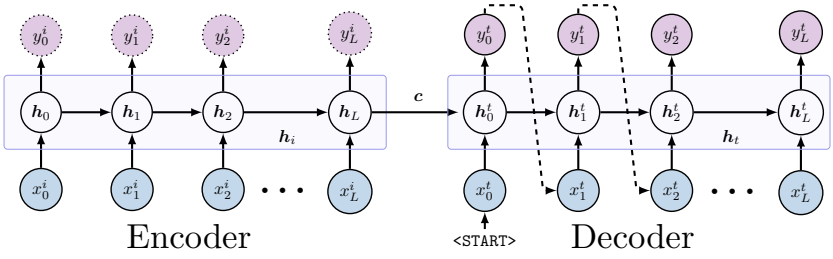


FIGURE 2.6: Recurrent neural network encoder-decoder construct where the hidden state of the encoder is passed to the decoder along with any additional information to form a context vector. The context vector is then passed to the decoder along with a start token as input, with the aim of learning the output sequence auto-regressively.

or in the case of attention (discussed in more detail in Chapter 4) the alignment scores.

The goal of the context vector is to encapsulate all the information of the input sequence in a compressed form such that it can then passed along to the decoder on the right of Figure 2.6 to help make predictions.

The decoder then takes as input the context vector and a start token (`<START>` shown in Figure 2.6) in place of x_0^t signifying the start of a new sequence. For each time-step in the decoder, the predicted output is used as the input for the next time-step (*i.e.* auto-regressively), ultimately building to a full target sequence prediction for $\boldsymbol{y} = [y_0, y_1, \ldots, y_L]$.

2.4.3 AlexNet for Time-series Classification

CNNs, RNNs and many such variants have come to dominate nearly all areas of supervised learning. Naturally, this also came to bear for the task of time-series classification, and it was the area of activity recognition that first encouraged a platform for experimentation of applying deep learning methods following the release

of the Wireless Sensor Data Mining (WISDM) project (Kwapisz et al., 2011) and then again with an extended updated version (Weiss et al., 2019). Similar to the SNPhotCC and PLAsTiCC challenges mentioned earlier, these datasets were presented to the community such that a comparison of methods could be evaluated on a standard multivariate time-series dataset.

Work by Bagnall et al. (2018) collated a varied set of multivariate time-series datasets with the intention of benchmarking the many classification algorithms that were emerging. The dataset itself came to be known as the multivariate time-series benchmark dataset (MTS). As with the astronomical challenges, early work used traditional machine learning methods but soon moved into the deep learning direction. Hoping to find a similar success story of `AlexNet` (Krizhevsky et al., 2012) for the domain of time-series classification, Fawaz et al. (2019) explored the latest trends in using deep learning for this task, but also considered the quality of the more traditional machine learning approaches. Of the best performing methods, CNNs were frequently among the top when comparing against the many different datasets contained in the MTS. While the other approaches such as dynamic time-warping (Shokoohi-Yekta et al., 2017) and shaplets (Ye and Keogh, 2011) still perform well, the appeal of minimal feature engineering with deep learning architectures is quite apparent in the new methods proposed of late (Ruiz et al., 2021).

Likewise for photometric classification, there have been recent attempts to apply deep learning to minimise the laborious task of feature selection, and in some cases input raw time-series information only. Work by Brunel et al. (2019) used an Inception-V3 (Szegedy et al., 2015b) inspired convolutional neural network (CNN) and earlier work by Charnock and Moss (2017) used a long-short-term-memory (LSTM) recurrent neural network (RNN) for Supernovae classification. Möller and Boissière (2020) also achieve good results building upon the success of RNNs. Extending to the general transient case and utilising an alternative RNN architecture, gated reticular units (GRUs), work by Muthukrishna et al.

(2019, `RAPID`) showcased the impressive results one could achieve by using the latest methods borrowed from the domain of sequence modelling and natural language processing (NLP). While these deep learning methods have been shown to yield excellent results, both RNNs and CNNs have several limitations when it comes to dealing with time-series data.

RNNs tend to struggle with maintaining context over large sequences due to the unstable gradients problem (Hochreiter et al., 2001) as described in Section 2.4.2. When an input sequence becomes long, the probability that we will be able to maintain the context of one input to another decreases significantly with the distance from that input (Madsen, 2019). The shorter the paths between any set of positions in the input and output sequence, the easier it is to learn dependencies (Hochreiter et al., 2001). Note that the *maximum path length* of an RNN is then given by the length of the most direct path between the first encoder input and the last decoder output (Grosse and Ba, 2019) (*e.g.* x_0^i to y_L^t as shown in Figure 2.6). Another problem faced by the RNN family is the inherently sequential structure, making parallelisable computation difficult as each input point needs to be processed one after the other, resulting in a computational cost of $\mathcal{O}(L)$, where L is the sequence length (Vaswani et al., 2017).

CNNs overcome these problems, to some extent, with trivial parallelism across layers and, with the use of the dilated convolution, distance relations can become an $\mathcal{O}(\log L)$ operation, allowing for processing of larger input sequences (Oord et al., 2016). However, CNNs are known to be computationally expensive with a complexity per layer given by $\mathcal{O}(w \cdot L \cdot d^2)$, where w is the kernel window size and d the representational dimensionality (Vaswani et al., 2017). For contrast, RNNs have complexity per layer $\mathcal{O}(L \cdot d^2)$.

2.5 Research Overview

This section briefly describes the work that has been conducted over the course of my doctoral research. Fortunately I have had the opportunity to be involved in several collaborative projects during this time. However, many of these projects are auxiliary to the main work present in the thesis. Nevertheless, I first discuss the works that I was involved in, and briefly describe my contributions to the research. Second, I give an outline of my work that *does* feature in this thesis, along with the associated chapters where they are described.

2.5.1 Collaborative Contributions

Metric Design

The wide science goals of the Legacy Survey of Space and Time at the Vera C. Rubin Observatory (LSST) demand improved metrics that consider the deluge of low-signal-to-noise data that is expected, and allow for probabilistic results. Traditional metrics that return a single value disregard the notion of uncertainty, which is extremely important for possible follow-up analysis that may occur. Work by Malz et al. (2019b) developed a performance metric for a citizen science challenge: the Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC), that required probabilistic classifications. A weighted modification of the cross-entropy metric was proposed as it could be meaningfully interpreted, as well as provide a level of uncertainty around the final result.

My contributions involved assessing the potential probabilistic metrics to investigate. Through implementation of trial metrics in `proclam`[2], for which I took ownership of developing the associated

integration testing framework, we could evaluate their suitability on typical data (Malz et al., 2019b).

Cadence Optimisation

The cadence that is eventually settled on for the Vera C. Rubin Observatory will play an important part in our ability to do SNIa cosmology. With the expected number of SNe being far larger than previous surveys, it is not feasible for all to be spectroscopically followed up. Therefore, being able to photometrically classify transients well will allow us to leverage the power of the datasets LSST will provide and further constrain cosmological parameters. This work used an established machine learning pipeline (Lochner et al., 2016, `snmachine.v1`) to comparatively study different proposed observing strategies of the LSST. The aim was to determine the optimal cadence suited for classification of SNe light curves. In order to conduct this analysis, SNANA (Kessler et al., 2009) was used to generate the light curves that correspond to different cadences runs from the observing strategy simulator (Marshall et al., 2017, `OpSim`) outputs.

My preliminary results featured in initial observing strategy white papers (Lochner et al., 2018b; Scolnic et al., 2018b) and conference proceedings in Allam Jr et al. (2019). An updated analysis, led by Alves et al. (2022a), now using `snmachine.v2` (Lochner et al., 2021) compares the different cadence strategies, where I contributed to the release and upgrading of `snmachine.v2`[3], as well as integration testing, to be able to handle the new simulated light curve data.

Real-time Science Infrastructure

To enable science with the large time-domain alert streams such as the one from the upcoming LSST, development of a brokering system that can easily digest the large volume of data and make

[2] github.com/aimalz/proclam
[3] github.com/LSSTDESC/snmachine

it consumable to everyday scientists, is necessary. FINK (Möller et al., 2021) is a broker infrastructure that allows for a wide range of applications and services to connect to streams of alerts, and facilitates traditional astronomy broker features such as automated ingestion and annotation, as well as identification of promising alerts for transient science.

As an early member of the FINK team, I contributed to the open-source project design, initial infrastructure planning and implementing integration tests for the FINK broker[4]. I was also involved with drafting the first version of the publication (Möller et al., 2021).

2.5.2 Thesis Outline

The driving force for the research contained in this thesis is the need for real-time classification algorithms that can scale well for the large surveys of tomorrow. With the need for low-latency high-throughput systems to eventually be deployed to enable real-time science, there is a focus towards developing model independent classifiers[5], that can input raw time-series, but with resource constraints in mind. As such, our novel architectures showcased herein leverage and extend the latest developments in efficient machine learning research to minimise computational cost, yet still provide state-of-the-art classification performance. To demonstrate effectiveness for real-world application, my later research takes a more pragmatic approach for the development of these models for deployment into production systems working with real-world streaming data.

The remainder of this thesis is structured as follows. Presented in Chapter 3 is a new architecture dubbed the *astronomical-transient xception* (`atx`), developed for photometric classification, but suitable for general time-series classification, that extends ideas from the current state-of-the-art in multivariate time-series classifi-

[4] github.com/astrolabsoftware/fink-broker

cation by leveraging the depthwise-separable convolution for faster, more efficient operations. Chapter 4 introduces another novel architecture, also developed for photometric classification but applicable for general time-series classification, called the *time-series-transformer* (`t2`), that uses self-attention to achieve significant performance gains over previous models, at reduced computational cost and model size. Using real ZTF streaming data, Chapter 5 presents the engineering work involved in testing, validating and optimising from a computational perspective for deployment of these architectures as science modules into the production brokering system of FINK mentioned in 2.5.1 for real-time photometric classification. Finally, the conclusion is given in Chapter 6 with a discussion of the works presented and a look towards the future, highlighting possible avenues of further research.

[5]Historically, light curve features derived from model template fits such as SALT2 (Guy et al., 2007) have been used to photometrically classify SNe.

Part II

Research

What I cannot create, I do not understand

— Richard P. Feynman

3

An Astronomical Xception: Depthwise-Separable Convolutions for Efficient Photometric Classification

"*To deal with a 14-dimensional space, visualize a 3-D space and say 'fourteen' to yourself very loudly. Everyone does it.*"

— **Geoffrey Hinton.**

In this chapter we propose a new architecture applicable for general multivariate time-series classification, that leverages the efficient machine learning operation of depthwise-separable convolutions to achieve near state-of-the-art photometric classification. The astronomical-transient xception (`atx`) network moves away from explicit feature selection and can achieve good performance with raw time-series data alone. Nevertheless, the option to supplement with additional features such as redshift is available, and we achieve a logarithmic-loss of 0.739 on imbalanced data in a representative setting with data from PLAsTiCC. Furthermore, `atx` achieves a micro-averaged receiver operating characteristic area under curve of 0.98 and micro-averaged precision-recall area under curve of 0.87.

3.1 Introduction

In the modern era of large scale transient surveys, such as the upcoming LSST (Ivezić et al., 2019), a deluge of time-series data,

an order of magnitude greater than has been observed before is expected. To handle the sheer volume of events, machine learning is critical to be able to accurately photometrically classify events, as well as to discern what is of particular astrophysical interest for further spectroscopic follow-up.

The general problem of time-series classification is one that extends to a vast number of disciplines, many of which look to machine learning for improved performance. Traditional machine learning approaches involved hand-crafted feature engineering to uncover patterns that would be useful for classification. Today, with the sheer volume of data, deep learning methods are being investigated as a promising alternative to previous methods for classification (Fawaz et al., 2019).

A common drawback that comes with using deep learning methods is the associated computational cost in both space and time, *i.e.* memory footprint and runtime. The goal for the doctoral research presented in this thesis is to develop novel deep learning architectures that are computationally efficient and can perform state-of-the-art photometric classification, in real-time.

This chapter is structured as follows. Section 3.2 recounts the evolution of modern CNNs and how this led to the development of the Xception network (Chollet, 2017), the architecture for which motivated much of the work in this chapter. Section 3.3 reviews the fundamental component of the Xception network, the depthwise-separable convolution, in more detail and describes how it can improve computational efficiency in 3.3.2. Section 3.4 presents our Xception-inspired architecture, the astronomical-transient xception network (`atx`), that leverages 1-dimensional depthwise-separable convolutions for efficient photometric classification. We then describe the implementation and performance metrics used to evaluate our models in Section 3.5. Subsequently, the results of applying our astronomical-transient xception architecture to PLAsTiCC data (The PLAsTiCC team et al., 2018b) are showcased in Section 3.6. Finally, in Section 3.7 a discussion of the work carried out and prospects for further analyses is given.

3.2 The Convolutional Neural Network Story

This section walks through the recent history of CNN architectural advances over the last decade, culminating with the Xception network and the use of depthwise-separable convolutions, which are at the core of our new architecture for time-series classification.

As touched on in Section 2.4, the first landmark use of CNNs was by LeCun et al. (1989c, `LeNet-5`) for handwritten digit recognition. Impressive for its day, several factors[1] meant that CNNs conceptually lay dormant until the seminal work by Krizhevsky et al. (2012, `AlexNet`) sparked the resurgence of CNNs for practically all areas of image analysis. Since their winning submission to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Deng et al., 2009) in 2012, all successive winning submissions have revolved around some form of CNN architecture.

The `AlexNet` architecture introduced two new components, local response normalization and rectified linear unit (ReLU) activation functions, with the latter becoming a staple in CNN architectures since. The architecture was an order of magnitude larger than `LeNet-5` with around 60 million parameters (*i.e.* weights and biases), mostly concentrated in the final fully connected layers. It is worth noting that the overwhelming majority of the computations were within the convolutional layers, yet these layers only contained a small fraction of the total parameters.

The following year saw another CNN (Zeiler and Fergus, 2014, `ZFNet`) take the crown by improving hyperparameter selection such as filter size and stride length. It was later argued that further improvements could be achieved by making the networks *deeper* and work by Liu and Deng (2015, `VGG-16`) showed that by increasing depth of the network more non-linearities could be captured by the model. However, a consequence of going deeper was an increase in

[1] Including a restrictive patent which barred even the creator, Yann LeCun, from working on its development for many years (LeCun, 2021).

parameter count and hence memory footprint, with approximately 138 million parameters.

Paying homage to the original `LeNet-5`, Google introduced a network that built upon the idea that deeper networks would improve performance, called `GoogLeNet` (Szegedy et al., 2015a)[2]. Their focus was to not only achieve good classification results, but to accomplish this with better efficiency than before. Removal of the final fully connected layers allowed for total parameter counts to be reduced to around 5 million, and much of the network's ability to still reach the top spot in ILSVRC-14 was due to their introduction of the *Inception* module.

The Inception module contains several convolutional operations all within a single "block", or collection of components that is repeated throughout the network (an example of the original `Inception-V1` module is shown in Figure 3.2). The key idea was to use parallel pathways of convolutions that had different kernels followed by a concatenation in order to capture different features together. Another helpful component was the use 1×1 convolutions, also referred to as a *pointwise convolution*, inspired by Lin et al. (2013) to allow for dimensional reduction and extension where necessary. A visual representation of the pointwise convolution in action can be seen in Figure 3.1 that applies a 1×1 ($\times M$ channels) convolution N number of times to achieve the desired output dimension.

Unfortunately, the charm of simply going deeper and adding more layers for better performance hits a stumbling block beyond a certain point. Using the standard backpropagation update algorithm described in Equation 2.3, when the gradient of the weights is less than one, it can be difficult for the resulting update to be meaningfully propagated and learning becomes extremely slow, resulting in the *vanishing-gradients-problem*. Similar to the unstable gradients problem described in the previous chapter, the gradients of the weights, $\mathbf{w}$ with respect to the loss $\mathcal{L}$ of early layers *i.e.* near

[2]Also known as the `Inception-V1` architecture

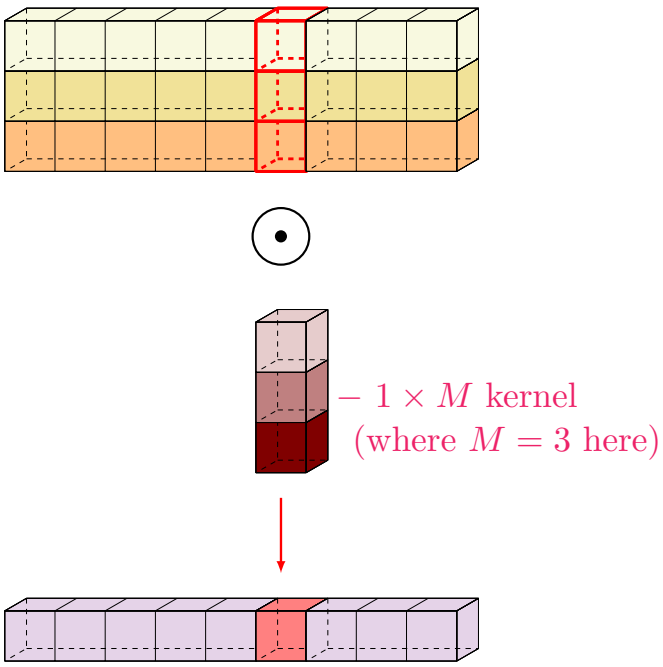


FIGURE 3.1: Pointwise convolution introduced by Lin et al. (2013) to help control dimensionality in CNNs. By successively applying pointwise convolutions, also know as 1×1 convolutions, on an input, one can control the number of output channels that are produced. This allows for arbitrary scaling from M input channels to N output channels.

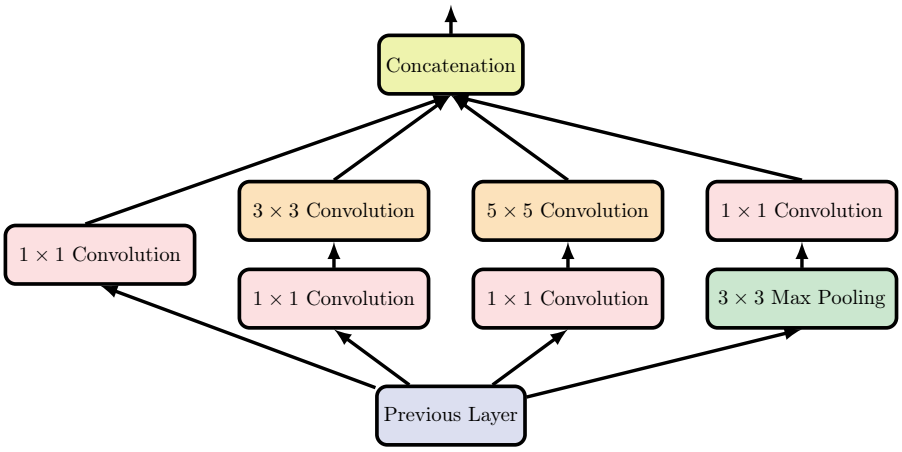


FIGURE 3.2: The Inception block as featured in `GoogLeNet` a.k.a `Inception-V1` (Szegedy et al., 2015a). Inputs as they come in are branched out with kernels of varying sizes operating on each factorised input. The final output is formed from the concatenation of the results of each convolution. Depending whether a dimension reduction or extension is necessary a side channel 1×1 convolution can assist in this regard.

to the input are obtained via the chain rule from the downstream layers near the output. As a result, if the gradients towards the end are near zero, their effects will have *vanished* in the eyes of the early layers. Another problem that also emerges with very deep networks is the *degradation problem*, where training and test errors tend to degrade as the number of layers increases, even when observed to not be overfitting (He et al., 2016).

To overcome these problems He et al. (2016, `ResNet`) introduced the concept of *residual-blocks* which are connected to one another through identity mapping like that shown in Figure 3.3. They hypothesised that if one considers $\mathcal{H}(\mathbf{X})$ as the underlying mapping that is to be fit by a set of stacked layers with $\mathbf{X}$ referring to the input to these layers, it is instead easier to optimise the residual, $\mathcal{F}(\mathbf{X}) := \mathcal{H}(\mathbf{X}) - \mathbf{X}$, than the underlying mapping directly. In this way, one can reformulate the original mapping $\mathcal{H}(\mathbf{X})$ to be defined as $\mathcal{F}(\mathbf{X}) + \mathbf{X}$ as depicted in Figure 3.3. Under the general hypothesis that consecutive non-linear layers can asymptotically approximate complicated functions, they suggest it should be equivalent to say that consecutive non-linear layers can asymptotically approximate the residual function, $\mathcal{H}(\mathbf{X}) - \mathbf{X}$ also[3]. As such, the residual blocks simply learn the deviations from the input instead learning the identity mapping directly.

The ability to *skip* layers and learn only the residual mapping resulted in faster training convergence and a better top-1 error rate, which describes the error rate of a classifier that outputs a probability score across all classes to give the highest score to the correct class, and ultimately won them 1st place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Deng et al., 2009) of 2015. The `ResNet` architecture showcased the usefulness of simple skip connections by being able to go to greater depths than before without badly degrading performance. It was also one

[3]Under the assumption that the input dimensions are the same as the output. To fix the situation where the dimensions are not the same, a 1×1 convolution can be used for dimensionality conversion

of the first significant uses of batch normalization proposed by Ioffe and Szegedy (2015), and using 1×1 convolutions for dimensional manipulation, it was the first network to surpass human-level accuracy.

Later versions of the Inception module by Szegedy et al. (2017, `Inception-V4`) also incorporated this idea of having residual blocks, with the architecture often referred to as the `Inception-ResNet` network in the literature as well. This was added to the previous version (Szegedy et al., 2015b, `Inception-V3`) which leveraged spatially separable convolutions that breaks down a typical single kernel of size $k \times k$ into two composite kernels of size $k \times 1$ and $1 \times k$ applied one after the other to save on computational cost and number of parameters whilst still operating over the same receptive field (Chen et al., 2020).

3.2.1 The Inception Hypothesis

Compared to previous architectures of simple sequential layers of convolutions such as `VGG-16`, the Inception module with varying kernels and parallel towers of operations appeared to be capable of learning superior representations with fewer parameters. Work by Chollet (2017) sought to better understand the mechanisms of the Inception module and asked three prime questions: *How do they work? How do they differ from regular convolutions?* and *What design strategies come after Inception?*

In search for answers to these questions, the *Inception Hypothesis* was put forward by Chollet (2017) which gave the interpretation that there is a spectrum of different types of convolution operations one could apply. Under this interpretation, they state that Inception modules in CNNs sit in-between normal convolutions at one end of the spectrum, and *depthwise-separable convolutions* (Sifre and Mallat, 2014) at the other. As described in Chapter 2, normal convolutional kernels are tasked with simultaneously mapping cross-channel correlations and spatial correlations. With the Inception module, this is made easier and more efficient by breaking

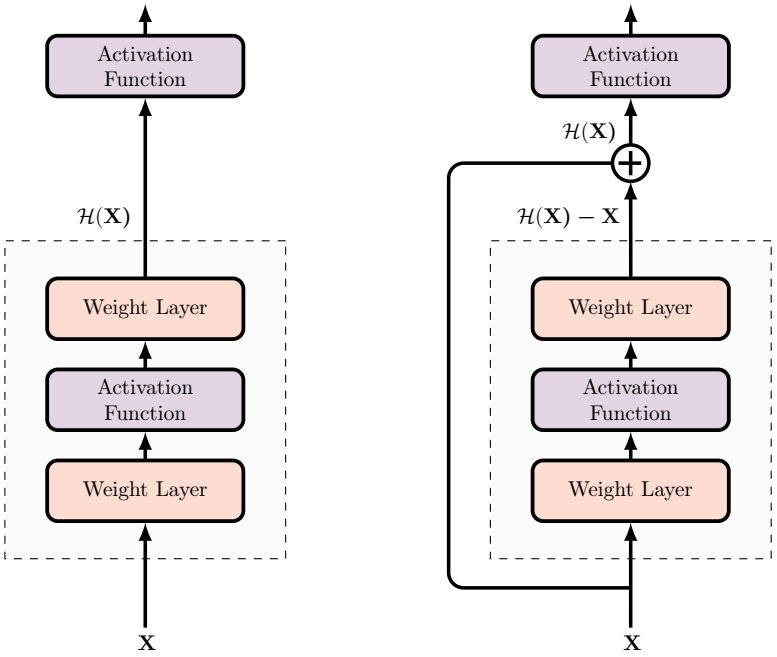


FIGURE 3.3: Example of residual blocks as presented in He et al. (2016). In a normal block (left) the set of stacked layers that are shown within the grey box must directly learn the mapping from the inputs to the output of $\mathcal{H}(\mathbf{X})$. However, the same set of stacked layers in the residual block (right) only needs to learn the residual mapping $\mathcal{H}(\mathbf{X})-\mathbf{X}$. If the underlying mapping should be that of the identity mapping $\mathcal{H}(\mathbf{X}) = \mathbf{X}$, then learning is made considerably easier as all that is required is to set the preceeding weight layer to zero. For this set up to work, where the dimensionality of the input is different from that of the outgoing weight layer, a 1×1 convolution can be used to bring the identity back into alignment for the addition operator.

down the procedure into a series of operations that independently find cross-channel correlations and spatial correlations. In the Inception module, cross-channel correlations are found using 1×1 convolutions and the spatial correlations found via 3×3 or 5×5 convolutions.

Ultimately, the Inception hypothesis suggests it is better to decouple cross-channel correlations and spatial correlations instead of attempting to map them together. A stronger hypothesis put forward by Chollet (2017) is that these can be mapped completely separately. Their *extreme* version of the Inception hypothesis advocates for 1×1 convolutions first to find cross-channel correlations and then to separately map the spatial correlations of each output channel. This extreme form of an Inception module described by Chollet (2017) is essentially the depthwise-separable convolution of Sifre and Mallat (2014), with the main difference being the order of operations since depthwise-separable convolutions typically map spatially first and then look for cross-channel correlations through 1×1 convolutions.

The conclusions from Chollet (2017) observations was that one could improve beyond the performance of `Inception-ResNet` by simply using only stacked depthwise-separable convolutions in place of Inception modules. An empirical study using a new type of architecture that employed just that suggestion, called the `Xception` network (Chollet, 2017), managed to achieve better top-1 error, using the same number of parameters of the `Inception-V3` network. The `Xception` network, shown in Figure 3.4, uses the same building blocks of the successful architectures that came before, such as stacked layers (`VGG-16`), skip connections, and batch normalisation (`ResNet`). The architecture uses the ReLU activation function throughout (Agarap, 2018) and makes use of MaxPooling and Global Average Pooling to remove invariances and help control the scale of the outputs. MaxPooling applies a maximum filter in the same way that a typical kernel would be applied to the input, whereas Global Average Pooling averages out the spatial information helping with translation invariance (Lin et al., 2013).

3.3 Efficient Learning with the Depthwise-Separable Convolution

While the theoretical grounding for using the depthwise-separable convolutions has been shown in Section 3.2.1, there is also a practical advantage that comes with using this operation.

This section highlights the computational efficiency gains one can achieve when switching to use of a depthwise-separable convolution in exchange of normal convolutions.

3.3.1 1D Depthwise-Separable Convolutions

The depthwise-separable convolution is in essence two operations chained together to yield a low-rank factorisation of the normal convolution (see Figure 2.4 for a visualisation of a normal convolution). This factorised version of the normal convolution first does a depthwise convolution across the channels of the input (depicted in Figure 3.5a), followed by a pointwise convolution (shown in Figure 3.5b). In the computer vision literature, a pointwise convolution is often referred to as a 1×1 convolution whereby the convolution acts on a window of height and width equal to one, and where the number of channels for which it operates is implicit. This is equivalently described as $1 \times 1 \times M$ convolution where M is the number of channels the pointwise convolution is operating on. Since we are dealing with 1-dimensional signals of time-series data with M channels, the pointwise convolutions is then described here as $1 \times M$, which indicates a window size of width 1, convolving across the M channels. The combination of Figure 3.5a and Figure 3.5b is what forms the depthwise-separable convolution, as shown in Figure 3.5.

To better understand the operations involved, we shall recap some of the properties of the normal convolution and then walk through the equations for depthwise and pointwise convolutions. Recall a convolution is mathematically described as:

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau, \tag{3.1}$$

where $f, g : \mathbb{R}^d \to \mathbb{R}$.

This describes a *mixing* of the two signals f and g, where the function $f(\boldsymbol{\tau})$ is weighted by the function $g(-\boldsymbol{\tau})$ shifted by t. For each index of $\mathbf{t}$, the weighting function $g(\mathbf{t} - \boldsymbol{\tau})$ indicates the importance of different parts of the input function, $f(\boldsymbol{\tau})$. As we are dealing with discrete signals sampled over time, the above equation becomes:

$$(f * g)[q] = \sum_{p} f[p]g[q-p], \tag{3.2}$$

where g has finite support in the set $\{1, 2, \ldots, p-1\}$

Building from Equation 3.2 we shall now put into context the nature of our problem. In simpler terms, the normal 1-dimensional convolution can be understood as a sliding dot product between a kernel (or filter) $\mathbf{K} \in \mathbb{R}^{1\times w}$ and input $\mathbf{I} \in \mathbb{R}^{M\times L}$, where w is the width of the kernel, M is the number of input channels of our signal and L is the length of our input sequence. This produces an output feature map $\mathbf{F} \in \mathbb{R}^{N\times L}$, where N is the number of output channels and L represents the length of the output[4]

$$\begin{aligned} \mathrm{Conv}_{k,n}(\mathbf{K}, \mathbf{I}) &= \sum_{i,m} \mathbf{K}_{i,m,n} \cdot \mathbf{I}_{k+i-1,m} && (3.3)\\ &= \mathbf{F}_{k,n}, && (3.4) \end{aligned}$$

where n indexes the output channels and m indexes the input channels, k indexes the kernels, and i refers to the relative position in the sequence as the kernel is moved across the input.[5].

[4]Typically the length of the output would differ as $L' = L - w + 1$, but with zero-padding and a stride of one, $L' \equiv L$, and as such will remain the notation throughout

[5]It should be noted the subtle distinction between Equation 3.2 and Equation 3.3. Instead of using the difference between indices, $k-i$, we consider $k+i$.

By inspecting Equation 3.3, it can be seen that both the channel and spatial (akin to temporal in this context) features are extracted at the same time. However, we want to be able to learn the channel cross-correlations separate to the spatial correlations as mentioned in Section 3.2.1, where the depthwise-separable comes to shine.

We mentioned previously that the depthwise-separable convolution is formed from successive operation of a depthwise convolution (DConv) and a pointwise convolution (PConv). Assuming a stride of one and zero-padding, we can write the depthwise convolution below:

$$\text{DConv}_{m,l}\left(\mathbf{K}, \mathbf{I}\right) = \sum_{i} \mathbf{K}_{m,i} \odot \mathbf{I}_{m,l+i-1}, \tag{3.5}$$

where the $\odot$ operator represents the element-wise multiplication (Hadamard product), which produces the intermediate output $\text{DConv} \in \mathbb{R}^{M \times L}$, with M representing the number of channels, and L representing indexing the length of our sequence.

The depthwise convolution separates the input into its individual channels m and performs a normal 1-dimensional convolution on each one separately. In effect, this collects features from each channel, called depthwise features. The number of kernels is equal to the number of channels in the input M.

The pointwise convolution on the other hand is simply a normal convolution with k kernels of window size equal to 1 that collects features at each index l in the sequence across the channels. With the same assumption of stride and padding as before, the output for PConv can be written as:

$$\text{PConv}_{k,l}\left(\mathbf{K}, \mathbf{I}\right) = \sum_{i} \mathbf{K}_{k,m} \cdot \mathbf{I}_{m,l+i-1}. \tag{3.6}$$

When operating in this way, the equation becomes the cross-correlation equation. Deep learning frameworks, such as `TensorFlow` (Martín Abadi et al., 2015), often implement the convolution as cross-correlation as there is little practical difference when kernels are learnable or symmetric.

In this instance $\mathbf{K}_{PW} \in \mathbb{R}^{M\times 1}$, compared to the kernel in Equation (3.5) which has size $\mathbf{K}_{DW} \in \mathbb{R}^{1\times w}$, where as before m indexes the input channels, k indexes the kernels used, and i referring to the relative position in the sequence as the kernel is moved across the input, and l indexes the sequence.

Combining Equation (3.5) and Equation (3.6) together for the complete depthwise-separable convolution (SepConv) gives an operations described as follows:

$$\text{SepConv}(\mathbf{K}_{PW}, \mathbf{K}_{DW}, \mathbf{I}) = \text{PConv}\left(\mathbf{K}_{PW}, \text{DConv}\left(\mathbf{K}_{DW}, \mathbf{I}\right)\right). \quad (3.7)$$

3.3.2 Improved Computational Complexity

In addition to the ability to decouple spatial and channel correlations, one of the benefits when using depthwise-separable convolutions in place of normal convolutions is the saving in space and computational cost.

For normal convolutions, computation of a kernel with the number of channels in the input occurs at each position of the input sequence. This is repeated for the number of channels in the output. Therefore, the number of computations becomes:

$$\mathcal{O}(w \cdot M \cdot N \cdot L), \quad (3.8)$$

where the computational cost depends on the number of input channels M, the number of output channels N, the kernel size w and the input, or feature map, size L.

In contrast, the first component of the depthwise-separable convolution, the depthwise convolution, has a computational cost of:

$$\mathcal{O}(w \cdot M \cdot L). \quad (3.9)$$

While depthwise convolutions are clearly more efficient, their use alone does not allow for the rich feature extractions one hopes for when using convolutions as it only filters the input channels, but does not combine them. Hence, the additional step of using a

pointwise convolution, to form a linear combination of the output of the depthwise layer, is required to generate new features. The computation for the pointwise convolution is simply the number of desired output channels, N, with the output sequence length, L:

$$\mathcal{O}(N \cdot L). \tag{3.10}$$

The combination of depthwise convolution and pointwise convolution is referred to as the depthwise-separable convolution, originally proposed by Sifre and Mallat (2014), but made popular with the work by Chollet (2017) with the `Xception` architecture, results in a full computation cost of:

$$\mathcal{O}(w \cdot M \cdot L + M \cdot N \cdot L). \tag{3.11}$$

By comparing Equation 3.8 and Equation 3.11, it can be seen there is a computational saving of the order of:

$$\mathcal{O}\left(\frac{1}{N} + \frac{1}{w}\right). \tag{3.12}$$

Concretely, the SepConv operation can improve efficiency when using larger number of output channels and size of kernel compared to the normal Conv operation. This observation was empirically shown by Chollet (2017), and emphasised in the work of Howard et al. (2017, `MobileNetV1`), that significant computational cost savings can be achieved.

Further to the computational savings, is the reduction in model size and space saving when using SepConv. Also shown in Chollet (2017) and highlighted in Kaiser et al. (2017) is the difference in number of parameters each operation uses. For typical convolutions, as described in Equation 3.3, the number of parameters for each layer can be expressed as:

$$\mathcal{O}\left([w \cdot M \cdot N] + N\right), \tag{3.13}$$

where M is the number of input channels, w refers to the kernel window size, and N is the number of output channels.

If we ignore the addition of N at the end in the equation above for now[6], which refers to the bias term, and consider the number of input and output channels to be the same, which we will label as d for the general dimensionality we are dealing with, Equation 3.13 then becomes:

$$\mathcal{O}\left(w \cdot d^2\right). \tag{3.14}$$

Comparatively, the depthwise-separable convolution parameters can be expressed with:

$$\mathcal{O}(w \cdot d + d^2), \tag{3.15}$$

where we again ignore the bias term and consider the number of input and output channels to be the same.

It can then also be seen that one can achieve considerable space savings with the reduction of parameters, as well as with the computational cost savings described earlier. Hence, by leveraging depthwise-separable convolutions in place of normal convolutions we can reduce model size and improve runtimes for inference and training. In recent times, the demand for deploying neural networks in resource constrained settings has exploded, and as such the desire to leverage efficient operations is one that saw pointwise convolutions, described in Section 3.3.1, used extensively in modern CNNs, as well as the depthwise convolution for decoupled spatial and channel correlations to be found. These reasons are the main motivations for using such operations for photometric classification, working towards enabling real-time science.

3.4 atx: The Astronomical-Transient Xception

The Inception hypothesis described in Section 3.2.1 revealed a path to improved performance by replacing the Inception module

[6]Discussion as to why this is ignored follows in Section 3.4.4

with stacked depthwise-separable convolutions instead. Extending Inception-*like* architectures in this way allow for improved computational efficiency at a reduction in model size. The current state-of-the-art for multivariate time-series classification by Fawaz et al. (2020, `InceptionTime`) presents an Inception style architecture adapted for use on multivariate time-series. We propose a novel architecture, called the astronomical-transient xception, or `atx`, that combines the insight by Chollet (2017) to the domain of multivariate time-series classification.

With our focus on photometric classification, we develop a variant of the `Xception` architecture with particular modifications to work with astronomical data. However, by virtue of developing the architecture for photometric data, we have also made it generally applicable to multivariate time-series data. The astronomical-transients xception works with raw time-series data, and also allows for additional features.

3.4.1 Architecture

The astronomical-transient xception, thereafter referred to as `atx`, is shown in Figure 3.6. It has the same overall structure as that presented in Chollet (2017) (see Figure 3.4) but with principled modifications made to work well for time-series data, and more specifically for photometric data.

We use a 1-dimensional depthwise-separable convolution, instead of the original 2-dimensional depthwise-separable convolution (where spatial correlations now refer to the width of the kernel used instead of height and width). To avoid overfitting, we scale the number of filters down by a factor of 4 and do not repeat the *Middle-Flow*. An additional layer for interpolating the irregularly sampled photometric data is included in our network, as well as an optional input layer for concatenating arbitrary additional features to supplement classification scores.

As with the architecture described in Chollet (2017), all normal convolution and all separable convolution layers are followed by a

batch normalization unit (BatchNorm) (Ioffe and Szegedy, 2015). In this context, a batch refers to a sub-set of the input data that is being operated on at one time, where the batch size is set by the amount of available memory on the device being used for training. This brings an additional set of parameters to the model in the way of two trainable parameters, namely a learned scaling factor, γ, and a learned offset factor, β, but also two non-trainable parameters, the moving mean and the variance, that are persisted but not updated via backpropagation (discussed further in Section 3.4.4).

3.4.2 Data Interpolation with Gaussian Processes

The raw time-series that is observed is irregularly sampled with heteroskedastic errors. There exists several methods for data interpolation such as linear interpolation or using cubic splines (De Boor and De Boor, 1978) but a technique that is widely used to overcome missing data, and that can also provide uncertainty information is Gaussian process regression (Rasmussen, 2004). This technique is a popular method that has been applied to Supernovae light curves for many years, e.g Lochner et al. (2016). Gaussian processes represent distributions over functions f that when evaluated at a given point x is a random variable $f(x)$, with mean $\mathbb{E}[f(x)] = m(x)$ and covariance between two sampled observations x, x' as $\text{Cov}(f(x, x')) = \mathbf{K}_f(x, x')$, where $\mathbf{K}_f(\cdot, \cdot)$ is a kernel.

An important aspect of applying Gaussian process interpolation to data is the choice of kernel which affects light curve fits and ultimately classification results down-the-line (Revsbech et al., 2018). The work by Lochner et al. (2016) (described in Chapter 2) used a squared-exponential kernel for fitting SNe light-curves. Their choice at the time was motivated by flexibility and ease of use, however later studies in Revsbech et al. (2018) showed the Gibbs kernel to be more suitable for SNe-like light-curves by avoiding overfitting around the peak. Extending to the general transient case, it was discovered in Boone (2019) that for general transients a 2-dimensional Matern kernel that incorporates both wavelength (*i.e.*

passband) information as well as time works well. It can be seen in Boone (2019) that by use of the 2-dimensional kernel, correlations between passbands are leveraged and predictions in passbands that do not have any observations are still possible. Therefore we note that while other specialised kernels may be better suited for individual classes, we use the same Matern kernel (Rasmussen, 2004) shown here,

$$\mathbf{K}_{\nu=3/2}(x, x') = \left(1 + \frac{\sqrt{3}r}{\boldsymbol{\ell}}\right) \exp\left(\frac{\sqrt{3}r}{\boldsymbol{\ell}}\right), \tag{3.16}$$

where r is the Euclidean distance between x and x', and $\boldsymbol{\ell}$ is a 2-dimensional vector for the length scale of time and wavelength. It is parametrised by ν which controls the smoothness of the resulting function and set to $\nu = 3/2$. Note as $\nu \to \infty$ the kernel becomes equivalent to the kernel defined in Equation 2.2. By performing Gaussian process regression and then sampling the resulting Gaussian process at regular intervals, we transform our previously irregular multivariate time-series to a now well sampled regular multivariate signal. The Gaussian process mean is sampled at regular points in time to produce $\mathbf{X} \in \mathbb{R}^{L \times M}$, where L is the sampled time sequence length and M is the number of passbands. This procedure is illustrated in Figure 3.7. It should be stated that by virtue of using Gaussian processes for data interpolation the quality of the interpolated signal is highly depended on the choice of kernel, which has the possibility to produce unphysical results in regions of no data. It could be helpful to study the affects of model predictions in comparison to other interpolation techniques mentioned above such as linear interpolation and cubic splines. Furthermore, while we do not use the Gaussian processes' errors explicitly as input, a study that directly includes the Gaussian process error as input would be worth investigating.

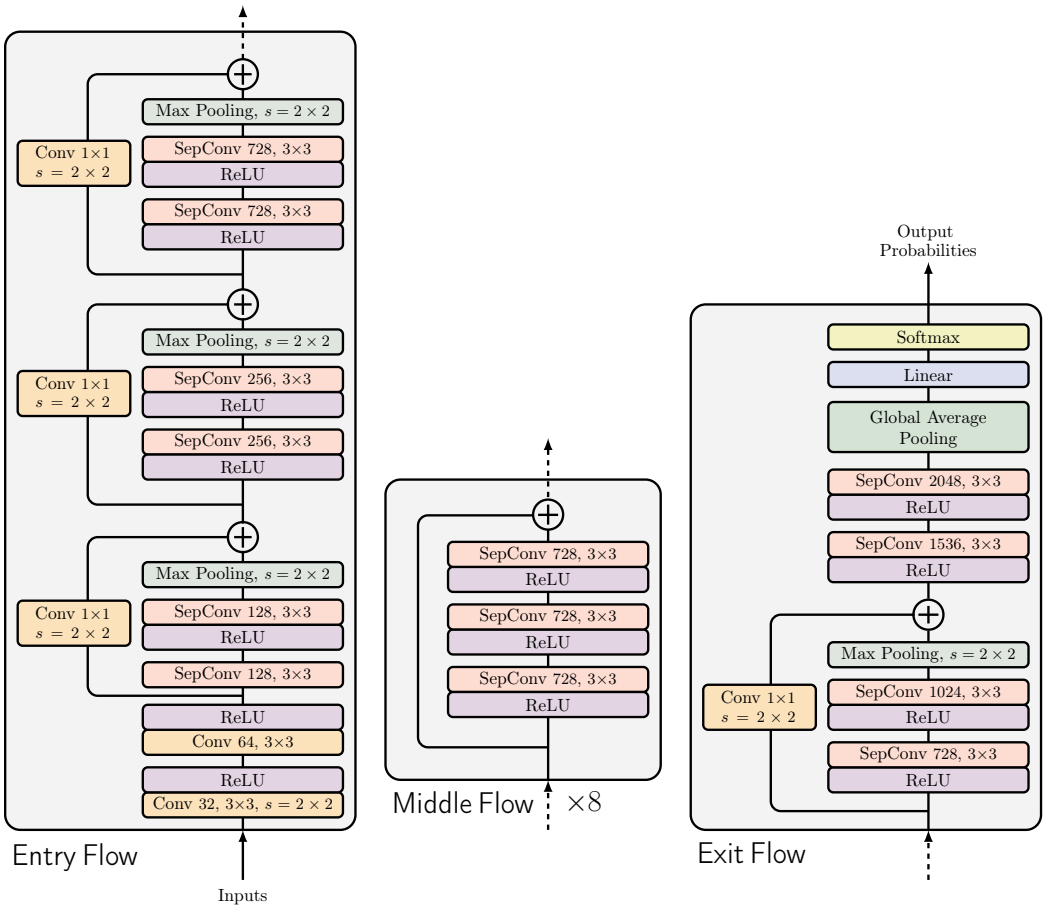
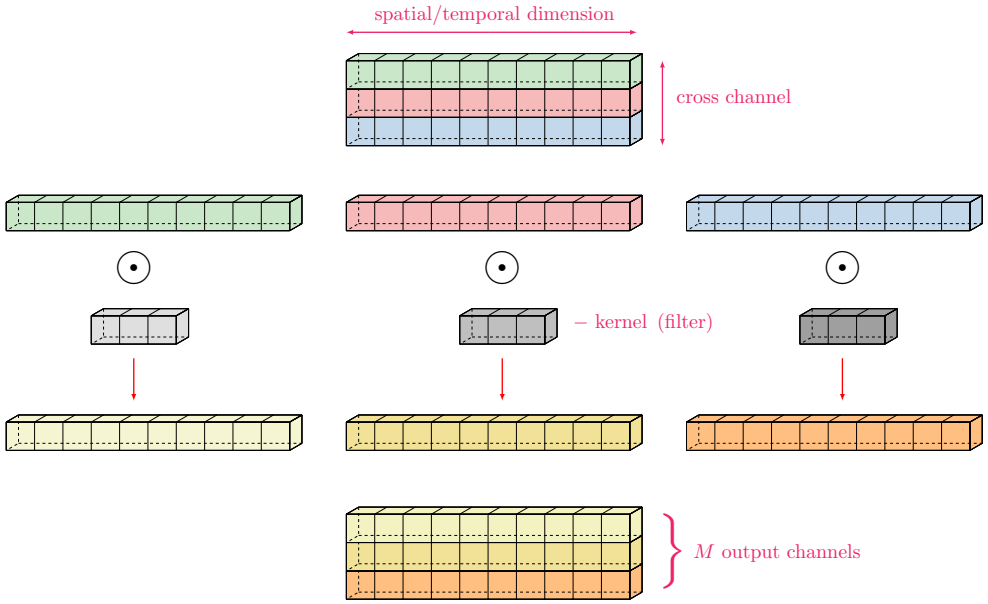
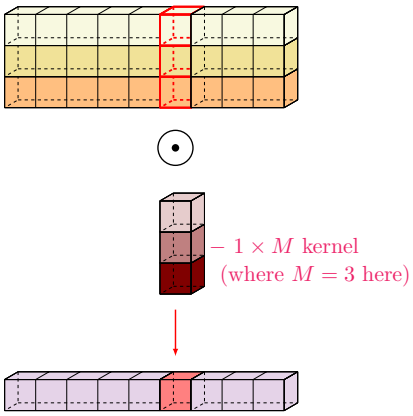


FIGURE 3.4: Arrangement of the original xception architecture presented in Chollet (2017). Where applicable, the notation for convolutions operations is given as the number of filters, the kernel size, followed by the stride length s. For the looped 1×1 convolution connections, the number of filters is equal to the number of filters of the outgoing separable-convolution it is being connected to. Note, the *Middle Flow* is repeated 8 times, with the output feeding back into itself. As referred to in Chollet (2017), all convolution layers are followed by batch normalisation (Ioffe and Szegedy, 2015), which is not shown in the diagram.



(a) Depthwise convolution.



(b) Pointwise convolution.

FIGURE 3.5: Depthwise-separable convolution. Chaining a and b gives the depthwise-separable convolution. The number of filters in the depthwise convolution is fixed by the number of channels M in the input, as is the size of the pointwise convolution *i.e.* $1 \times M$. Repeatedly applying the depthwise-separable convolution to an input simply adds new output channels, allowing for dimensionality to scale as $M \to N$, where N is the number of times it is applied.

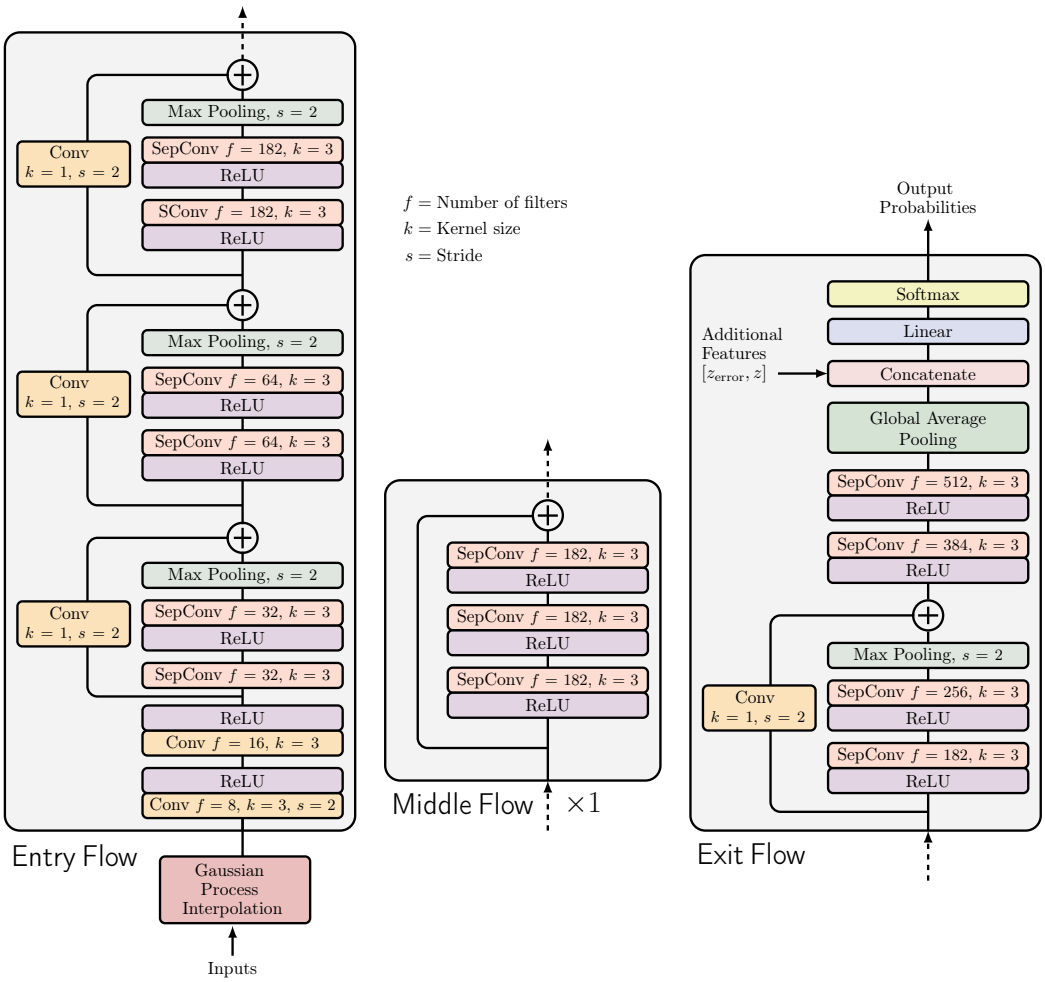


FIGURE 3.6: Schematic of the astronomical-xception (`atx`) architecture. Photometric time-series data is processed through the Gaussian process interpolation layer, before being passed into the *Entry-Flow*. From this point, the overall architecture is akin to that described in Section 3.3 and depicted in Figure 3.4 with the key difference being the use of a 1-dimensional depthwise-separable convolution, compared to the 2-dimensional depthwise-separable convolution used in the original work by Chollet (2017). Furthermore, to avoid overfitting, the number of filters has been scaled down by a factor of 4, as well as reducing the number of times the *Middle Flow* section is repeated. We include a new concatenation layer in the *Exit Flow* block where one can include an arbitrary number of additional features, such as redshift information. A final linear layer with softmax is applied to output class prediction probabilities for the objects.

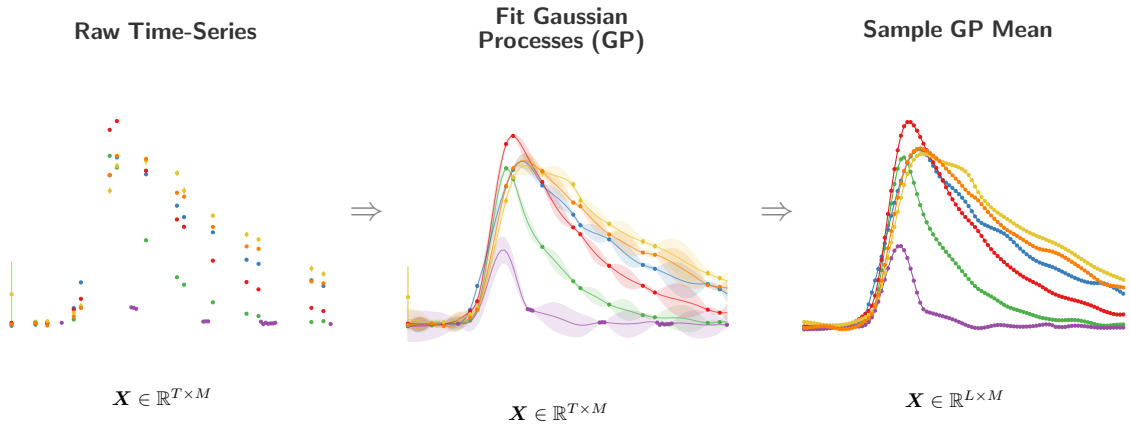


FIGURE 3.7: Process of transforming an irregularly sampled transient light curve to a well sampled multivariate time-series using Gaussian process interpolation. First, we fit a 2-dimensional Gaussian process using passband wavelength and time information. Then we evaluate the Gaussian process mean at L time points l for each of M passbands, to give a well sampled multivariate time-series.

3.4.3 Inputting Additional Information

The astronomical-transient xception, `atx`, is designed to be malleable such that one can add further features if desired. We achieve this by adding a new input layer to the Xception architecture (see. Exit Flow in Figure 3.6) that allows for additional features to be concatenated to the features that emerge from the network, but before the final linear layer. We chose to input additional features here to allow for the rest of the architecture to build rich time-series only features. It was felt best to clearly separate time-series information from auxiliary features and allow for the 1-dimensional temporal operations to find patterns within the multivariate signal alone. Having said that, there is scope to include features at any point in the network and would be of interest for further study to investigate the impact on performance should they be included elsewhere. For the purposes of the current study of photometric classification, only redshift information has been added. In many photometric classifiers, photometric redshift z, has consistently been a feature of high importance (*e.g.* Boone, 2019). As one particular example of the type of additional features that can be added, photometric redshift z and the associated error z_{error} are included.

3.4.4 Trainable Parameters and Hyperparameters

The `atx` architecture that we propose consists of a set of trainable parameters, some of which were described in Section 3.3.2, as well as tunable hyperparameters which ultimately control the performance of the network. Of the layers in `atx`, it is only the SepConv and Conv blocks, as well as the final linear layer that contain the trainable parameters since there are no trainable parameters in the MaxPool, ReLU or Softmax layers.

Recall that for the original `Xception` architecture, following each depthwise-separate and normal convolutions is a batch normalization layer (BatchNorm). BatchNorm applies a transformation to ensure the mean of each batch is near zero and that the standard deviation centres around one. As described in Section 3.4.1,

applying BatchNorm adds four further parameters, where only two are trainable γ and β. The other two non-trainable parameters are the moving average and moving variance. As described in Ioffe and Szegedy (2015), when applying BatchNorm, the role of the bias is subsumed by the learned offset factor, β, so it does not feature as a parameter of the network.

Considering this, the two operations then become:

$$\text{SepConv}_{\#\text{params}} = \underbrace{\left[w \times d + d^2\right]}_{\text{DConv + PConv}} + \underbrace{\left[\gamma + \beta + \mu + \sigma^2\right]}_{\text{BatchNorm}} \tag{3.17}$$

$$\text{Conv}_{\#\text{params}} = \left[w \times d^2\right] + \left[\gamma + \beta + \mu + \sigma^2\right] \tag{3.18}$$

Lastly, the final fully connected layer, which *does* contain bias terms is given as:

$$\text{Linear}_{\#\text{params}} = \left([d \times C] + C\right), \tag{3.19}$$

where C refers to the number of classes.

From the above equations, there are certain parameters that are set by way of our problem, *i.e.* number of classes, C. Yet other parameters, that may not necessarily be trainable parameters but affect the overall performance of the network, need to be set. One such *hyperparameter* is choice of kernel window size, w, which controls the amount of temporal correlations to consider at a time. Another is the pooling window size, `pool_size`, which dictates the window to perform max-pooling over. A hyperparameter we introduce specifically for our architecture is called the `scaledown_factor`.

Since `atx` is constructed from the same setup as the `Xception` network, we use the same number of filters for each layer in the network, however, as `Xception` was designed for the task of image classification trained on extremely large datasets, the number of filters used originally would cause overfitting for our problem of photometric classification. Thus, we introduce a new hyperparameter which scales the original `Xception` down by a certain factor,

reducing the number of filters at each layer simultaneously, and hence the overall number of parameters. Further still, to reduce overfitting even more, we present another hyperparameter that controls number of times the *Middle-Flow* is repeated. In the original work by Chollet (2017) this was repeated 8 times but, as previously stated, it is expected that this would be too much for our problem, and so we include a hyperparameter, N, to learn the optimal number of repeats of this section. Lastly, a hyperparameter that controls the learning rate, as described by η in Equation 2.3, but referred to as `learning_rate` hereafter, is used for training.

3.5 Implementation, Evaluation Metrics & Training

In order to develop and evaluate our architecture, modern machine learning frameworks were used that allow for a modular implementation, permitting easy extensions or modifications in the future. This section explains the choice of loss function that is used for training is discussed as well as how model hyperparameters are optimised.

3.5.1 Implementation

The `atx` architecture described in this chapter has been implemented using the machine learning framework `TensorFlow` (Martín Abadi et al., 2015) and the `tf.keras` application programming interface (API) (Chollet et al., 2015), with training and inference carried out on a NVIDIA Tesla V100 GPU. Other essential software used for data processing includes `pandas` (McKinney, 2010), `numpy` (Harris et al., 2020) and `george` (Ambikasaran et al., 2015) for fitting the Gaussian processes. The code is open-sourced and available under Apache 2.0 licence[7].

[7] github.com/tallamjr/astronet

3.5.2 Multi-Class Logarithmic-Loss

The underlying algorithm that governs the usefulness of neural networks is the stochastic gradient decent (SGD) optimisation algorithm that updates the weights of the network according to the backpropagation algorithm (Rumelhart et al., 1986). While performance metrics give an indicator as to how well a model is able to distinguish between classes, to be able to train and improve the model one must have a differentiable loss function. Extensive investigations by Malz et al. (2019b) showed that the most suitable differentiable loss-function for the problem of transients classification is a probabilistic loss function. Probabilistic loss functions are used in cases where the uncertainty of a prediction is useful and the problem at hand is best served with quantification of the errors rather than a binary answer of correct or incorrect. The probabilistic loss function they suggest is the multi-class weighted logarithmic-loss that up-weights rarer classes and defines a perfect classifier as one that achieves a score of zero, and is given by

$$\boldsymbol{\mathcal{L}} = -\left(\frac{\sum_{i=1}^{C} w_i \sum_{j=1}^{N_i} \frac{y_{ij}}{N_i} \ln p_{ij}}{\sum_{i=1}^{C} w_i} \right), \tag{3.20}$$

where C refers to the number of classes in the dataset and N_i the number of objects in the i-th class. The predicted probability of an observation i belonging to class j is given by p_{ij} with a truth label y_{ij} equal 1 if the object j belongs to class i and 0 otherwise. Finally the class weight w_i is used to bolster performance to specific classes. For our investigation we opt for a flat-weighted multi-class logarithmic-loss as described in Boone (2019) that assigns all classes in the training set the same weight of $w_i = 1$. To consider the original metric put forth in Malz et al. (2019b) and use the weighting scheme designed for the PLAsTiCC competition, one would also need to include the additional anomaly classes (class 99) that existed in the PLAsTiCC test set. By ignoring class 99 one can better compare later analyses between the original PLAsTiCC

training set and our modified dataset (described in upcoming Section 3.6.1).

3.5.3 Training

In order to train a model with the `atx` architecture, we need to establish the choice of optimisation algorithm and associated parameters that will be used to update the weights of the network. We use a variant of the SGD optimisation algorithm mentioned in Section 3.5.2 called ADAM (Kingma and Ba, 2014). An important aspect to consider when training a model using any optimisation algorithm is the learning schedule and corresponding learning rate. The initialisation value of the learning rate can be seen as a hyperparameter to be optimised for separately with hyperparameter optimisation (discussed in the next section). It is typically beneficial to introduce a learning schedule to reduce the learning rate as training progresses (Goodfellow et al., 2016). We indeed adopt a learning schedule, reducing the learning rate by 10% if it is observed that our loss value does not decrease within 5 epochs, where a single epoch refers to one full forward pass and one full backward pass of all the examples in the training set. To ensure the model does not overfit, the ratio of validation loss with the training set loss is monitored.

3.5.4 Hyperparameter Optimisation

As discussed in Section 3.4.4, `atx` contains a set of fixed parameters such as M and C, and a set of tunable hyperparameters. Choosing the best set of hyperparameters can be framed as an optimisation problem expressed as

$$\theta^* = \arg\min_{\theta \in \Theta} g(\theta), \tag{3.21}$$

where $g(\theta)$ is an objective score to be minimised and evaluated on a validation set, with the set of hyperparameters θ being able to take any value defined in the domain of Θ. The objective score for

our purposes is the logarithmic-loss defined in Equation 3.20 and the set of hyperparameters that yield the lowest objective score is θ^*. The goal is to find the model hyperparameters that yield the best score on the validation set metric (Koehrsen, 2018).

Traditionally hyperparameter optimisation has been performed with either random search or a grid search over the set of parameters in Θ, which can be time consuming and inefficient. Instead a Bayesian optimisation approach is used that attempts to form a probabilistic model mapping hyperparameters to a probability distribution for a given score.

To choose the best performing hyperparameters we use the Tree-structured Parzen Estimator (TPE) algorithm (Bergstra et al., 2011) that is implemented in the `optuna` package (Akiba et al., 2019) with 5 fold cross-validation.

3.6 Results

We apply our astronomical-transient xception architecture to the problem of photometric classification of astronomical transients. In order to also gauge the network's versatility towards general multivariate time-series data we apply our new architecture to the MTS (described in Section 2.4) too. However, as the focus of this thesis is in relation to photometric classification we refer the reader to Appendix A for the MTS results. As noted in Section 3.1 typical astronomical data that are available for training a photometric classifier are highly imbalanced, with a large number of spectroscopically confirmed SNIa compared to other classes, and non-representative, since observations are biased towards lower redshift objects. Consequently, the training data are non-representative of the test data. For robust and accurate classification, training datasets should be representative of the test data. Works by Revsbech et al. (2018), Boone (2019) and Alves et al. (2022a) present

techniques that help address this problem of non-representativity, transforming the training data to be more representative of the true test data through data augmentation. This process is involved and can be decoupled from the design of architecture of the classifier. Therefore in this current work, as a first step we consider training data that is representative in redshift but imbalanced. In future work we will consider the combination of `atx` with augmentation techniques to address the representativity problem.

3.6.1 Astronomical Transients Dataset

To be able to evaluate our architecture in a representative setting, but also to test the model's resilience to class imbalance, we utilise the PLAsTiCC dataset (The PLAsTiCC team et al., 2018b). The complete dataset contains synthetic light curves of approximately 3.5 million transient objects from a variety of classes simulated to be observed in 6 passbands using a cadence defined in Kessler et al. (2019).

The majority of events that exist in the dataset were simulated to be observed with the Wide-Fast-Deep (WFD) mode, which compared to the Deep-Drilling-Fields (DDF) observing mode, is more sparsely sampled in time and has larger errors. Originally crafted for a machine learning competition[8], the entire PLAsTiCC dataset was divided into two parts, with $< 1\%$ initially being given to participants in the competition that was highly non-representative of the other part. Following the close of the competition all data are now publicly available[9]. For our purposes, we use the complement to what was initially released and construct a new training and test set from the remaining 99% of the data (without anomaly class 99), where 70% of the data is used for training and the remaining 30% is used for a disjoint test set. By doing so, the dataset is now representative in terms of redshift, but remains highly imbalanced in terms of the classes. The number of samples per class used to eval-

[8] kaggle.com/c/PLAsTiCC-2018

TABLE 3.1: Number of samples of the PLAsTiCC data used for evaluation of the `atx` model. Note the largely imbalanced dataset distribution of SNIa objects compared to other classes.

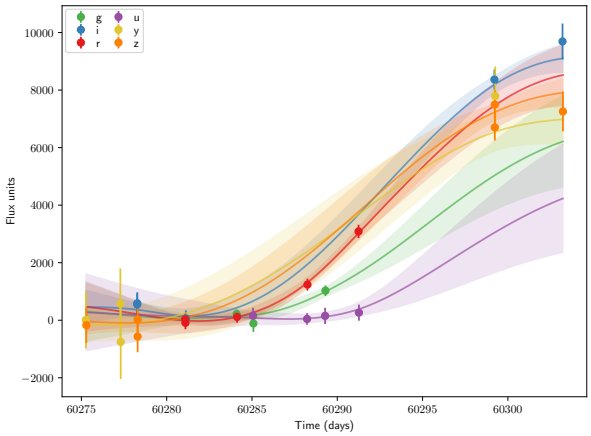
Class	Number of Samples (%)
μ − Lens-Single	1,303 (0.037%)
TDE	13,552 (0.389%)
EB	96,560 (2.775%)
SNII	1,000,033 (28.741%)
SNIax	63,660 (1.830%)
Mira	1,453 (0.042%)
SNIbc	175,083 (5.032%)
KN	132 (0.004%)
M-dwarf	93,480 (2.686%)
SNIa-91bg	40,192 (1.155%)
AGN	101,412 (2.915%)
SNIa	1,659,684 (47.700%)
RRL	197,131 (5.666%)
SLSN-I	35,780 (1.028%)
Total	3,479,456 (100%)

uate our architecture can be found in Table 3.1. We present some example light-curves (that have been pre-processed using the Gaussian Process interpolation method described in Section 3.4.2) here in Figure 3.8 for an illustrative overview of the data. For further examples please see figures in Appendix B.

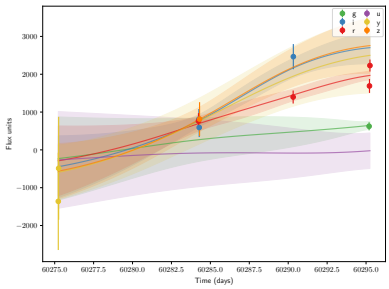
3.6.2 Classification Performance

Several hyperparameters of the astronomical-transient xception, `atx`, need to be optimised as discussed in Section 3.5.4. By applying the TPE Bayesian optimisation method to the validation data, the hyperparameters shown in Table 3.2 are found. Due to

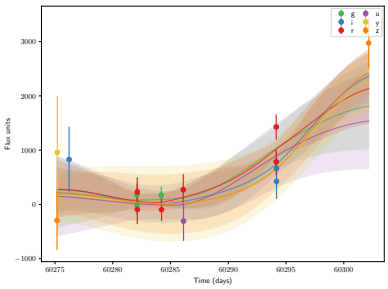
[9] zenodo.org/record/2539456#.YIiVA5NKjlz



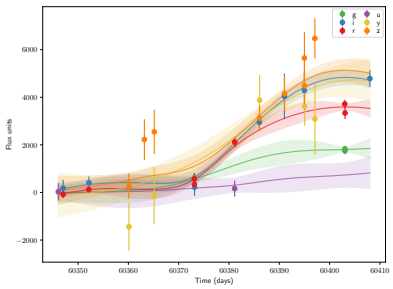
(a) SNIa



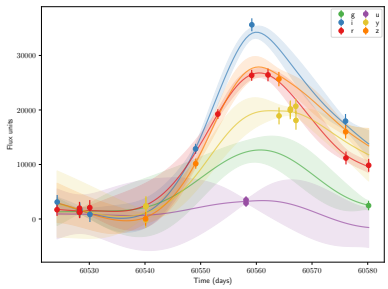
(b) SNIbc



(c) SNII



(d) SNIax



(e) SNIa-91bg

FIGURE 3.8: Example Supernovae light curves of varying types compared to SNIa shown in a showing similarities at the rise and overall light curve profile.

TABLE 3.2: The astronomical-transient xception, `atx`, contains 5 hyperparameters to be optimised. Using the technique of TPE Bayesian optimisation, the set of parameters and learning rate initialisation that yield the lowest score with 5-fold cross validation on 10% of the training data are shown here. Note, `learning_rate` shown to 3 decimal places for brevity. See code for exact details.

Parameter	Value
w	3
N	1
`pool_size`	3
`learning_rate`	0.018
`scaledown_factor`	4

computational constraints, 5-fold cross-validation was done using what equates to only 10% of the full training set.

Using the hyperparameters defined in Table 3.2 the `atx` architecture is able to achieve a score of 0.739 on the logarithmic-loss metric on the test set (derived from the PLAsTiCC dataset discussed in Section 3.6.1). This is in comparison to the state-of-the-art results of Boone (2019) which achieved a flat weighted-logarithmic loss of 0.468. It can be seen from the confusion matrix of Figure 3.10 that the performance over all classes is good even with highly imbalanced data. This is also seen in the ROC curve of Figure 3.11, which yields a micro-averaged and macro-averaged AUC of 0.98. With regards to the precision-recall trade-off, shown in Figure 3.12, `atx` is able to achieve micro-averaged AUC of 0.81. Though, understandably, the model struggles to obtain high precision for Kilonovae events due to the extremely low number of samples in the training set (see Table 3.1). Performance on SNIax is also relatively low; this may be due to miss-classification with Supernova Type Ia, which are known to be very similar near maximum light (Jha, 2017) and can seen in Figure 3.8. The SN types of SNIbc, SNII, SNIax and SNIa-91bg, shown in 3.8b, 3.8c, 3.8d

and 3.8e respectively can be seen to share similarities around the rise and peak and is a major cause of cross contamination when classifying SNIa. Our SNIa purity is 0.92 with a core-collapse SNe (SNe Ib/c and SNe II) cross contamination of $\sim 6.68\%$. With studies from DES (Vincenzi et al., 2021) and Pan-STARRS (Jones et al., 2018) reporting an acceptable range for cross-contamination to be $\sim 8\%$ and $\sim 5\%$ respectively, our results are also within the bounds for cosmological analyses of dark energy equation of state. We also repeated the analysis without the additional features of redshift and the associated redshift error. Unsurprisingly, the level of performance achieve reduced, resulting in a logarithmic-loss of 0.929.

The astronomical-transient xception, `atx,` achieves good classification performance, close to other leading methods such as those laid out in Boone (2019), and does so with the ability of inputting raw time-series data alone. Its shortcomings, in terms of purity of low sample classes such as Kilonovae with a precision of 0.04, would surely be alleviated with proper augmentation, yet under the circumstances, `atx` is still able to separate classes well. As such, an investigation into using augmentation presents a possible avenue for further research, should other architectures not bear improved results.

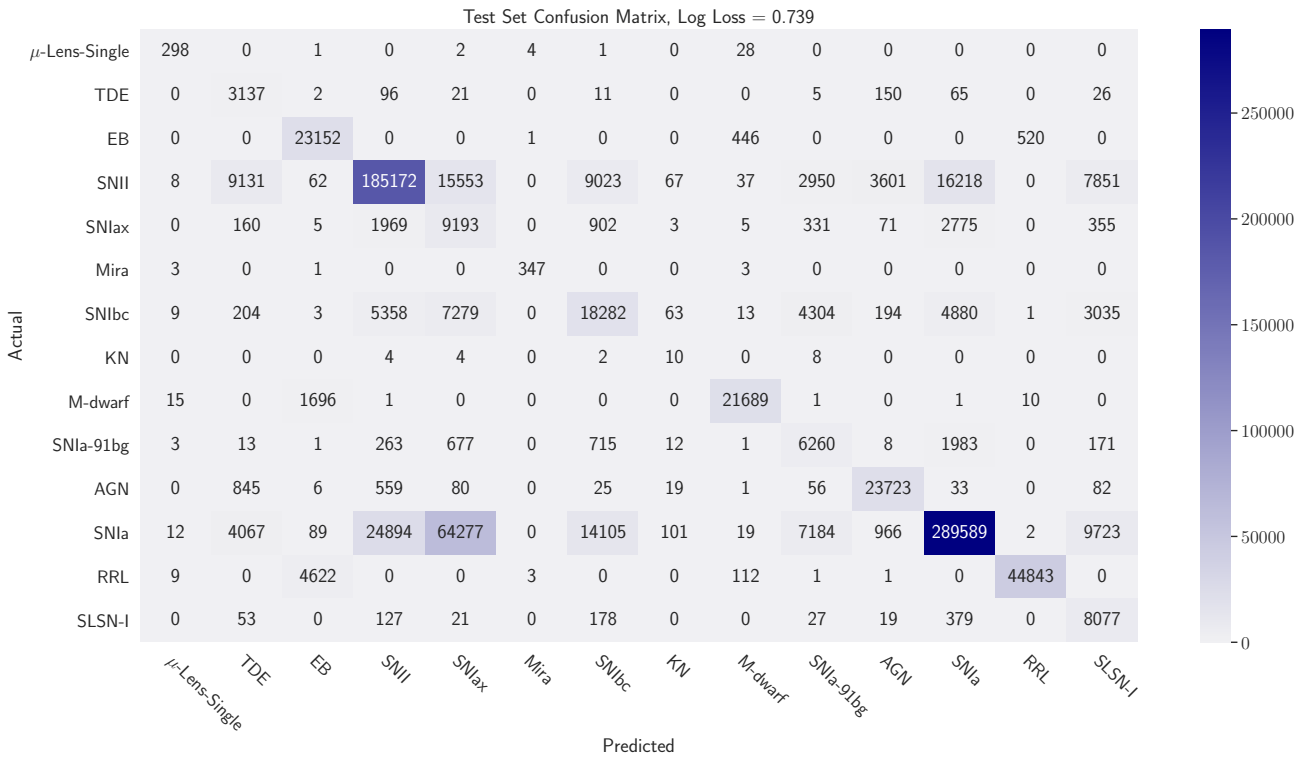


FIGURE 3.9: Raw count confusion matrix resulting from application of the astronomical-xception network, `atx`, to the PLAsTiCC dataset in a representative setting with imbalanced classes, achieving a logarithmic-loss of 0.739 and a SNIa purity of 0.92 at 6.7% core-collapse SNe cross contamination

3.7 Conclusions

This chapter has presented a novel architecture that is designed for photometric classification of astronomical transients, but that is also suitable for general multivariate time-series classification tasks. The astronomical-transient xception, also known as `atx` is able to not only input raw light curve data for good classification performance, but also can be supplemented with arbitrary additional information. When combined with redshift and redshift error, `atx`, is able to achieve scores close to the state-of-the-art in photometric classification.

The efficient nature of the depthwise-separable operation that astronomical-transient xception is built upon allow reduced parameter count when compared to other CNN approaches for photometric classification. Consequently, `atx` should reveal faster training and inference times as well which should appeal for potential deployment within alert brokering systems currently under construction such as FINK (Möller et al., 2021), ANTARES (Matheson et al., 2021) *etc.*

Through extensions of the Inception hypothesis to the world of time-series, the astronomical-transient xception's ability to work with time-series data alone and considerably fewer parameters than other networks in its class, it is hoped that `atx` can influence further use of the depthwise-separable convolutions for light curve analysis as well as move one step closer to enabling real-time science for the next generation of surveys.

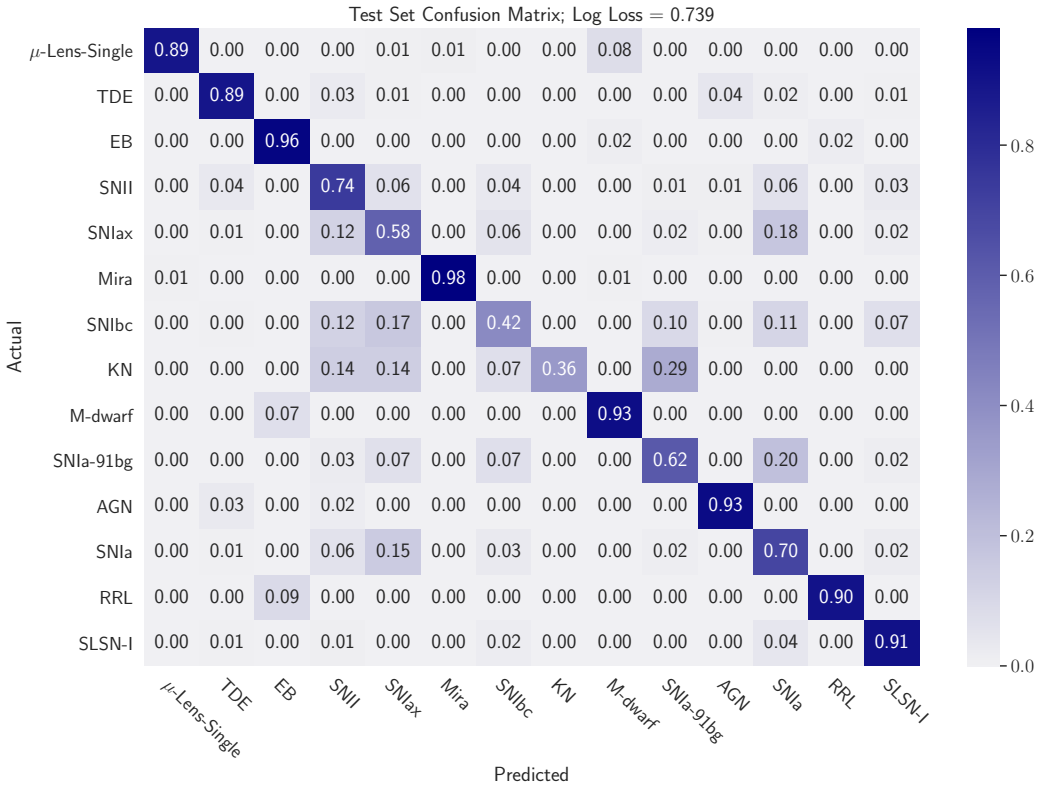


FIGURE 3.10: Normalised confusion matrix resulting from application of the astronomical-xception network, `atx`, to the PLAsTiCC dataset in a representative setting with imbalanced classes, achieving a logarithmic-loss of 0.739.

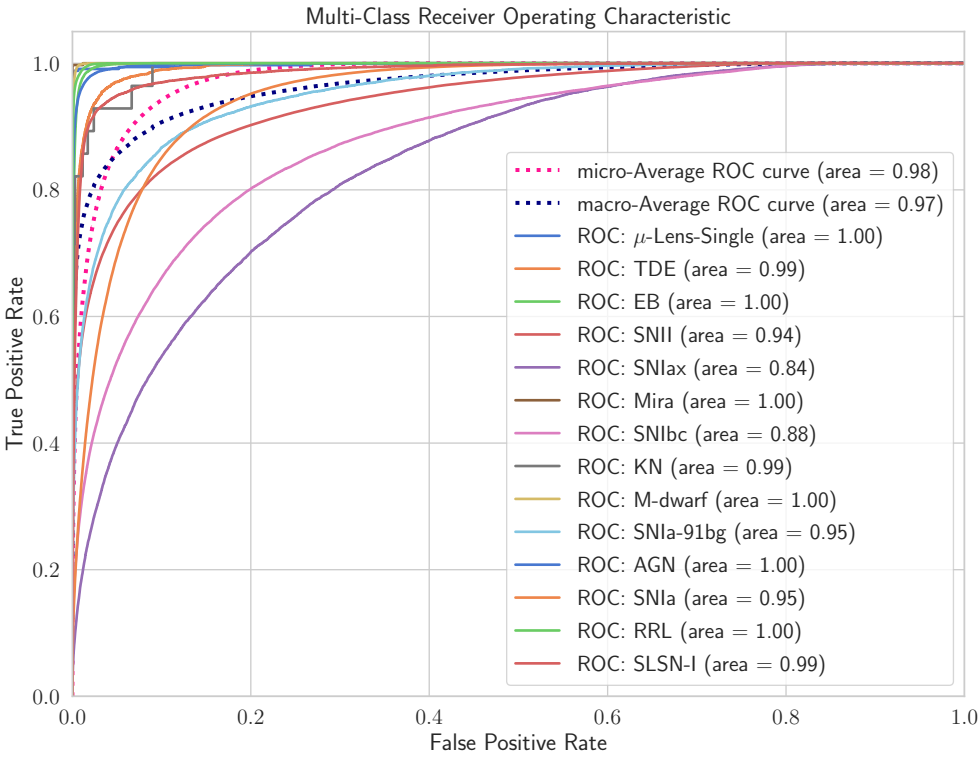


FIGURE 3.11: Receiver operating characteristic (ROC) curve, under the same setting as those described in Figure 3.10. Micro- and macro-averaged AUC scores of 0.98 and 0.97 are achieved across the classes respectively.

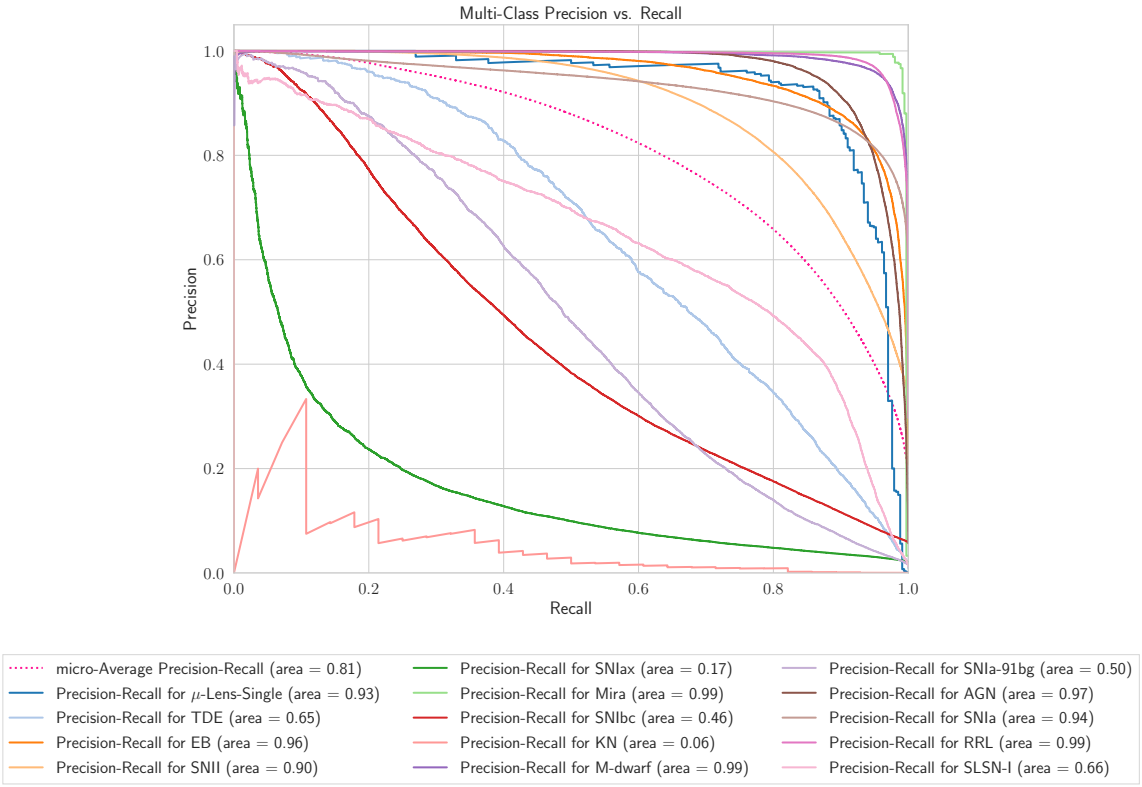


FIGURE 3.12: Precision-recall trade-off curve, under the same setting as those described in Figure 3.10. A micro-averaged AUC score of 0.81 is achieved across the classes. With only 0.004% of the training sample constituting to the Kilonovae class, our model expectedly finds it difficult to classify this extreme minority class correctly.

4

Paying Attention to Astronomical Transients: Photometric Classification with the Time-Series Transformer

In this chapter we develop a new transformer architecture, which uses multi-head self attention at its core, for general multivariate time-series data. Furthermore, the proposed time-series transformer architecture supports the inclusion of an arbitrary number of additional features, while also offering interpretability. We apply the time-series transformer to the task of photometric classification, minimising the reliance of expert domain knowledge for feature selection, while achieving results comparable to state-of-the-art photometric classification methods. We achieve a logarithmic-loss of 0.507 on imbalanced data in a representative setting using data from the Photometric LSST Astronomical Time-Series Classification Challenge (PLAsTiCC). Moreover, we achieve a micro-averaged receiver operating characteristic area under curve of 0.98 and micro-averaged precision-recall area under curve of 0.87.

4.1 Introduction

Self-attention mechanisms and the related transformer architecture, proposed by the NLP community, have been introduced to overcome the computational woes of CNNs and RNNs described in Section 2.4 (Vaswani et al., 2017). Complexity per layer is described by the sequence length L and the dimensionality d as $\mathcal{O}(L^2 \cdot d)$.

With a maximum path length of $\mathcal{O}(1)$ and embarrassingly parallelisable operations, the mechanism has revolutionised the field of sequence modelling and is at the heart of the work presented here. We develop a new transformer architecture for the classification of general multivariate time-series data, which uses a variant of the self-attention mechanism, and that we apply to the photometric classification of astronomical transients.

This chapter is structured as follows: Section 4.2 reviews the recent breakthroughs in the domain of sequence modelling and NLP that have inspired this work, and presents a pedagogical overview of transformers and the attention mechanism that overcome some of the challenges faced by RNNs and CNNs. Section 4.3 outlines the attention-based architecture of the time-series transformer developed in this work, with the goal of photometric classification of astronomical transients in mind. Section 4.4 describes the implementation and performance metrics used to evaluate models. Section 4.5 presents the results obtained from applying the transformer architecture developed to PLAsTiCC data (The PLAsTiCC team et al., 2018b). Finally, in Section 4.6 a summary of the work carried out and the key results is discussed.

4.2 Attention Is All You Need?

This section gives a pedagogical review of the attention mechanism, and specifically self-attention, which is the foundational element of our proposed architecture. We step through the original architecture that uses self-attention at its core and inspired this work, the transformer (Vaswani et al., 2017), and how it is generally used in the context of sequence modelling.

4.2.1 Attention Mechanisms

As humans, we tend to focus our *attention* when carrying out particular tasks or solving problems. The incorporation of this concept to problems in NLP has proven extremely successful, and in particular the development of the *attention mechanism* has been shown to have a major impact, not only in the world of sequence modelling, but also in computer vision and other areas of deep learning.

The attention mechanism originates from research into neural machine translation, a sub-field of sequence modelling often referred to as Seq2Seq modelling (Sutskever et al., 2014). Seq2Seq modelling, as shown in Figure 2.6 in Chapter 2, attempts to build models that take inputs represented as a sequence of embedding vectors $\boldsymbol{x} = [x_0, x_1, \ldots, x_L]$ of dimensionality d and tries to find a mapping to the target sequence $\boldsymbol{y} = [y_0, y_1, \ldots, y_L]$. In the domain of NLP, these inputs are word embeddings that are transformations of a word at a given position into a numerical vector representation for that word such as provided by the `word2vec` algorithm (Mikolov et al., 2013). In the field of photometric classification, the inputs may be a vector representation of the flux and flux error values for each passband at each time-step (discussed further in Section 4.3.2). Seq2Seq has traditionally been done by way of two RNNs that form an encoder-decoder architecture (see. Figure 2.6), with the encoder taking the input sequence $\boldsymbol{x}$ and transforming it into a fixed length context vector $\boldsymbol{c}$, and the decoder taking the context through transformations that lead to the final output sequence $\boldsymbol{y}$. The hope is that the context vector is a compressed representation of the *entire* input sequence that is able to contain all the information of the inputs such that it can passed along to the decode to help make predictions. However, trouble arises with use of RNNs due to the inherent Markov modelling property of these sequential networks, where the state is assumed to be only dependent on the previously observed state. As a consequence RNNs need to maintain memory of each input in the sequence, albeit a compressed representation, up to the desired context length. Therefore,

RNNs suffer greatly with computationally maintaining memory for large sequences (Madsen, 2019).

Attention mechanisms (Bahdanau et al., 2014) were introduced to mitigate these issues and to allow for the full encoder state to be accessible to the decoder via the context vector. This context vector is built from hidden states $\boldsymbol{h}$ of the encoder and decoder as well as an alignment score α_{ti}, between the target t and input i. This assigns a score α_{ti} to the pair (y_t, x_i), *e.g.* in neural machine translation, the word at position i in the input and the word at position t in the output, according to how well they align in vector space. It is the set of weights $\{\alpha_{ti}\}$ that define how much of each input hidden state should be considered for each output. The context vector $\boldsymbol{c}_t$ is then defined as the weighted sum of the input sequence hidden states $\boldsymbol{h}_i$, and the alignment scores α_{ti}. This can be expressed as

$$\boldsymbol{c}_t = \sum_i \alpha_{ti} \boldsymbol{h}_i. \tag{4.1}$$

A common global attention mechanism used to compute the alignments is to compare the current target hidden state $\boldsymbol{h}_t$ to each input hidden state $\boldsymbol{h}_i$, as follows (*e.g.* Luong et al., 2015):

$$\alpha_{ti} = \text{align}(\boldsymbol{h}_t, \boldsymbol{h}_i) = \frac{\exp\big(\text{score}(\boldsymbol{h}_t, \boldsymbol{h}_i)\big)}{\sum_{t'i'} \exp\big(\text{score}(\boldsymbol{h}_{t'}, \boldsymbol{h}_{i'})\big)}, \tag{4.2}$$

where score can be any similarity function. For computational convenience this is often chosen to be the dot product of the two hidden state vectors, *i.e.*

$$\text{score}(\boldsymbol{h}_t, \boldsymbol{h}_i) = \boldsymbol{h}_t \boldsymbol{h}_i^\top. \tag{4.3}$$

See Weng (2018) for a summary table of several other popular attention mechanisms and corresponding alignment score functions.

4.2.2 Self-Attention

Self-attention is an attention mechanism that compares different positions of a single input sequence to itself in order to compute a

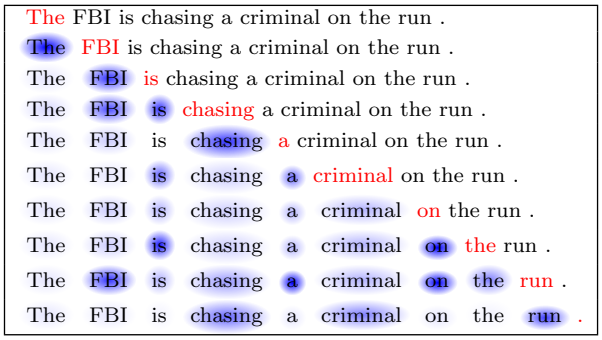


FIGURE 4.1: A model using the attention mechanism, reading the sentence: *The FBI is chasing a criminal on the run.* Blue represents the attention for the input sequence up to the end word in red. The level of shading dipicts the attention weighting for each word in the input sequence. Reproduced in full from Cheng et al. (2016).

representation of that sequence. It can make use of any similarity function, as long as the target sequence is the same as the input sequence. Prominent use of self-attention came from work in machine reading tasks where the mechanism is able to learn correlations between current words in a sentence and the words that come before (see Figure 4.1). This type of attention can thus be useful in determining the correlations of data at individual positions with data at other positions in a single input sequence.

Drawing from database and information retrieval literature, a common analogy of query $\boldsymbol{q}$, key $\boldsymbol{k}$, and value $\boldsymbol{v}$, is used when referring to the hidden states of encoder and decoder subcomponents. The query, $\boldsymbol{q}$, can been seen as the decoder's hidden state, $\boldsymbol{h}_t$, and the key $\boldsymbol{k}$, can be seen as the encoder's hidden state, $\boldsymbol{h}_i$. The similarity between the query and key can then be used to access the encoder value $\boldsymbol{v}$. In the case of dot-product self-attention, *learnable*-weights, $\mathbf{W}$, are attached to the input $\mathbf{X} \in \mathbb{R}^{L \times d}$ for sequence length L and embedding dimension d for each of $\boldsymbol{q}$, $\boldsymbol{k}$ and $\boldsymbol{v}$, which can be visualised in Figure 4.2. This results in a set of queries $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q \in \mathbb{R}^{L \times d_q}$, keys $\mathbf{K} = \mathbf{X}\mathbf{W}^K \in \mathbb{R}^{L \times d_k}$ and values $\mathbf{V} = \mathbf{X}\mathbf{W}^V \in \mathbb{R}^{L \times d_v}$ that can be calculated in parallel, where d_q,

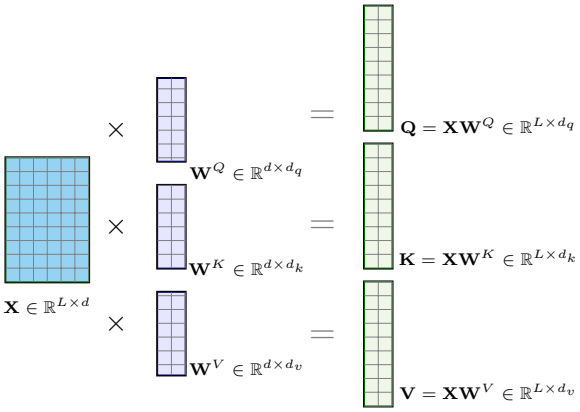


FIGURE 4.2: Diagrammatic representation of the computation of the attention matrix $\mathbf{A}$. An input sequence of length L and embedding dimension d is combined with learned weights to produce query, key and value matrices $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ respectively.

d_k, and d_v are the respective dimensions. A self-attention matrix $\mathbf{A} \in \mathbb{R}^{L \times d_v}$ can then be computed by

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A} = \text{softmax}\left(\mathbf{Q}\mathbf{K}^\top\right)\mathbf{V}. \tag{4.4}$$

4.2.3 The Rise of the Transformer

Seminal work by Vaswani et al. (2017) introduced an architecture dubbed the transformer, which is constructed entirely around self-attention. They showed that state-of-the-art performance in neural machine translation can be achieved without the need for any CNN or RNN components; as they put simply "attention is all you need". Such was the impact of this work that there has since been an explosion of transformer variants as researchers strive to develop more efficient implementations and new applications (Tay et al., 2020). It is the original architecture by Vaswani et al. (2017) that inspired the architecture proposed in this chapter, and as such the

remainder of this section focuses on describing the inner workings of this model.

As can be seen in Figure 4.3, the transformer consists of two sections: an *encoder* and a *decoder*. Within each encoder and decoder there exists a transformer-block, which contains the multi-head attention mechanism. In the context of neural machine translation, one could think of this set up as the encoder encoding a sentence in English, transforming the input into a certain representation, and the decoder taking this representation and performing the translation to French. To ensure the model only attends to words it has seen up to a certain point when decoding, an additional causal mask is applied to the input sentence. As an example, this may be the equivalent of only providing inputs $x_0^i \dots x_2^i$ of an input sequence of say $L = 5$ but requiring the decoder to output predictions up to $y_{L=5}^t$.

We focus our discussion on the transformer block without this causal mask since it is this block that is most relevant when we come to classification tasks later in this chapter. Notwithstanding, there is scope for further study to investigate the usefulness of applying a causal mask to the input sequence for early light curve classification. This would present an architecture that does not require full phase light curve information for predictions. By applying a causal mask, one can build a classifier that can ingest partial light curves and still provide predictions. Then by varying the amount of masking (*i.e.* increasing or decreasing the amount of the light curve that is visible to the network) we can investigate the feasibility of early light curve classification.

Multi-Headed Scaled Dot Product Self-Attention

Whilst the main building block used by Vaswani et al. (2017) is indeed the self-attention mechanism, they modified the typical dot-product attention by introducing a *scaled* element. This resulted in a new mechanism called the *scaled dot-product attention* which is similar to Equation 4.4 but with the input to the softmax scaled

down by a factor of d_k. The motivation for introducing a scaling factor is to control possible vanishing gradients that may arise from large dot-products between embeddings. The new formulation for this mechanism can be expressed as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \tag{4.5}$$

This now scaled version of the self-attention module was extended further to also have multiple heads h, which allows for the model to be able to learn from many representation subspaces at different positions simultaneously (Vaswani et al., 2017). Similar to normal self-attention calculations show in Section 4.2.2, this can be pictorially understood with Figure 4.4 and by concatenating the attentions for each head:

$$\text{MultiHead}\,(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A} = \text{Concat}[\mathbf{A}_1, \ldots, \mathbf{A}_h]\mathbf{W}^O, \tag{4.6}$$

where $\mathbf{A}_i = \text{Attention}\,(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)$. With each $\mathbf{A}_i \in \mathbb{R}^{L \times d_v}$, the result of a final linear transformation of all concatenated heads, $\text{Concat}[\mathbf{A}_i, \ldots, \mathbf{A}_h] \in \mathbb{R}^{L \times hd_v}$ with learned output weights $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d}$, produces the multi-headed attention matrix $\mathbf{A} \in \mathbb{R}^{L \times d}$.

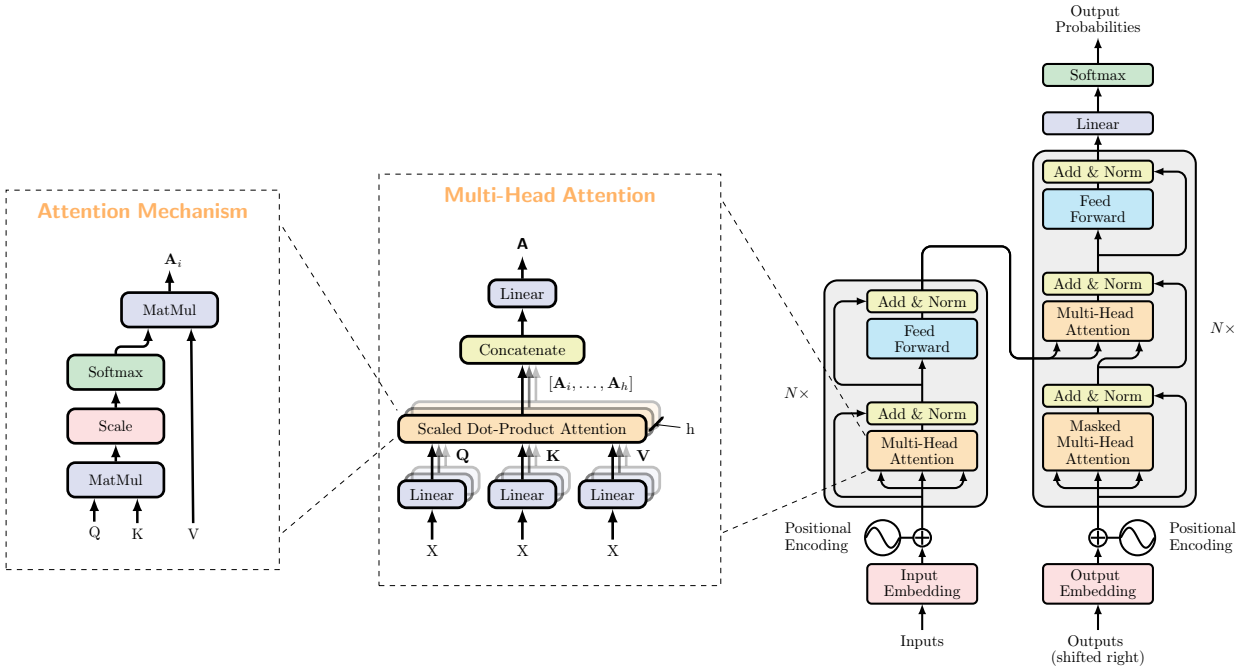


FIGURE 4.3: Layout of the original transformer architecture defined in Vaswani et al. (2017). The multi-head attention unit has been zoomed-in to reveal the inner workings and key component of the scaled dot-product attention mechanism. Note the two grey boxes on the left and right of the architecture. These are both transformer blocks, with N indicating how many times each block is stacked upon itself.

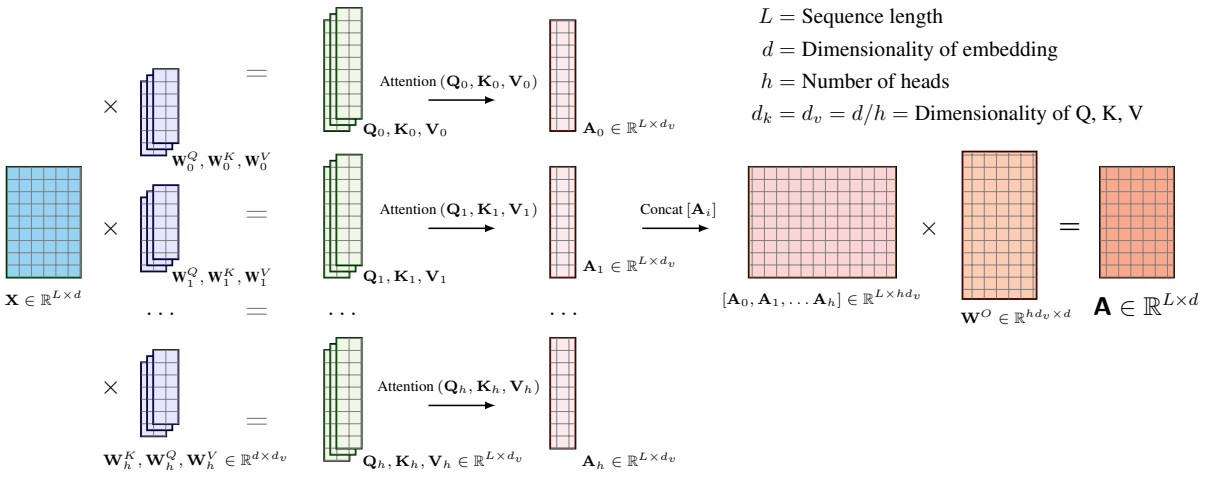


FIGURE 4.4: Diagrammatic representation of the computation of the multi-head attention. Instead of computing attention once, the multi-head mechanism divides the input with sequence length L and embedding dimension d by number of heads h to compute the scaled dot-product attention over each subspace simultaneously. These independent attention matrices are then concatenated together and linearly transformed into an attention matrix $\mathbf{A} \in \mathbb{R}^{L \times d}$. The above diagrammatic representation assumes the dimensionality of keys d_k is the same as the dimensionality of the values d_v.

Additional Transformer-Block Components

As can be seen Figure 4.3 inside the transformer-block, there is also a pathway that skips the multi-head attention unit and feeds directly into an *Add & Norm* layer. This skip-connection, often referred to a residual connection, allows for a flow of information to bypass potentially gradient-diminishing components. The information that flows around the multi-head attention block is combined with the output of the block and then normalised using layer normalisation (Ba et al., 2016) by

$$\mathbf{X} \leftarrow \text{LayerNorm}(\text{MultiHeadSelfAttention}(\mathbf{X})) + \mathbf{X}. \tag{4.7}$$

A feed-forward network follows, comprised of two dense layers with the first using ReLU activation (Nair and Hinton, 2010) and the second without any activation function. A similar skip connection occurs, but instead bypasses the feed-forward network, before being combined again and layer-normalised. It should be noted that all operations inside the transformer-block are time-distributed, which is to say that each word or vector representational embedding, is applied independently at all positions. When combining these elements together, this results in a single transformer-block:

$$\begin{aligned} \mathbf{X} &\leftarrow \text{LayerNorm}(\text{MultiHeadSelfAttention}(\mathbf{X})) + \mathbf{X} \\ \mathbf{X} &\leftarrow \text{LayerNorm}(\text{FeedForward}(\mathbf{X})) + \mathbf{X}, \end{aligned} \tag{4.8}$$

where $\mathbf{X}$ is the input embedding to the transformer-block.

Input Embedding and Positional Encoding

The inputs to the transformer are word embeddings created from typical vector representation algorithms such as `word2vec`. Applying this transformation projects each word token into a vector representation on which computations are made. Additionally, recall that attention is computed on sets of inputs, and thus the computation itself is permutation invariant. While this gives strengths to this model in terms of parallelism, a drawback of this is the loss of

temporal information that would usually be retained with RNNs. A consequence of this is the need for positional encodings to be applied to the input embeddings. In Vaswani et al. (2017) the positional encoding $\mathbf{P} \in \mathbb{R}^{L \times d}$, which is used to provide information about a specific position in a sentence (Weng, 2018), is computed by a combination of sine and cosine evaluations at varying frequencies. Assume l to be a particular position location in an input sequence, with $l = 1, \ldots, L$, and embedding index $k = 1, \ldots, d$, then

$$\mathbf{P}_{lk} = \begin{cases} \sin(\omega_k.l), & \text{if } k \text{ even} \\ \cos(\omega_k.l), & \text{otherwise} \end{cases} \tag{4.9}$$

where $\omega_k = \dfrac{1}{10000^{2k/d}}$.

Provided the dimension of the word embedding is equal to the dimension of the positional encoding, the positional vector $\mathbf{p}_l \in \mathbb{R}^d$ corresponding to a row of the matrix $\mathbf{P}$ is added to the corresponding word embedding $\mathbf{x}_l$ of the input sequence $[\mathbf{x}_1, ..., \mathbf{x}_n]$ (Kazemnejad, 2019):

$$\mathbf{x}_l \leftarrow \mathbf{x}_l \oplus \mathbf{p}_l. \tag{4.10}$$

For a visual representation of the position encoding see Figure 4.5, which depicts the positional encoding for a 128-dimensional by 100 sequence length input embedding. Using positional encoding in this way allows for the model to have access to a unique encoding for every position in the input sequence. The motivation for using sine and cosine functions are such that the model is also able to learn relative position information since any offset, $\mathbf{p}_{l+\text{offset}}$ can be represented as a linear function of $\mathbf{p}_l$ (Vaswani et al., 2017).

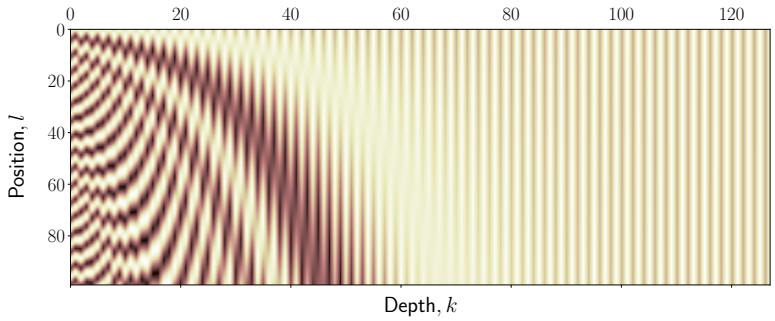


FIGURE 4.5: A 128-dimensional positional encoding for a sequence of length of 100. This can be understood as each row representing the encoding vector for that position in the sequence.

4.3 t2: The Time-series Transformer

In this section we present our transformer architecture for time-series data, which is based on the self attention mechanism and the transformer-block. Our work is motivated by photometric classification of astronomical transients but generally applicable for classification of general time-series. The time-series transformer architecture that we propose supports the inclusion of additional features, while also offering interpretability. Furthermore, we include layers to support the irregularly sampled multivariate time-series data typical of astronomical transients.

4.3.1 Architecture

Our architecture, referred to from herein as t2, shown in Figure 4.6, has several key differences compared to the original transformer shown in Figure 4.3. The first of these differences is the removal of the decoder. As the task at hand is classification, a single transformer-block is sufficient (Tay et al., 2020). Another difference can be seen with the additional two layers prior to posi-

tional encoding unit, which are *Gaussian Process Interpolation* and *Convolutional Embedding*. In conjunction with these two layers is a *Concatenation* layer that is able to add an arbitrary number of additional features to the network. These layers process the astronomical input sequence data and pass it to a typical transformer-block. The output of the transformer-block is then passed through a new *Global Average Pooling* layer, before finally being passed through a softmax function that provides output probabilities over the classes considered.

4.3.2 Convolutional Embedding

With neural machine translation applications the inputs to the original transformer architecture take in word embeddings that had been derived from a typical vector representation algorithm such as `word2vec`. In a similar manner, embeddings for the now interpolated time-series data are required. We adopt a simple 1-dimensional convolutional embedding, with a kernel size of 1 and apply a ReLU non-linearity. Inspired by Lin et al. (2013), a 1-dimensional convolution allows for a transformation from k-dimensional space to a k'-dimensional space whilst operating over a single time window of size of 1. For our purposes, using this convolution allows for dimensionality to be scaled from M to d dimensions without affecting the spatio-temporal input. Therefore, this operation transforms the original input of M-dimensional time-series data points, *i.e.* time-series data points across M passbands, into a d-dimensional vector representation ready for input into the transformer-block. We present the full input processing pipeline including the Gaussian process interpolation stage described in Section 3.4.2 in Figure 4.7. Our convolutional embedding operation is akin to a time-distributed, position-wise feed-forward neural network operating on each input position.

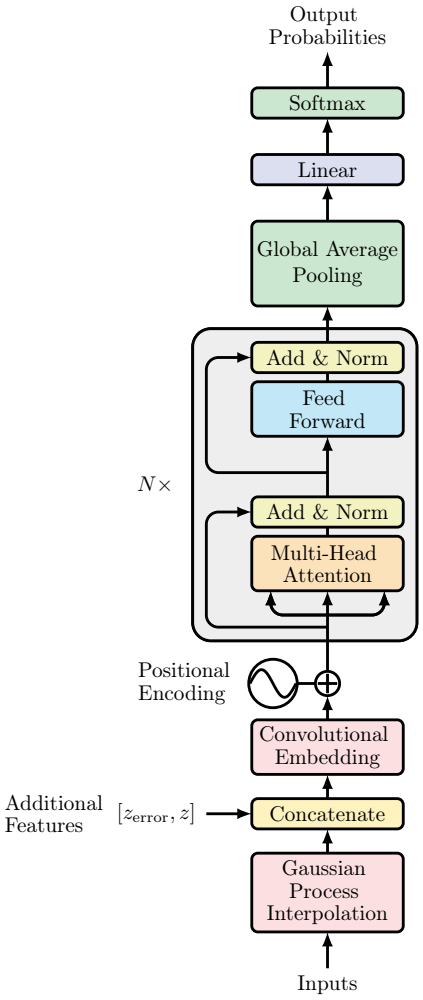


FIGURE 4.6: Schematic of the time-series transformer (`t2`) architecture. Raw time-series data is processed through the Gaussian process interpolation layer, followed by a concatenation layer to include any additional features. A convolutional embedding layer follows to transform the input into a vector representation, with a positional encoding applied to the embedding vector. This is passed as input into the transformer-block, where the multi-head attention block is the same as that shown in Figure 4.3. The output of which is then passed through a global average pooling layer and finally a linear layer with softmax to output class prediction probabilities for the objects.

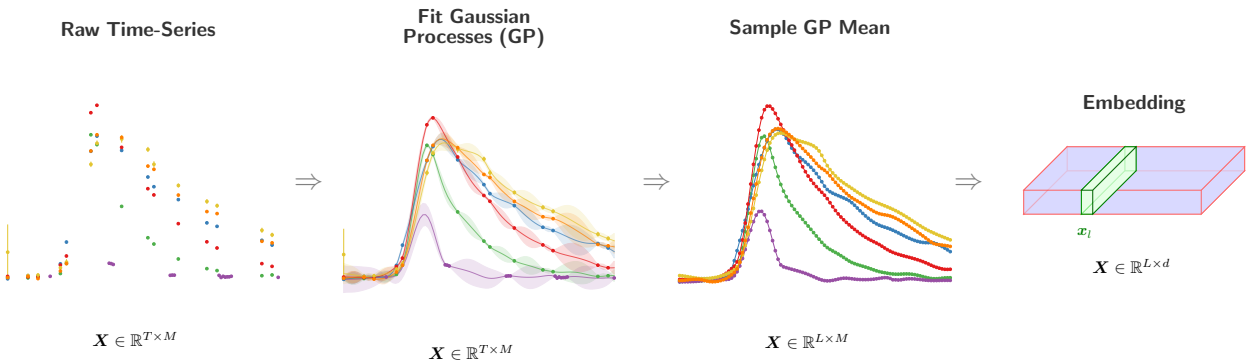


FIGURE 4.7: The three-stage process of transforming an astronomical transient light curve from raw photometric time-series data to a vector representation suitable as input to the time-series transformer. First, Gaussian process regression is carried out to regularly sample the light curve. The Gaussian process mean is evaluated at L time points l for each of M passbands, and projected to dimension d via a 1-dimensional convolutional embedding (Lin et al., 2013) at each point l such that final input sequence $\boldsymbol{X} \in \mathbb{R}^{L \times d}$. The resulting embedding matrix is then shown in blue, where for example a d-length vector for a single input position highlighted in green.

4.3.3 Global Average Pooling

We also introduce a layer that performs global average pooling (GAP) on the output of the transformer-block. The motivation for adding a GAP layer following the transformer-block was inspired by work in Zhou et al. (2015). The GAP layer, originally proposed by Lin et al. (2013), has become a staple in modern CNN architectures due to its usefulness in interpretable machine learning, and also featured in our Xception-derived network in Chapter 3. In previous works on 2-dimensional images, GAP layers are used as a replacement to common fully connected layers to avoid overfitting since there are no parameters to optimise. Another useful advantage over the fully connected layer is that the averaging in a GAP layer averages out the spatial information leaving it more robust to translations of the inputs (Lin et al., 2013). Similar to 2-dimensional inputs, using a GAP layer on the 1-dimensional time-series, proves robustness to translations in the input.

By adapting the description found in Zhou et al. (2015), one can apply a GAP layer to a time-series. Let $f_k(l)$ represent the activation of a particular embedded dimension k at a location l, where $k = 1, \ldots, d$ and $l = 1, \ldots, L$. Then a GAP layer can be computed by taking the average over time for each feature map $F_k = \sum_l f_k(l)$.

4.3.4 Class Activation Maps (CAM)

A nice feature of using a GAP layer is that one can determine the influence of $f_k(l)$ on predictions for a given class $c \in C$ by considering the associated score S_c that is passed into the softmax layer (Zhou et al., 2015). This is calculated from the final fully connected weights w_k^c and the feature maps F_k as $S_c = \sum_k w_k^c \cdot F_k = \sum_l \sum_k w_k^c \cdot f_k(l)$.

The class activation map (CAM) for a given class c is then given

by

$$M_c(l) = \sum_k w_k^c f_k(l). \tag{4.11}$$

Since $S_c = \sum_l M_c(l)$, it is possible to use $M_c(l)$ to directly gauge the importance of the activation at input location l in leading to the classification of class c.

4.3.5 Inputting Additional Information

Additional features could in principle be incorporated in the time-series transformer in a variety of different manners. To leverage the power of neural networks to model complex non-linear mappings, such additional features should feed through non-linear components of the architecture. On the other hand, recall from Section 4.3.4 that in order to compute a CAM (class activation map), the output of the GAP layer must pass directly into the linear softmax layer. Hence, incorporating additional features at this stage of the architecture will not be effective unless a non-linear activation is introduced, which would destroy the interpretability of the model.

To preserve our ability to compute CAMs, there are several other possible locations in the architecture where one could consider including additional features. The most natural point is immediately prior to the convolutional embedding layer (see Figure 4.6). Adding features at this location allows for all information to be passed throughout the entire network. Nevertheless, there are alternative ways in which additional features can be incorporated at this point.

The most obvious way to incorporate additional features is to essentially consider them as a additional channels and concatenate in the dimension of the M passbands to redefine the input as $\mathbf{X} \in \mathbb{R}^{L \times M} \rightarrow \mathbf{X} \in \mathbb{R}^{L \times M'}$, where $M' = M + R$, and R is the number of features to add. This essentially broadcasts the additional information to each input position in l.

The alternative is to concatenate in the dimension of the L time

sequence samples, which transforms the input as $\mathbf{X} \in \mathbb{R}^{L\times M} \rightarrow \mathbf{X} \in \mathbb{R}^{L'\times M}$, where $L' = L + R$. There are several advantages for choosing the approach of concatenating to L rather than M. Firstly, this approach allows one to pay attention to the additional features explicitly. Secondly, it gives activation weights for the additional features, which in our case is redshift and redshift error, so the impact of the additional features can be interpreted. So, while in principle one could consider concatenating to either L or M, we advocate concatenating to L.

4.3.6 Trainable Parameters and Hyperparameters

The time-series transformer, `t2`, model contains a set of trainable parameters that stem from the weights contained in the transformer-block as well as learned weights at the embedding layer and final fully connected layer. The first layer with trainable parameters is the convolutional embedding layer. The numbers of parameters for a general convolutional layer is given by

$$[M \times w \times d] + d,$$

where M denotes the number of input channels or passbands, w refers to the kernel window size, which in this case is 1, and d is the dimensionality of the embedding. Continuing through the model, the number of trainable parameters for the multi-head attention unit has 4 linear connections, including $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ and one after the concatenation layer, *i.e.* $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ and $\mathbf{W}^O$. Recall that for multi-head attention we set $hd_v = d$ (see Figure 4.4), hence the number of parameters for $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ and $\mathbf{W}^O$ across all of the h heads is identical. The number of layer normalisation parameters is simply the sum of weights and biases together with the feed forward neural network weights of the input, multiplied by the weights of the output plus the output biases (Chai, 2020). Combining all units inside the transformer block together yield

$$N \times \underbrace{\big(4 \times \big[(d \times d) + d\big]\,\big)}_{\text{Multi-Head Attention}} + \underbrace{\big(2 \times 2d\big)}_{\text{Layer Norm}} + \underbrace{\big(d \times d_{\text{ff}} + d_{\text{ff}}\big) + \big(d_{\text{ff}} \times d + d\big)}_{\text{Feed Forward}}$$

where N refers to how many times one stacks the transformer-block upon itself, and d_{ff} refers to the number of neurons in the feed forward network inside the transformer-block. Since there are no trainable parameters with the GAP layer, the final fully connected linear layer with softmax results in a remaining number of trainable parameters of

$$\left([d \times C] + C\right),$$

where C refers to the number of classes.

Of the parameters discussed above, there are some that are fixed due to the problem at hand, such as M number of passbands and C classes to classify. But there are also other parameters that are not necessarily trainable that are considered *hyperparameters*. These include: the dimensionality of the input embedding d, the dimensionality of the feed forward network inside the transformer-block d_{ff}, the number of heads to use in conjunction with the multi-head attention unit h, the percentage of neurons to drop when in training using the dropout method (Srivastava et al., 2014) `droprate`, the number of transformer-blocks N, and the learning rate `learning_rate` (discussed further in Section 4.4.2).

4.4 Implementation and Training

We leverage modern machine learning frameworks to develop the time-series transformer implementation, `t2`, in a modular manner for ease of use and future extension. We use the same evaluation metrics of those described in Section 3.5 to measure the performance of the classifier. Later in Section 4.4.3 we recap the loss function used for training and how hyperparameters are optimised.

4.4.1 Implementation

We use the machine learning framework of `TensorFlow` (Martín Abadi et al., 2015) with the `tf.keras` API for the implementation of our `t2` architecture. Our code is available under Apache 2.0 licence and open-sourced[1]. Key data processing software of `pandas` (McKinney, 2010) and `numpy` (Harris et al., 2020) has been used heavily for manipulation of input data, with `george` (Ambikasaran et al., 2015) used for fitting the Gaussian processes. Training and inference of our model has been carried out on a NVIDIA Tesla V100 GPU.

4.4.2 Training

Using the typical stochastic gradient decent (SGD) method mentioned in Section 3.5.2, we train our `t2` model with the same optimisation algorithm described in Section 3.5.3, namely the ADAM optimiser (Kingma and Ba, 2014), along with the same learning schedule with a reduction of the learning rate of 10% if the loss does not decrease within 5 epochs. We also keep track of the validation loss as to not overfit to the training data.

4.4.3 Hyperparameter Optimisation

Much like our astronomical transient xception network (Chapter 3), the time-series transformer has fixed parameters that are set by the problem itself *i.e.* number of passband filters M and number of classes C, as well as a set of hyperparameters that can be tuned for optimal performance. To search for the best set of hyperparameters we formulate the problem in the same way as that described in Section 3.5.4 by considering the optimisation problem of

$$\theta^* = \arg\min_{\theta \in \Theta} g(\theta), \tag{4.12}$$

[1] github.com/tallamjr/astronet

with $g(\theta)$ the objective score to be minimised and evaluated on the validation set and θ being the set of hyperparameters, that take values defined in the domain of Θ. Using the logarithmic-loss function defined in Equation 3.20 as our objective score, we look for the set of hyperparameters that give the lowest objective score, θ^*.

To avoid a costly grid search for the best hyperparameters, we again use a Bayesian optimisation technique of Tree-structured Parzen Estimator (TPE) (Bergstra et al., 2011), that is implemented in the `optuna` package (Akiba et al., 2019), with 5 fold cross-validation.

4.5 Results

Much like with our previous network described in Chapter 3, we first tested our architecture on MTS to gauge general multivariate time-series classification performance. These results are presented in Appendix A. Although found to be widely applicable to all forms of multivariate time-series data, we present the results here for the application of our time-series transformer architecture to the task of photometric classification of astronomical transients. For this we use the representative but imbalanced in redshift dataset constructed from PLAsTiCC (outlined in 3.6.1) to showcase the suitability of our model for photometric classification, with a view to extend our methods to include augmentation techniques that can address the representativity problem (*e.g.* Alves et al. 2022a; Boone 2019; Revsbech et al. 2018) in the future.

4.5.1 Classification Performance

Of the model parameters in the time-series transformer, `t2`, there are a subset of hyperparameters that are tunable and can be optimised for (see Section 4.4.3). Through application of the TPE

TABLE 4.1: The time-series transformer, `t2`, contains 6 hyperparameters to be optimised. The set of parameters and learning rate that scored the lowest objective score using 5-fold cross-validation and the TPE Bayesian optimisation method is shown here. To be concise we only show `learning_rate` to 3 decimal places and advise the reader to refer to the code for full details.

Parameter	Value
d	32
h	16
d_{ff}	128
N	1
`droprate`	0.1
`learning_rate`	0.017

Bayesian optimisation method on a validation set constructed from 10% of the training set, using 5-fold cross-validation we obtained the parameters which gave the lowest objective score. The results of which can be found in Table 4.1.

When we build our time-series transformer with the hyperparameters shown in Table 4.1, and train a model using the training data set described in 3.6.1 we are able to achieve a logarithmic-loss of 0.507. This is comparable to the best performing model of Boone (2019) that achieves 0.468. The confusion matrix depicted in Figure 4.9 shows good performance across all classes, and betters our previous efficient architecture performance presented in Chapter 3. Both receiver operating characteristic (ROC) and precision-recall (PR) plots, Figure 4.10 and Figure 4.11 respectively, show reasonable multi-class classification accuracy, with the exception being towards the Kilonovae and SNIax classes. We suspect this is purely down to the scarcity of sample for Kilonovae and light curve similarity to SNIa in the case of SNIax as mentioned in 3.6. We achieve a SNIa purity of 0.94 with a core-collapse SNe (SNe Ib/c and SNe II) cross contamination of $\sim 4.81\%$. This compares to the reported

by DES (Vincenzi et al., 2021) and Pan-STARRS (Jones et al., 2018) of acceptable range for cross-contamination of $\sim$ 8% and $\sim$ 5% respectively, allowing for our results to useable for cosmological analyses of dark energy equation of state. The performance of our model expectedly degrades when auxiliary information of redshift and redshift error is not included. However, we find it promising that our model with raw time-series information only can still achieve a logarithmic-loss of 0.873, beating `atx` in this regard also.

It is expected that if a full hyperparameter search can be performed on the full training set by leveraging greater computational resources, it is likely better parameters could be discovered leading to improved performance. While a direct comparison with other methods presented in Hložek et al. (2020b) cannot be made since they have been trained with non-representative datasets, the time-series transformer is able to achieve excellent classification performance with minimal feature selection and few trainable parameters by deep learning standards.

It is often the case with machine learning models that, as remarked upon in Hložek et al. (2020b) and Lochner et al. (2016), in order to overcome a classification bias towards particular classes, an equal distribution of samples among the classes is often necessary for accurate classification. However, the `t2` architecture is able to handle class imbalance very well, and as such our model did not require any data augmentation in order to achieve a good score, unlike other methods. It is uncertain at this time whether this is an inherent property of transformers or the attention mechanism, or perhaps the architecture is simply able to find sufficient discriminative features with far fewer training samples than was previously thought is required for deep learning approaches such as CNNs and RNNs. As discussed already, we are yet to consider the case of data that is not representative in redshift, where augmentation techniques will certainly be necessary, which will be the focus of future work.

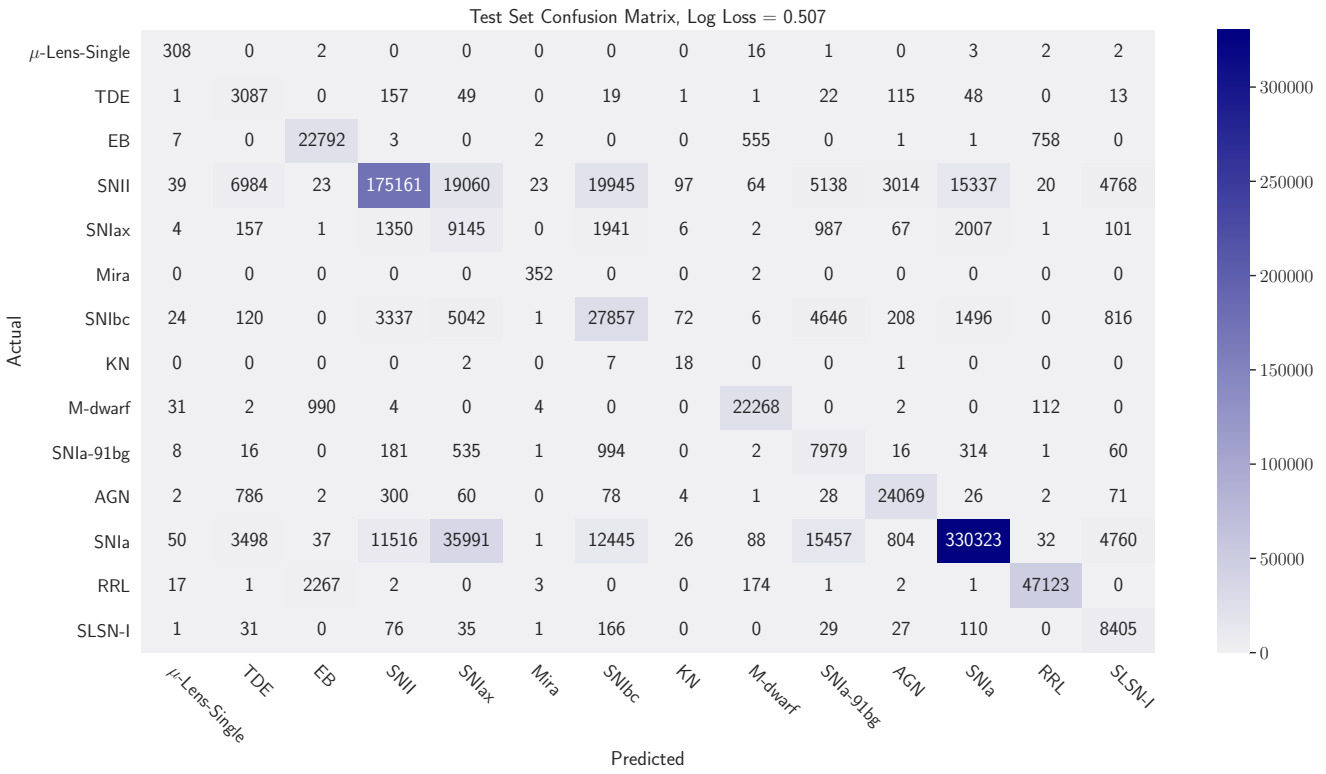


FIGURE 4.8: Raw count confusion matrix resulting from application of the time-series transformer, `t2`, to the PLAsTiCC dataset in a representative setting with imbalanced classes, achieving a logarithmic-loss of 0.507 and SNIa purity of 0.94 with a core-collapse SNe cross contamination rate of 4.8%

4.5.2 Interpretable Machine Learning

Work by Zhou et al. (2015) lead the way forward with major improvements for model interpretability. Their use of the GAP (global average pooling) layer for the localisation of feature importance helped researchers discover methods of visually inspecting a classifier's performance. In a similar regard, a GAP layer is included in the `t2` architecture to allow for model interpretability through the visualisation of the various feature maps as a function of sequence length, l. As discussed in Section 4.3.4, one can compute a CAM (class activation map) which can help determine how the features at each input position have influenced the final prediction. Also recall from Section 4.3.5 that `t2` allows for concatenation of arbitrary additional features; in this work we consider the addition of redshift information.

Of the two options for concatenation, either in time or passband, we adopt the approach of concatenating to L in time to give $L' = L + R$, where $R = 2$ with redshift and redshift error added as additional features. This has the advantage that we explicitly pay attention to redshift information and also get interpretability with respect to redshift information (see Section 4.3.5). For completeness, we also re-run the photometric classification analysis discussed previously by concatenation to M, but we do not observe as good a performance as concatenating to L. As we suspected, this may well be because we are explicitly paying attention to redshift in the multi-head attention mechanism, whereas by concatenating to M we do not get this benefit.

The CAM can then be computed by Equation 4.11, where $M_c(l')$ indicates the influence each position of the input sequence has on classification, which also includes redshift information, *i.e.* $l' = 1 \ldots, L + R$. We apply a min-max scaling and normalise the CAM such that $\sum_1^{L+R} M_c(l') = 1$, so that the relative activation weights can be interpreted as a percentage.

We show in Figure 4.12 illustrative CAMs for two Supernova classes, over-plotted with the light curves themselves. In each panel

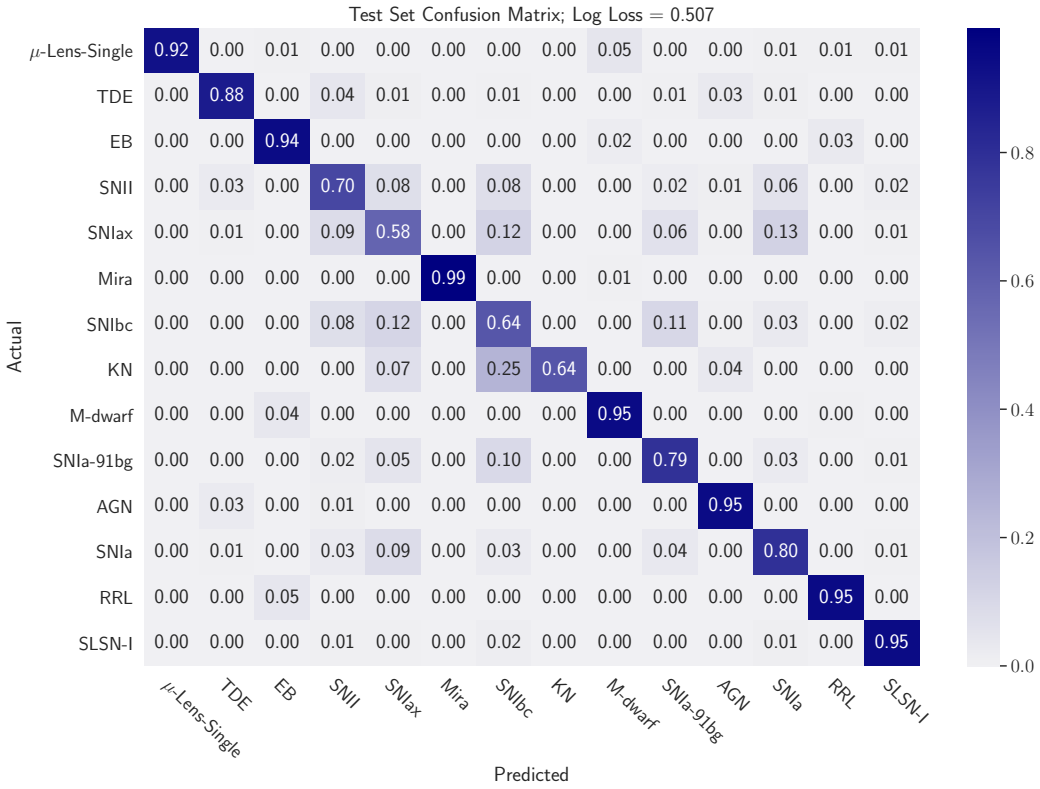


FIGURE 4.9: Normalised confusion matrix resulting from application of the time-series transformer, `t2`, to the PLAsTiCC dataset in a representative setting with imbalanced classes, achieving a logarithmic-loss of 0.507.

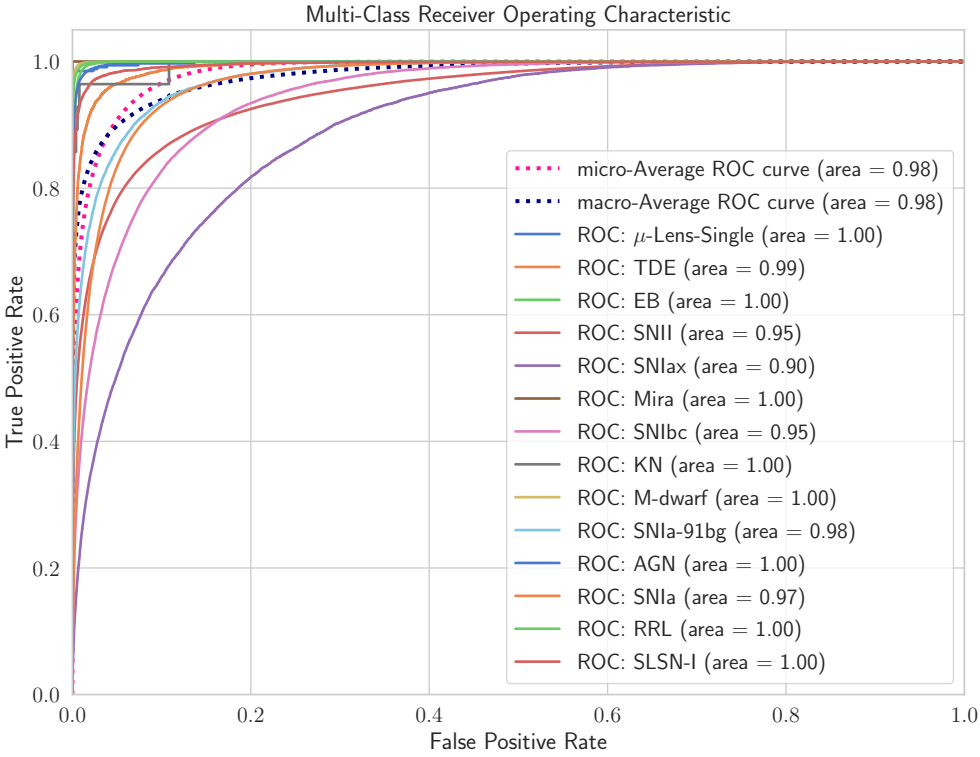


FIGURE 4.10: Receiver operating characteristic (ROC) curve, under the same setting as those described in Figure 4.9. Micro- and macro-averaged AUC scores of 0.98 are achieved across the classes.

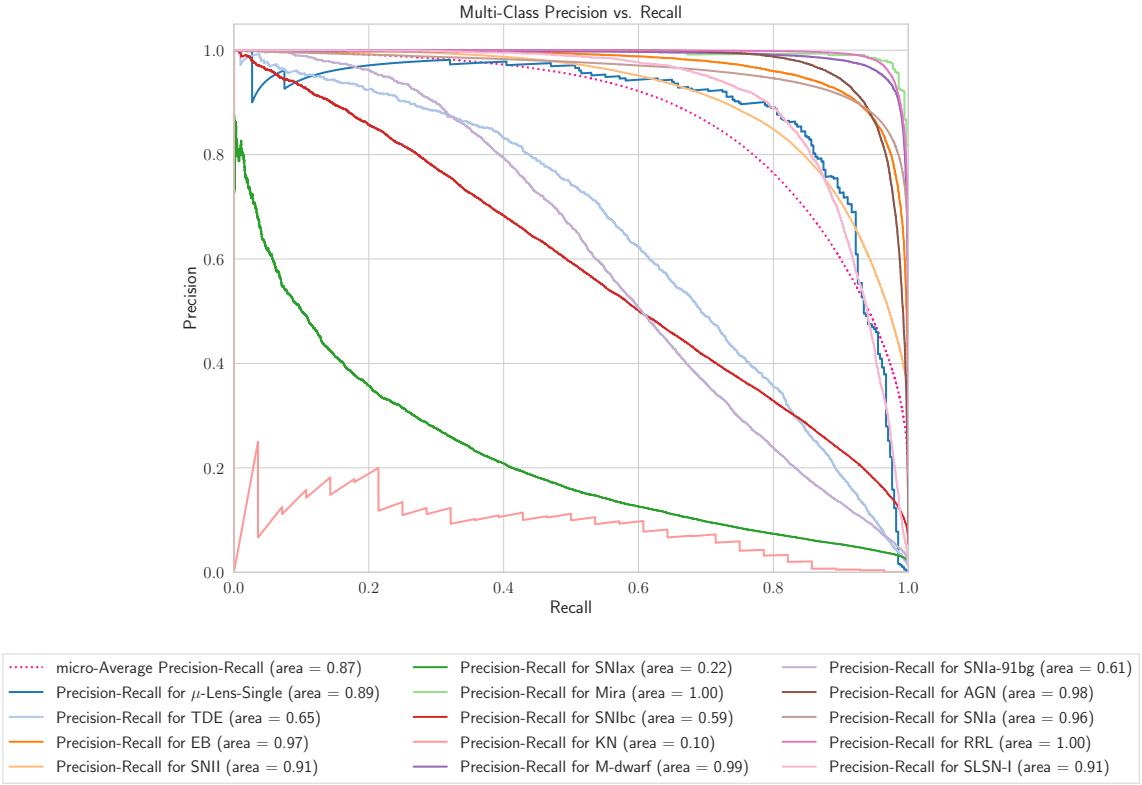
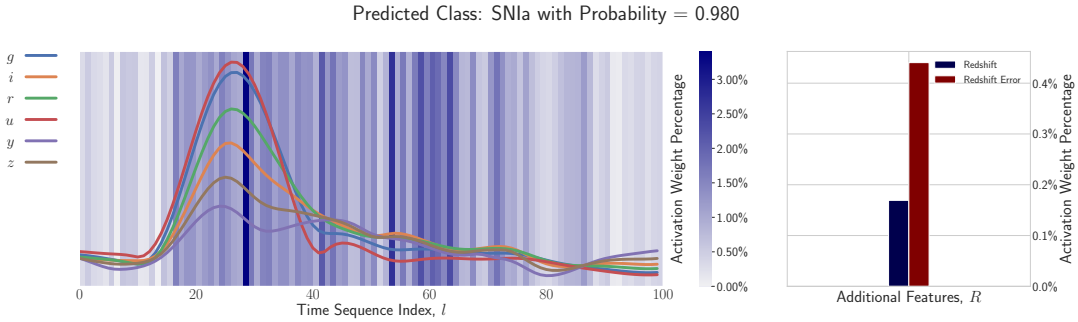
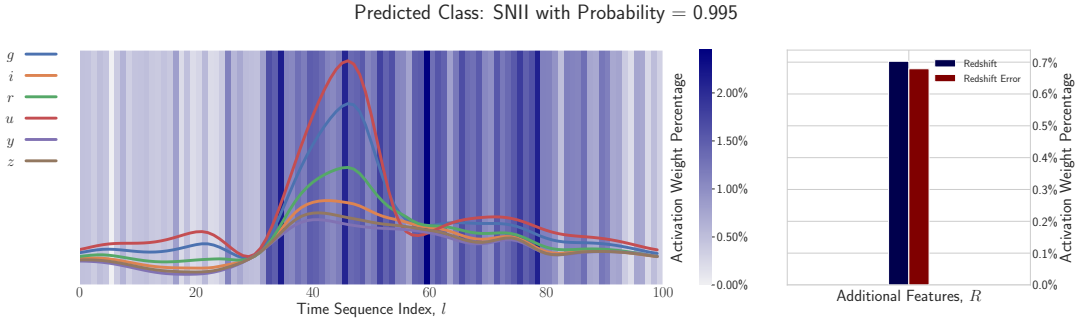


FIGURE 4.11: Precision-recall trade-off curve, under the same setting as those described in Figure 4.9. A micro-averaged AUC score of 0.87 is achieved across the classes. The model understandably struggles with precision for Kilonovae (KN), which only constitutes 0.004% of the training sample.



(a) Class activation map for a Supernova Type Ia drawn from the test set.



(b) Class activation map for a Supernova Type II drawn from the test set.

FIGURE 4.12: Class activation maps (CAM) for two types of Supernova drawn from the test set, with light curves for bands *giruyz* over-plotted. For visualisation purposes, a min-max scaling is applied to the class activations as well as a normalisation to each CAM such that $\sum_{l'} M_c(l') = 1$, such that the relative activation weights can be interpreted as a percentage. The left hand side depicts the percentage of activation weight attributed to each position in the sequence, while on the right hand side we show the percentage activation weights associated with any additional features that have been added; in our case redshift and redshift error. Notice that for both examples the activation weight is generally low before the initial rise of the light curve, larger at the rise, with the strongest weight around the peak. Moderate weights are observed in the tail, presumably to detect any secondary peak, with the weights becoming insignificant once the light curve falls back to zero. The influence of redshift and information can be seen on the right hand side, with $\sim 0.6\%$ and $\sim 1.4\%$ of the total activation weight being attributed these additional features for each object, respectively.

CAM probabilities for each light curve time point are shown, in addition to the CAM probabilities for the additional features of redshift and redshift error. Notice that for both examples the activation weight is generally low before the initial rise of the light curve, larger at the rise, with the strongest weight around the peak. Moderate weights are observed in the tail, presumably to detect any secondary peak, with the weights becoming insignificant once the light curve falls back to zero.

As our architecture is able to include additional features, these can also be inspected and visualised to gain further understanding as to how much importance the model is paying towards them. In our case, with the addition of redshift and redshift error information, we also include bar plots in Figure 4.12 that depict the activation weight for redshift and redshift error. We inspect the distribution of the activation weights for redshift and redshift error for all classes combined, which can be seen in Figure 4.13. The majority of activation weighting relating to redshift and redshift error falls around 1%. We also explored this distribution on an individual class by class basis but did not find a significant difference across classes. Therefore, there does not seem to be a particular class that benefits from redshift information over another. The distributions indicate that for most objects redshift information accounts for a relatively small proportion of the total activation weights, with a mean of $\sim 1.92\%$. However, it should be noted that this is related to the $L = 100$ regularly sampled points on the light curve, many of which are highly informative. Furthermore, we recall that redshift on the whole is indeed important for accurate classification where we achieve a close to state-of-the-art logarithmic-loss (set by Boone (2019) with 0.468) of 0.507 when including redshift information and 0.873 when it is not included (Section 4.5.1).

While we have shown CAMs to be useful for a first attempt to bring interpretability to light curve classification, we acknowledge more recent saliency mapping techniques that address some of the shortcomings of CAMs. We commented earlier that in order to compute CAMs we require the GAP layer. Although we have pro-

vided separate motivation for using a GAP layer (see Section 4.3.3) it may be the case that in the pursuit of better interpretability, requiring a GAP layer unnecessarily restricts the flexibility of our model for possible model extensions. Therefore it would be preferable to have an explainable methodology that does not impose certain characteristics on the architecture itself, and that can ideally probe a model in a black-box fashion. Follow-up work by Selvaraju et al. (2017) presented Grad-CAM that did away with the need for a GAP layer to feed directly into the softmax and was agnostic to the downstream task, but still required access to the internals of the model with gradients. An interpretability method proposed by Petsiuk et al. (2018) introduced randomised input sampling for explanation of black-box models (RISE) to better estimate how salient aspects of the input are for a model's prediction, without the need for access to model internals nor re-implementation of existing models. A further useful interpretability tool that was not used here but would assist with identifying which features were most helpful with correct prediction is SHAP (SHapley Additive exPlanations) values (Lundberg and Lee, 2017). SHAP assigns each feature an importance value for a particular prediction and can build intuition of a models performance (Lundberg and Lee, 2017).

While in this work we explore only CAMs as an initial investigation into the feasibility of using interpretability methods for photometric classifiers, we could have equally used other methods described above such as Grad-CAM, and this would be an interesting area to investigate further. Having said that, it is expected that future studies for interpretability of photometric classification architectures will look to methods such as SHAP for feature importance or using techniques similar to RISE that can treat the model as a black-box and yet provide more refined saliency maps.

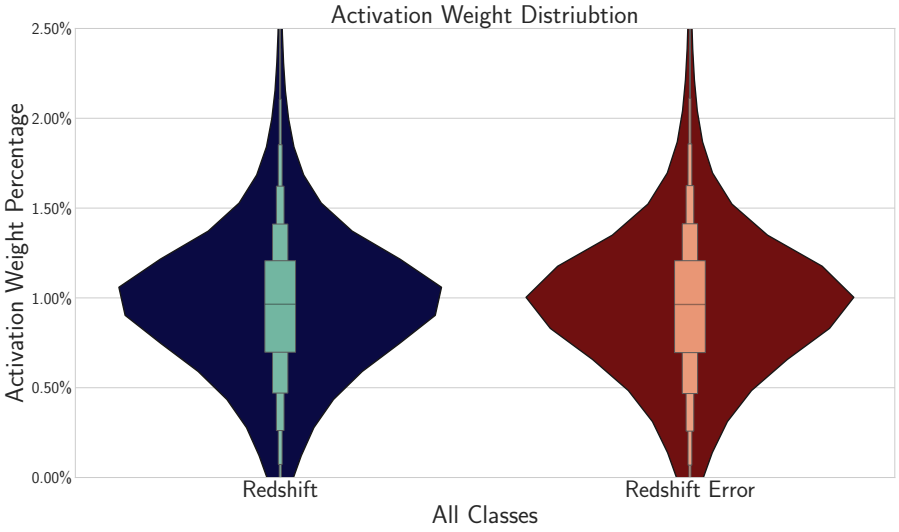


FIGURE 4.13: Distribution of activation weights for redshift and redshift error for all classes combined. This plot is constructed for all classes combined (minimal variability was observed across classes when plotted separately). The mean redshift and redshift error activation weights are both 0.96. In the centre of the plotted distribution we plot letter-value plots (Hofmann et al., 2011) that are better suited to large datasets such as this. The middle box contains 50% of the data, with the median indicated by a line at the midpoint. The next smaller boxes combined contain 25% of the data, with each successive level outward continuing in this fashion containing half of the remaining data.

4.6 Conclusions

We have constructed a new deep learning architecture designed for photometic classification of astronomical transients that we call the time-series transformer or `t2`. The architecture is designed in such a way to pay attention not only to light curves but also to any additional features considered (*e.g.* redshift information) and to also provide interpretability, again not only to light curves but also to additional features. While we are motivated by the problem of astronomical transient classification, the architecture is suitable for general multivariate time-series data.

The time-series transformer, `t2`, is able to achieve results comparable to the state-of-the-art in photometric classification and does so on extremely imbalanced datasets. Our architecture is able to achieve a logarithmic-loss of 0.507 on the PLAsTiCC dataset defined in Section 3.6.1 and Table 3.1. A direct comparison to other latest methods laid out in Hložek et al. (2020b) and Gabruseva et al. (2020) is understandably not possible since each classifier has been evaluated on different data under different conditions, nonetheless, `t2` is able to achieve the lowest logarithmic-loss on such imbalanced data, without the need for augmentation compared to the leading method of Boone (2019) that rebalanced classes using Gaussian processes. Having such an imbalanced dataset, one would expect that there would be bias towards the most common classes, but `t2` is robust enough to handle this. As noted in Lochner et al. (2016), accurate photometric classification requires a representative training dataset, but as discussed in Section 4.1 the data that will be observed with upcoming surveys will be non-representative of the training datasets that are currently available. While this work focuses on the representative setting, the architecture lends itself well to be able to be used in conjunction with latest augmentation techniques, particularly Boone (2019) and Alves et al. (2022a) with use of Gaussian processes, that should help to allevi-

ate non-representative training dataset issues, and as such this will be considered in detail in future work.

The relatively few parameters involved, and hence faster training times, compared to other deep learning methods makes `t2` an attractive architecture for potentially combining with active learning methods or even off-line retraining should new data become available. With the small model size, `t2` should also appeal to upcoming brokering systems such as FINK (Möller et al., 2021), ANTARES (Matheson et al., 2021) *etc.* that benefit from low latency and fast inference times when put into production. As we touched on in Section 4.2.3, the current architecture forgoes the additional decoder found in the original transformer architecture (Vaswani et al., 2017) that applies a causal mask to the input. However, the inclusion of such a mask would provide a natural mechanism within the time-series transformer architecture for early light curve classification, which provides another avenue of future work.

The time-series transformer, `t2`, minimises the reliance of expert feature selection. Moving away from feature engineering allows the model the freedom to discover patterns that are missed by humans but yet provide powerful discriminative information for classification. The architecture, by virtue of CAMs (class activation maps), offers up a helpful tool for interpretability by inspecting the importance of both light curves and any additional features that are included[2]. It is hoped that with the introduction of the attention mechanism to the field of astronomical photometric classification, further studies will build on this work to improve our ability to attend to the night sky.

[2]The reader is reminded of alternative interpretability techniques that may provide better explainability such as RISE (Petsiuk et al., 2018).

5

Deep Learning Deployment: Scaling Inference for Real-time Classification using Deep Model Compression

"*Premature optimization is the root of all evil.*"

— **Donald Knuth.**

In this chapter we present the model compression and model optimisation techniques applied to our deep learning model, the time-series transformer (Allam, Jr. and McEwen, 2021), for deployment as a science module within the alert brokering system FINK (Möller et al., 2021) for real-time classification. In order to perform real-time inference on the deluge of data that will be streamed from the Legacy Survey of Space and Time (LSST) of the Vera C. Rubin Observatory, the model that is to be deployed into production will need to be computationally lightweight, and yet be able to provide robust classification scores. The large volume of data calls for models that not only have ultra-low latency inference times, but can also scale out computations across many machines for high-throughput processing of the alerts. We showcase how the use of modern deep compression methods can achieve a 18× reduction in model size, whilst preserving classification performance. We also show that in addition to the deep compression techniques, careful choice of file formats can improve inference latency, and thereby throughput of alerts, on the order of 8× for local processing, and 5× in a live production setting. To test this in a live setting, we deploy this optimised version of the time-series transformer, `t2`,

into FINK (Möller et al., 2021) on real Zwicky Transient Facility (ZTF) (Bellm, 2014) alert data, and compare throughput performance with other science modules that exist in FINK. The results shown herein emphasise the time-series transformer's suitability for real-time classification at LSST scale, and beyond, and introduce deep model compression as a fundamental tool for improving deploy-ability and scalable inference of deep learning models for transient classification.

5.1 Inference in the Age of Large Synoptic Surveys

The turn of the century has seen a move towards ever larger astronomical surveys, collecting large volumes of synoptic data across the night-sky, as opposed to previous instruments that focus data collection for a single science case. Being able to conduct a large swath of science from a single data source is one of the main drivers for development and construction of such surveys, and allows for many science communities to benefit from a single instrument. Recent surveys such the Sloan Digital Sky Survey (SDSS; York et al., 2000), the Dark Energy Survey (DES; Abbott et al., 2016), the Panoramic Survey Telescope and Rapid Response System (Pan-STARRS; Kaiser et al., 2002) to name a few, are examples of astronomical surveys that map the sky without a particular astronomical *target* in mind. They are often limited in scope in terms of electromagnetic spectrum, but can serve as the precursor to more specialised instruments for follow-up observations. Typically, surveys are used to generate catalogues of astronomical objects, as well as logging astronomical transient events on the sky.

The detection of transient events is of particular importance for reasons described in Chapter 1. With difference imaging processing now done entirely in software, the need to automate pipelines for detection and classification of transient events comes from the sheer

volume of data these surveys generate, as well as the number of events they witness. Developments in instrumentation have allowed these surveys to scan larger areas of the sky and more detailed than ever before, with the consequence being that machine learning has now become a critical component in order to process the data.

When the Legacy Survey of Space and Time (LSST) at the Vera C. Rubin Observatory comes online, it is expected to observe 10 million transient events per night, generating on the order of 1TB of data per night[1]. The tsunami of transient alerts (see Figure 5.1) that is to be distributed globally, calls for machine learning systems that can scale to such data rates, and yet still provide robust identification of events. A classifier that can process an alert and provide classification scores in real-time will not only enable efficient allocation of resources for follow-up observations, but assist with labelling of the millions of events which is certainly beyond humans at this point. At this scale, storage space and computational costs becomes a real concern, and so for real-time processing and machine learning enriched catalogues to be feasible, classification modules need to be lightweight in terms of storage space and runtime memory footprint. Furthermore, computations should be done as efficiently as possible to not only save on time, but also money by minimising energy usage.

We structure this chapter in the following way. Section 5.2 discusses the challenges involved when dealing with a large influx of data from space surveys, and stresses the need for alert brokering systems. We touch on the policies in place for developing such brokers and describe what motivates the use of the Zwicky Transient Facility (ZTF) (Bellm, 2014) alert stream in preparation for the upcoming LSST data distribution in Section 5.2.2. Following that, our focus turns to one broking system in particular, FINK (Möller et al., 2021), which served as the platform for the research discussed

[1]Raw data volume for image and calibration data will not be distributed with the alerts to save space and reduce the data sent over the network, which would otherwise amount to 60PB over the course of 10 years of operation.

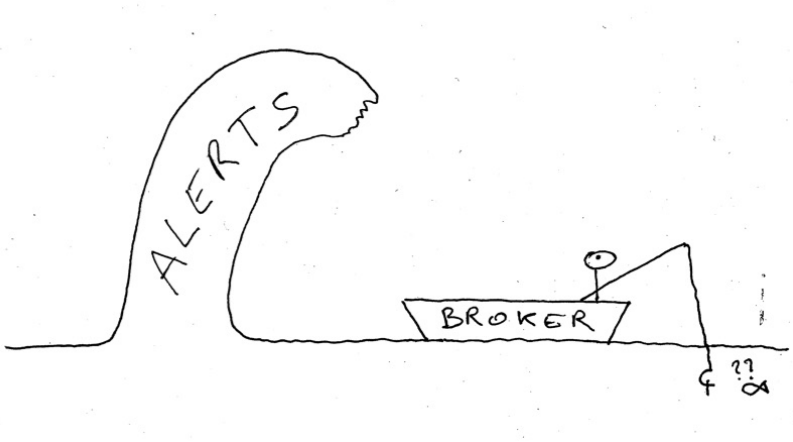


FIGURE 5.1: The transient alert tsunami expected from full LSST operations will equate to approximately 10 million alerts per night, amassing 3PB over the lifetime of the survey. Figure reproduced with permission from FINK Workshop 2020 (Peloton et al., 2020)

later in this chapter. Section 5.3 explains the ideas behind the engineering efforts that ultimately allowed for our deep learning model to successfully operate in a production system. Our preliminary results on local tests are showcased in Section 5.4, followed by the results of applying our methods in real-world conditions in Section 5.5. We then conclude in Section 5.6 with a discussion of these results, and how use of the methods described facilitate efficient deep learning and real-time inference, at LSST scale.

5.2 Calling All Brokers!

Due to computational constraints, and limits on bandwidth, the full distribution of alerts from LSST Data Facility[2] will only be sent out to a select few community brokering systems, whose primary purpose is to provide catalogue cross-match functionality

and photometric classification of objects, thereby enriching the raw alert packets with value-added information for downstream scientific investigations. A worldwide call for brokering systems was announced in 2019 to entice teams interested in potentially building such systems (Bellm et al., 2019b), which was soon followed by a call for concrete proposals the following year (Bellm et al., 2020).

5.2.1 Community Alert Brokers

Since the full alert stream can not be accessed directly by scientists, community brokering systems are essential software systems that will enable time-domain science (Bellm et al., 2019a). Further to the requirements of cross-matching and photometric classification, brokers are also expected to provide additional services to enable science such as a simplified user interface for easy querying of archival data, a triggering follow-up observing service, additional alert filtering[3], among others. The call for brokers was not limited to any institutions in particular, and the open call encouraged a variety of system designs. As long as there is capacity for petabytes (PB) of storage space, a large inbound bandwidth network, real-time machine learning classification modules, and of course sufficient funding, brokering teams were free to set out their plans in The Call for Proposals for Community Alert Brokers (Bellm et al., 2020). Naturally, brokers that offer a suite of services along with the necessary infrastructure capabilities, were seen more favourably by the selection committee, and in particular brokers that take advantage of the unique real-time aspects of the LSST alert stream (Bellm et al., 2020). Moreover, brokers that already exhibit integrations with follow-up resources and other surveys through existing agreements, as well as scope for community building, was also viewed positively.

Of the many teams that put forward proposals, seven teams

[2] lsst.org/about/dm/facilities

[3] The LSST Data Facility applies its own filtering of alert packets such as a criterion of SNR > 5 and before distribution

were ultimately chosen that showcased their ability to match the criteria laid out above, or at least showed the potential to realise the requirements come time of first light. The successful broker bids were from teams; The Automatic Learning for the Rapid Classification of Events (ALerCE) (Förster et al., 2021), AMPEL (Nordin et al., 2019), Arizona-NOAO Temporal Analysis and Response to Events System (ANTARES) (Matheson et al., 2021), BABAMUL (Duev and Graham, 2022), Pitt-Google (Wood-Vasey et al., 2022), Lasair (Smith, 2019) and FINK (Möller et al., 2021).

5.2.2 The ZTF Alert Stream: A Proxy for Success

In order to support development of the broking systems, LSST provided example alert streams to get broking teams familiar with the expected data formats and protocols. Much of these were inspired by the Zwicky Transient Facility (ZTF) already-in-action alert distribution system (Patterson et al., 2019). The Zwicky Transient Facility (ZTF) is an astronomical survey that observes in visible and infrared, primarily focusing on the detection of transient objects that change rapidly (Bellm, 2014). Its high cadence allows it to observe the entire northern sky in three nights over two passbands. Although generating only a tenth of the data expected from the LSST, the data pipelines and alert distribution systems in place with ZTF have been shown to be suitable to act as a precursor to the much larger data rates of fully operational LSST (Patterson et al., 2018).

The ZTF alert packets are in `Apache Avro`[4] format, a binary serialisation format, that contains difference imaging information of the detection, yet is still compact and lightweight enough for real-time worldwide publishing. Deserialisation is done in conjunction with a corresponding alert schema that defines the contents of the alert packet, and hence the information that can be used for processing. LSST is set to follow the same data format as well as the

[4] github.com/apache/avro

same *pub-sub* framework using `Apache Kafka`[5] for the distribution of the streaming data, where it collects data streams at source, from *producers* (*e.g.* the telescope itself) and arranges them into sets of *topics* that can be subscribed to by *consumers* downstream (*e.g.* community alerts brokers).

The ZTF alert production system has shown to successfully support streams of 1.2 million nightly alerts, which equates to approximately 70 GB per night, where each alert packet has been made available within 10 seconds of event detection (Patterson et al., 2018). On the order of 80,000 alerts per minute, the stability of the production system when dealing with such data rates gives support to the case for using the same technologies and protocols described in Patterson et al. (2018) for developing brokers that are to be suitable for the 10× larger upcoming Large Survey of Space and Time.

5.2.3 FINK: A Next Generation Broker

Of the seven brokers mentioned in Section 1.3.2 and Section 5.2.1 that were successful in securing a spot as one of the brokering systems, we discuss FINK in more detail here. FINK is the system that our deep learning model was deployed to, and was the ultimate test bed for investigating the real-time capabilities of the time-series transformer. FINK goes beyond typical brokering systems by providing real-time transient classification through fast state-of-the-art deep learning algorithms that can be re-trained at low cost in a short space of time, and by using active learning techniques, specifically online learning, that allow for continuous self-improvement of classification scores. Specifically designed to address the challenges outlined in Section 5.1, it uses industry standard tools for efficient big data processing. In order to carry out nightly processing of the terabyte data stream, FINK uses fault-tolerant `Apache Spark` (Zaharia et al., 2016) for scaling out computations across

[5] github.com/apache/kafka

many computers, and Spark Structured Streaming (Armbrust et al., 2018) to easily interact with `Apache Kafka` for *consuming* the data stream.

FINK currently has a memorandum-of-understanding (MoU) with the Zwicky Transient Facility (ZTF), allowing it to receive real alert packet data, in the form described in Section 5.2.2, each night. This makes FINK well suited for not only testing how well our deep learning model can perform under stress with real-time constraints, but also to test how well our model handles classification of *real* data. The FINK system diagram can be seen in Figure 5.2. Along with mapping the flow of data through the broker, Figure 5.2 also shows at which stage the redistribution of enriched alerts will take place. The interplay of the photometric classification modules in the Processing cluster, additional third-party survey data via the Communication cluster and aggregation of value-added information in the Storage cluster, form the foundations of the Science Portal, which can be used to enable scientific analysis and offline querying of the archival data for those in the community.

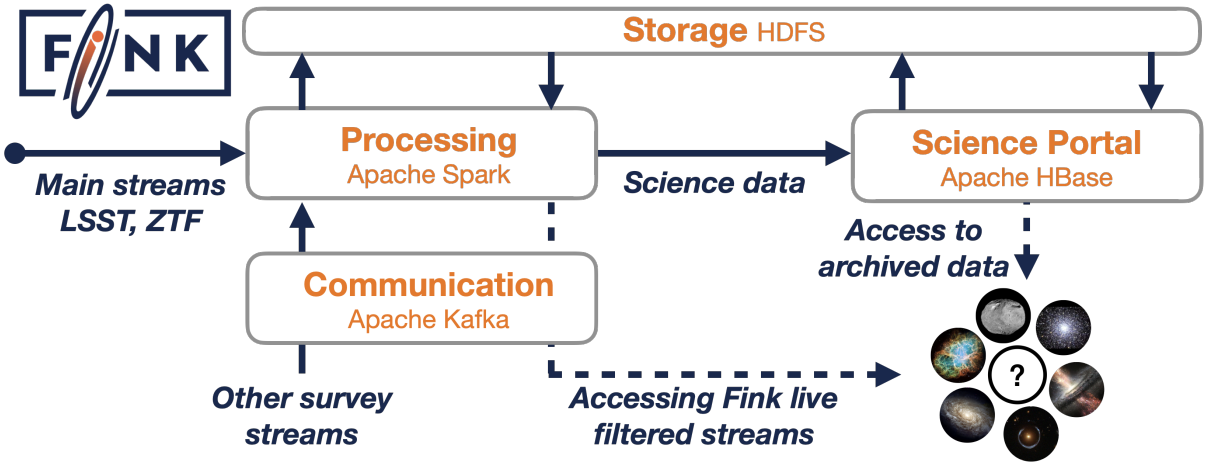


FIGURE 5.2: FINK pipeline and system architecture, where the main alert streams are processed in a distributed manner using `Apache Spark` (Zaharia et al., 2016) on a Processing cluster. Following the initial processing, a set of sub-streams are produced which users can subscribe to by way of the Communication cluster using `Apache Kafka`. Further survey data streams, such as those from LIGO, Fermi, *etc.* are ingested into the Processing cluster through the Communication cluster, to enrich alert packets with added information. This is used in conjunction with science modules within the Processing cluster that provide classification scores for alerts and added-value information. After a night of operation, the processed and enhanced data is written to the Hadoop Distributed File System (`HDFS`) in the Storage cluster which connects to a Science Portal backed by the distributed database of `Apache HBase`[6] to allow for interactive querying of archived alerts. Figure reproduced in full from Möller et al. (2021)

5.3 Performance Engineering for Deployment in FINK

Since photometric classifiers are to be housed in the Processing cluster (see Figure 5.2), low-latency, high-throughput algorithms are essential to handle the deluge of data that is to be processed. This section describes the research and development of a multi-stage compression process in order to ensure best classification performance, while optimising for low-latency inference and high-throughput processing of alerts.

5.3.1 Line Profile Analysis

In order to better understand the bottlenecks in our deep learning pipeline from alert packet processing to inference, as well as to investigate the overall systems performance, it is necessary to conduct a form of dynamic program analysis. In contrast to static program analysis, which evaluates a program without execution, dynamic program analysis focuses on the program's memory usage, time complexity, duration of function calls *etc.* . Such analyses are typically done through unit and integration tests, which themselves may include, or can exist separate to the main codebase and line profiling tests that scrutinise a program line-by-line. By observing time spent at each function call, one can see where in the pipeline optimisations can be made, and as such apply performance engineering techniques that reduce runtime and memory footprint of the program.

To run such a test, we simulate locally the full pipeline from ingestion of a real ZTF alert packet, to interpolated time-series, to model predictions, and gauge where optimisations should be applied by using the line profiling tool `kernprof`[7]. The line profiling software reports the time spent on each function and the number

[6] github.com/apache/hbase

of times that function has been called, for each line of code in a program.

Although initial expectations were that the main bottleneck would be the time-series interpolation through Gaussian processes, the major bottleneck in the pipeline was found to be loading our model into memory and applying the model to the input data for predictions (see Listing 5.1). To combat this, we looked into ways of reducing model size for faster model loading and operational changes to improve runtime latency.

Listing 5.1: Line profile report for initial run of alert packet pipeline. Superfluous lines that recored less than 0.1% time are removed for better readability. Note the function to generate the Gaussian process only takes 9.5% of the total time, with the majority of time taken up with model predictions. For the full report, see github.com/tallamjr/astronet/astronet/tests/reg

```
Total time: 5.85664 s
File: get_models.py
Function: get_model at line 29

Total time: 1.47076 s
File: ztf-load-run-lpa.py
Function: t2_probs at line 55

Line #    Hits       Time Per Hit \% Time Line Contents
==============================================================
...
206      16      139.6    8.7    9.5    df_gp_mean = generate_gp_all_objects()
...
...
...
...
...
212       8      180.8   22.6   12.3    X = df_gp_mean[cols]
213       8       12.3    1.5    0.8    X = rs(X)
...
...
...
217       8     1101.7  137.7   74.9    y_preds = model.predict(X)
```

[7] github.com/pyutils/line_profiler

5.3.2 *Deep* Compression

With the major bottleneck for fast inference identified to be at the model load and then prediction stage, we focus our attention to model optimisations that can be applied to alleviate this. Since our desire is to run the complex model in real-time, we look to exploit redundancies in the model, thereby reducing the storage size, lowering inference latency, and improving energy efficiency processing alerts. A relatively recent area of research that looks to reduce model size and memory footprint of deep learning models it that of *deep compression* (Han et al., 2015a). Originally proposed as a three-step process to reduce the computational cost and memory usage of deep networks on embedded devices, it is mostly driven today by the interests of industry for deploying deep learning models in-the-wild on resource constrained devices. This influential work saw a new field flourish that combines bit saving best-practises with deep learning architecture design to reduce storage size, whilst at the same time preserving model accuracy.

For our research, we restrict our investigation to the techniques broadly laid out in Han et al. (2015a), namely *weight-pruning* and *weight-clustering* with Huffman encoding, and *weight-quantization*. All of which can be applied in isolation or together, with the caveat being that if these techniques are chained together, the possibility for severe degradation in performance is high.

Pruning

Pruning is a technique that removes unimportant weights to yield improvements such as better generalisation and improved speed of learning and classification (LeCun et al., 1989b). It has been shown in recent times that deep neural network can be pruned to a significant degree with little reduction in model accuracy (Han et al., 2015a,b).

While there are many forms of network pruning such as layer-pruning (Lazarevich et al., 2021), channel-pruning (He et al., 2017), filter-pruning (Enderich et al., 2021) and connection-

pruning (Nguyen et al., 2021), we consider magnitude-based weight pruning here (Han et al., 2015a), where the weights are updated network-wide through a small number of fine-tuning epochs to zero out model weights that have a low impact on the final score, creating a sparse representation of the model. Sparse models[8] can then leverage standard lossless compression algorithms for large reduction in model size, as well as faster inference through fewer parameters and hence fewer computations.

Clustering

Another method that promotes sparsity in the network is through weight-clustering. Also commonly referred to as weight-sharing, clustering works by grouping the weights of each layer in a model into a predefined number of clusters. The centroid values for the clusters are then shared among the weights in the given cluster. By dividing the m original number of weights in the network, $W = \{w_1, \ldots, w_m\}$ into k clusters $C = \{c_1, \ldots, c_k\}$, where $m \gg k$ there is a great reduction in the number of *unique* weight values in a model, which in turn allows for greater storage efficiency and high data compression potential. If we consider there to be n possible connections in the network, where each connection is represented by b bits, then a fully connected network would be represented with $n \cdot b$ bits. A clustered network on the other hand, with k clusters, requires only a cluster encoding index of $\log_2(k)$ together with the number of clusters with shared weights. This yields a compression rate, r, of

$$r = \frac{n \cdot b}{n \cdot \log_2(k) + k \cdot b}. \tag{5.1}$$

An example using a single fully connected four-by-four neural network can be see in Figure 5.3. If we use Equation 5.1, we can see

[8]Stored in compressed sparse row (CSR) or Compressed Sparse Column (CSC) format gives $2a + n + 1$ numbers, where a is the number of non-zero elements, and n is the number or rows or columns, which is normally $\ll L \times M$ size of a complete matrix of all elements

that by setting $k = 4$ (using 4 distinct colours to signify separate clusters), one is able to reduce required number of bits for the original 16 weights at 32 bit precision, down by a notable factor with a compression rate of 3.2.

The canonical k-means clustering algorithm is used to find the clusters, but of great importance in terms of the eventual model accuracy is how the centroids are initialised. Of the three methods for centroid initialisation in Han et al. (2015a), random, density-based and linear, the authors report that random and density-based centroid initialisation achieve poor performance due to few centroids having large absolute values. Linear initialisation on the other hand does not suffer from this problem, and is shown by Han et al. (2015a) to work best under various conditions. A comparison of the different centroid initialisation schemes is shown in Figure 5.4. For training, a weight lookup table is necessary to maintain information about the shared weights and their connections among the clusters. The gradient for each shared weight is then calculated and used to update the actual shared weight, as can be seen in Figure 5.3. The gradient of the centroids is given by,

$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbb{1}\left(I_{ij} = k\right) \tag{5.2}$$

where $\mathcal{L}$ is the loss, C_k as the k-th centroid, and the centroid index of the weight matrix W_{ij} is I_{ij}.

At the stage for which the model is to be deployed, *i.e.* for inference operations only, the intermediate weight table can be *stripped* from the model to leave just the clustered weights, suitable for standard data compression algorithms to reduce model size on disk.

Quantization

Quantization is the simple procedure of reducing the number of bits used for representing numbers. Weight-quantization helps reduce the storage and computational requirement of the model, and in the case of our discussion here, applied after training is completed[9].

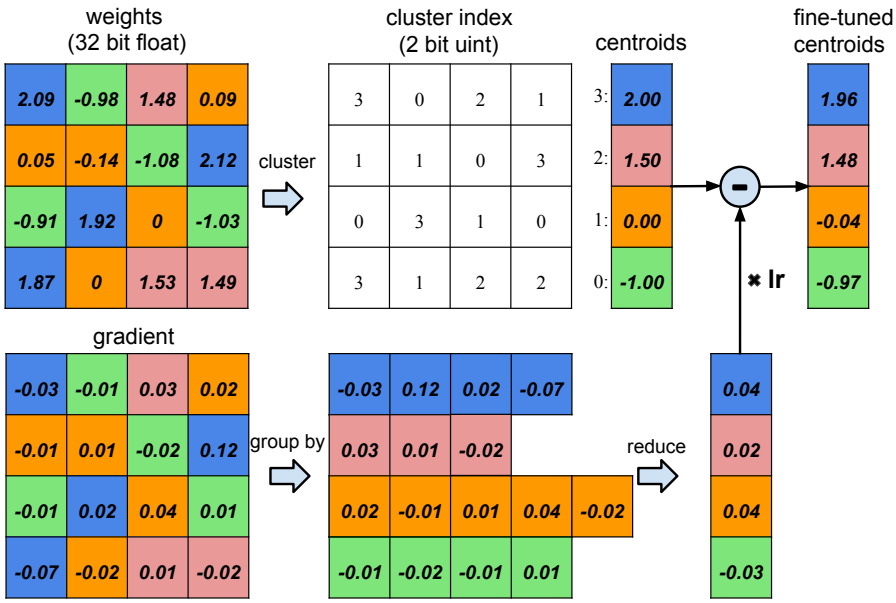


FIGURE 5.3: Weight clustering compression scheme, showing the weights of a single layer neural network that has four input and four output units. In total there are 16 weights, which are reduced to a set of 4 shared weights. The top row depicts the full weight matrix for the 4 by 4 input-output connections, whereas the bottom row shows the related gradient matrix. As an example, using 4 colours to denote the 4 clusters, the set of weights are put into one of 4 clusters, where all values in the same cluster share the same value. As such, an index mapping the weight to a particular cluster is stored. In the training phase, the gradients are grouped by colour (cluster), summed, multiplied by the learning rate, and finally subtracted from the shared centroids of the last iteration. Reproduced in full from Han et al. (2015a)

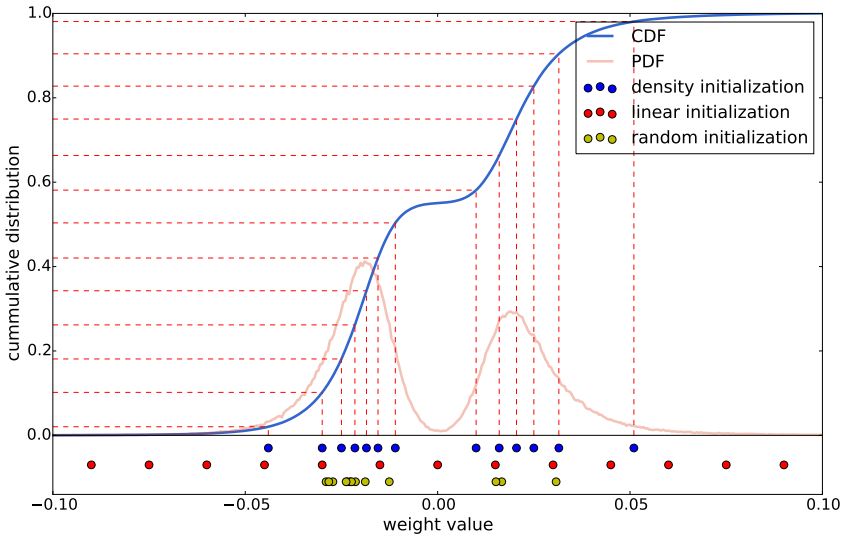


FIGURE 5.4: Centroid initialisation schemes, using the bimodal distribution of weights in CONV3 layer of AlexNet (Krizhevsky et al., 2012) as an example. Shown at the bottom are the 13 cluster centroids for this example layer using 3 different types of centroid initialisation schemes. **Random**: randomly selects k points from the data set and uses these as the initial centroids, shown in yellow. **Density**: uses the cumulative distribution of the weights to create a linear spacing on the y-axis, and then finds the corresponding x-axis value that intersects with the distribution, shown in blue. **Linear**: evenly spaces the x-axis of weights from min to max value and then places a centroid, shown in red. Random initialization tends to concentrate around the peaks of the distribution, as does density-based initialization, albeit more scattered. Linear is even more scattered, but to note the initialization scheme is invariant to the weight distribution. Reproduced in full from Han et al. (2015a)

Post-training quantization refers to the application of quantization by statically mapping the weight values to lower precision integers, where in this lower precision representation, the weights save significant amount of space on disk, and can even see improvements to latency by leveraging efficient integer kernel operators found in modern hardware accelerators.

5.3.3 Lossless Data Compression

As touched on in our brief overview of deep compression, many of the techniques lend themselves well to exploitation by standard lossless data compression algorithms. Both pruning and clustering induce redundancies in the model through repeated values. The canonical compression scheme to handle repeated values is Huffman encoding (Huffman, 1952), which assigns fewer bits to repeated values. As such, it is recommended to combine sparsity inducing methods of deep compression with lossless data compression algorithms. We use the DEFLATE algorithm (Deutsch, 1996) within `zlib`[10] which combines Huffman encoding with the LZ77 compression algorithm (Ziv and Lempel, 1977) to realise the full benefits of sparsifiying our network. Though, we are mindful of the potential trade-off between storage space savings when using lossless compression tools and the inevitable higher latency caused by the decompression overheads when loading models into memory.

5.3.4 Efficient File Formats and Frameworks

While application of deep compression techniques are likely to significantly reduce the size of our deep learning model on disk, the possible increase in latency times in relation to decompression overheads spurred an off-shoot evaluation which looked at alternative

[9] Quantization aware training on the other hand is a quantization procedure that is applied during training by way of integer arithmetic for computations.
[10] github.com/madler/zlib

file formats and lighter frameworks that could help improve runtime of model predictions.

Both `atx` (see Chapter 3) and `t2` (see Chapter 4) use `ProtocolBuffers`[11] as the serialisation format for saving models developed using the full TensorFlow framework (Martín Abadi et al., 2015). Inspired by the *TinyML* movement (David et al., 2021), that seeks to run deep learning on *extremely* resource constrained devices, we look at the possibility of using only a subset of the full TensorFlow framework, called TensorFlow Lite (TFLite) (Li, 2020). Compared to the some 1400 operations in the full framework, TFLite only has around 130 operations supported (David et al., 2021). A model developed using the lighter TFLite framework is represented in a different file format than that of the full TensorFlow model, called `FlatBuffers`[12]. This portable, efficient, binary file format offers a couple of advantages over using the `ProtocolBuffers` model file format, such as smaller file size through the reduced operations and code footprint, as well as much faster inference by way of zero-copy deserialisation for direct memory access without having to copy it into a separate part of memory first for an additional parsing or unpacking step.

For the most part, deep learning architectures are still designed and built using the full TensorFlow framework, and only when the practitioner is satisfied, is the model then *converted* to a TFLite model, using the helpful TFLite converter tool. The process of converting the original model to a TFLite version is where many optimisations actually take place, with the principle optimisation being operator fusion.

TensorFlow operations are themselves simple primitive operations which can be combined together to form more complex operations. The primitive operations appear as a single node in the computational graph that is constructed by TensorFlow at runtime. Composite operations that are built from multiple primitives, ap-

[11] github.com/protocolbuffers/protobuf
[12] github.com/google/flatbuffers

pear as separate nodes for each primitive operation. Fused operations, on the other hand, act as a single operation that comprises of all the computations that each primitive operation would normally have, as a single node in the graph. By taking advantage of the underlying kernel implementations, fused operations can maximize performance and reduce the code and memory footprint, perfect for the resource constrained devices that it was designed for, as well as for situations that demand low-latency inference, as in our case. A useful by-product of fused operators is a high level interface that helps define complex transformations such as quantization, that would otherwise be cumbersome to map network wide.

Under the TFLite framework, in conjunction with fused operations, quantization is far easier to achieve, and actually appears as a simple flag at conversion time when going from the original model to the TFLite version. As described in Section 5.3.2, quantization is the procedure of mapping floating point values to a reduced integer representation (see Figure 5.5), and in this TFLite setting, falls under post-training quantization umbrella. Perhaps unique to TFLite, is that at runtime, the model weights that are saved as integers, are scaled back to an approximation of the original floating point values, to allow for computations using floating point kernels to give better consistency with how inferences would have resulted had the model had not been quantized in the first place. The formulae for approximating floating point values from the saved integer weights can seen here,

$$R = (Q - Z) \times S, \tag{5.3}$$

where the real value R is approximated by a scale factor, S, that multiples the difference between the Q-bits representation (which is commonly taken to be 8 for 8-bit integer precision) and the zero-point value Z.

This section has described the compression schemes and optimisations applied to our deep learning model to improve latency and reduce model size, yet with the aim to preserve accuracy. Figure 5.6 shows where each of these techniques have been used in

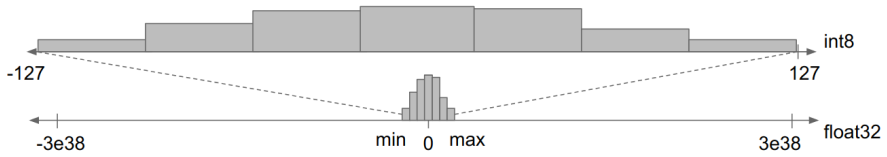


FIGURE 5.5: Quantization mapping of float representation to integers representation. With the full range for 32-bit floating points extending from $3e^{-38}$ to $3e^{38}$, there is often a remarkable amount of bits reserved for the precision, when in fact the majority of the numbers exists within a much narrow range on the number line. Integer numbers represented with 8-bits extend from -127 to 128 for signed values, and 0 to 255 for unsigned integers. With the appropriate mapping and scale factor, 32-bit numbers can be easily be approximated as 8-bit integers, though 8-bit precision only has 255 information channels, this is a lossy conversion. Reproduced in full from Bhuwalka et al. (2020)

the time-series transformer architecture. Notably, weight-clustering has not been used architecture-wide as it is not advisable to cluster weights in critical layers early in the network[13]. However, as weight-pruning and weight-quantization are done post-training, we are able to apply these techniques to the model as a whole.

5.3.5 Hardware-Accelerated Distributed-Training

So far we have discussed optimisations that can be applied to the model in order to save on disk space and improve runtime latency such that classification scores can be given in real-time on the high-volume of incoming alerts. However, of significant importance is the ability to quickly retrain and update the model as new data becomes available. It is expected that there will be a window of 8 hours where new models will have the opportunity to be retrained before a new round of data ingestion and processing takes

[13] tensorflow.org/model_optimization/guide/clustering

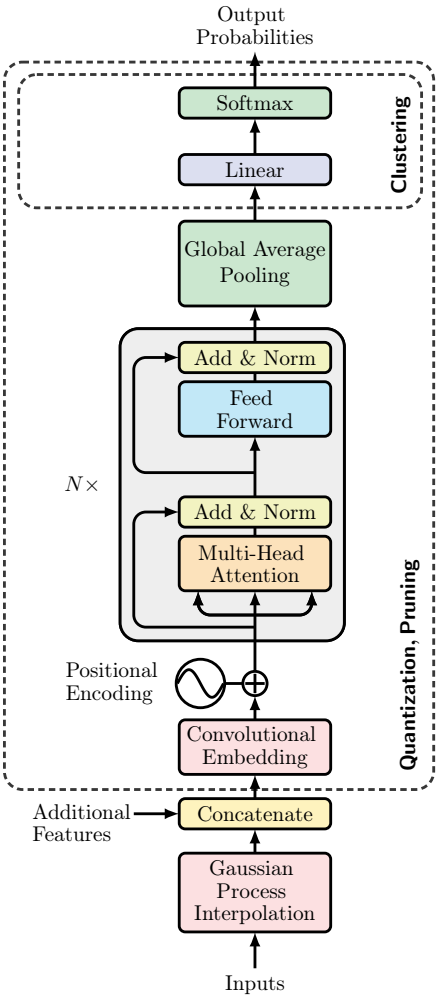


FIGURE 5.6: Locations within the time-series transformer (`t2`) architecture, where deep model compression techniques have been applied. We investigate three forms of model compression, weight-pruning, weight-quantization and weight-clustering. Both weight-pruning and weight-quantization techniques are applied post-training, and are applied on all weights in the network. Weight-clustering on the other hand, is applied *during* training, and only on the final dense layer to exploit parameter redundancy and to avoid the critical layers such as those in the attention block.

place (Möller et al., 2021). Therefore, to have a model that can be retrained, and hence improved with more refined data, within this window is highly desirable.

As with the case in FINK, which scales out computation across many CPU-only machines in the Processing cluster to be able to churn through the large amount of data quickly, we can take advantage of the same data parallelism principles to also scale out computations for retraining models. By simply splitting the dataset across multiple compute nodes, one can achieve a near linear-time speed up. However, beyond scaling out to more CPU cores, we re-implement the time-series transformer's training loop to scale out computations with multiple hardware accelerators, in this case graphical processing units (GPUs), for maximum speed up[14]. Compared to CPUs, GPUs do computations far more efficiently, saving time, energy and costs in the long-term, but to ensure one takes full advantage of the hardware accelerators capabilities, maximising the memory use at all times is essential. If a GPU is underutilised, depending on the model and data input size, severe training time degradation can be observed due to communication overheads from host device (CPU) to the GPU accelerator[15].

The main, and perhaps most straightforward, way one can ensure maximum utilisation is to increase data input *batch size*. It is worth clarifying here that in this setting batch size does not refer to the full dataset containing *all* training samples, rather it is a subset of training samples, equivalently called a *minibatch* (Good-

[14]We note that the codebase can easily be extended to run on even faster tensor processing units (TPUs), but this was not taken further due to lack of available resources.

[15]It is worth emphasising that the FINK system does not have GPU accelerators itself and all inference is done on CPU-only machines. Therefore at runtime in FINK our model needs to be loaded each time by the CPU. If on the other hand FINK used GPUs in the Processing cluster, one could leave the model in memory disregarding the problems of model size in relation to frequent loads from disk.

fellow et al., 2016). We shall use the terms interchangeably going forward.

Batch size in itself can have a major impact on model convergence, but it plays a significant role when striving for optimal performance and computational efficiency. Since TensorFlow uses 32-bit precision for floating point operations on the GPU, model parameters take up 4 bytes each. Using this information, it is possible to determine the largest practical batch size that can deliver maximum GPU utilisation.

In addition to greater computational efficiency, larger batch sizes on the GPU are also expected to yield slight classification performance gains. If we consider that the standard error of the mean is estimated from n samples as $\sigma/\sqrt{n}$, with σ as the standard deviation of the samples, we can see that with larger n, one can obtain a more reasonable estimate for gradients (Goodfellow et al., 2016, p. 271). While it would normally be the case that the non-linear scaling of gradient estimates would invoke a trade-off between how many samples to use and compute resources, such is the computational efficiency of GPUs that the limiting factor becomes the amount of memory that can be used instead. Therefore, by increasing the batch size to be as large as possible for a single GPU, and then scaling this by the number of GPUs available, through an *all-reduce* operation when running our stochastic gradient optimisation, we can realise the improved classification performance in addition to computational improvements as well.

5.4 Preliminary Results

This section presents the preliminary results of applying the model optimisations outlined in this chapter to the original time-series transformer model. We first run local processing tests on real ZTF alert packets to gauge the potential performance when deployed

into the production system of FINK (which is discussed in the next section).

5.4.1 Model Retraining

The original time-series transformer was trained using a single NVIDIA V100 GPU, with 32GB of memory. Section 5.3.5 explained how computational efficiency gains could be made by increasing batch size. As a first test, we extend the `t2` codebase to allow for multiple GPU training while ensuring the largest batch size possible is dispatched to each GPU. Through the model profiling tool of `model-profiler`[16] we determine the best minibatch size to be 4096 using the same GPU as before. Now on the order of 100 times larger batch size compared to the original model described in Chapter 4, we see far greater utilisation of the GPU.

Scaling out computations across many machines and increasing the global batch size gave remarkable speed up, bringing training time down from approximately 8 minutes per epoch to 2 minutes per epoch, where epoch refers to one full forward pass and one full backward pass of all the examples in the training set. With an average convergence rate of 130 epochs, we bring overall model training down from 17 hours to nearly 5 hours, now well within FINK's window of opportunity for retraining new models nightly. Note that this is a full model retrain, and simple fine-tuning can be done at a fraction of the time should this be the preferred method of model updating. By leveraging data parallelism in this way, where the data is distributed across multiple devices for training, should even more hardware accelerators become available, we can bring this down further still with a *near*-linear reduction in training time.

5.4.2 Local Processing Tests

With the knowledge that we can retrain models quickly within the specified window suitable for FINK nightly updates, we now move

[16] pypi.org/project/model-profiler

to our first application of deep model compression. We discussed in Section 5.3.2 that of the three methods: weight-clustering, weight-pruning, and weight-quantization, only clustering is applied during training. To enable this, we modify our original model to allow for clustering of weights in the network.

Clustering, otherwise known as weight-sharing, can indeed be done throughout the network. However it is advisable to avoid highly specialised layers such as attention blocks, and only focus clustering on the layers that are likely to have high redundancies. For this reason, we only apply clustering to the final dense layer, as shown in Figure 5.3.

Firstly, we inspect the impact weight-clustering alone has on the model performance compared to the original `t2` architecture. Along with application of model optimisations and compression, comes the expectation that model performance could be adversely affected. While it may be the case that the goal is to remove redundancies, by using these methods, there is inherent information loss compared to the original model, which must be considered. Under the same parameter settings and conditions as the original time-series transformer model (see Allam, Jr. and McEwen (2021), Table 2), which used the six passbands of u, g, r, i, z, y plus two additional features photometric redshift and photometric redshift error, we can see in Figure 5.7 that by using clustering, we not only preserve model accuracy, but this is improved slightly to a logarithmic-loss of 0.450 compared to 0.507 previously. This actually takes the performance beyond the previous state-of-the-art by Boone (2019) of 0.468, whilst maintaining good purity of 0.95 for SNIa and a core-collapse SNe (SNe Ib/c and SNe II) cross contamination of $\sim 4.65\%$, comparable to results calculated for DES (Vincenzi et al., 2021) and Pan-STARRS (Jones et al., 2018) with $\sim 8\%$ and $\sim 5\%$ respectively. However, at this point, we should note that we are potentially seeing the benefits from batch size enhancements and so slight improvements in logarithmic-loss could perhaps be attributed to this, in addition to weight clustering.

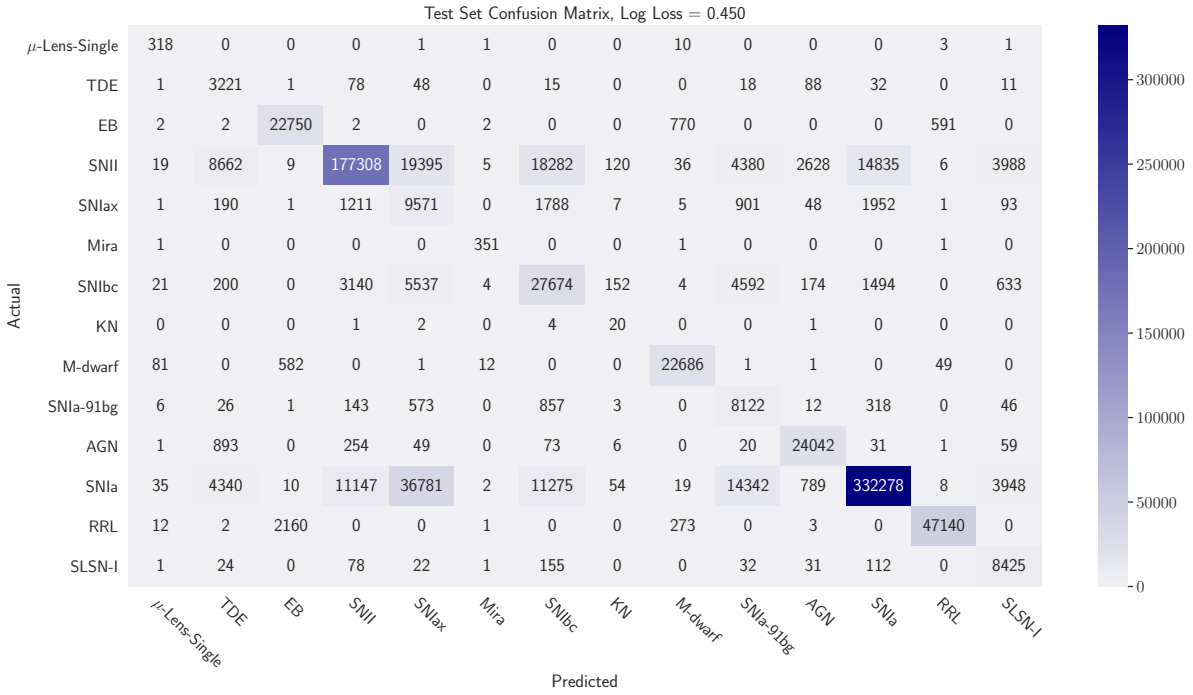


FIGURE 5.7: Raw count confusion matrix resulting from application of a *clustered* version of the time-series transformer (Allam, Jr. and McEwen, 2021), to the PLAsTiCC dataset in a representative setting with imbalanced classes, achieving a logarithmic-loss of 0.450, using all 6 passbands and additional information of redshift and redshift error. This results in a purity of 0.95 for SNIa and a core-collapse SNe (SNe Ib/c and SNe II) cross contamination of $\sim 4.5\%$

Nevertheless, redundancies have been exploited, allowing this model to achieve good classification scores at a fraction of model size on disk. With preservation of model performance confirmed, we continue to explore the impact of applying other compression methods to `t2` and inspect compression rate, inference latency and model performance trade-offs.

To continue our preliminary analysis gearing for deployment, we simulate the ingestion pipeline and use real ZTF alert packet data, such that it is akin to what the model would be expected to handle in the production system of FINK. To synthetically create ZTF-*like* dataset to retrain our model, we use the PLAsTiCC dataset as before, using only the time-series information *i.e.* no additional redshift features, and drop all passbands except for g and r. We then retrain to create a *new* model that can handle ZTF alert packets, and use this as our baseline, which achieves a logarithmic-loss of 0.968.

In a comparative study, we look at four main aspects when judging machine learning models for production performance: model size, model load time, model inference time, and finally model performance in terms of logarithmic-loss score. It is important to monitor any degradation in performance which would indicate whether a particular technique, or combination of techniques are still worth using. This is all shown in Table 5.1, which compares the baseline architecture of the original time-series transformer described above with various compression methods and optimisation techniques applied.

The first thing to note from Table 5.1 is the immediate space savings one can achieve through standard lossless compression algorithms. The original model is able to be reduced down by 4.5×, which is significant for space savings, but it is also clear that load and inference latency are not affected. For the gains we are hoping for, deep compression methods are required, and this will be the focus of the remainder of this discussion.

The first method we apply is that of weight-clustering. We described in Figure 5.4 the typical ways to initialise the centroids of

the clusters. We opt for using linear initialisation and set the number of clusters to be 16 for the reasons laid out in Han et al. (2015a), which puts poor performance of the other initialisation procedures down to fewer clusters containing large weight values. The immediate effects of clustering show improvements in logarithmic-loss from 0.968 to 0.836. We suspect this may be due to the reduction in the number of parameters, helping to generalise the model as alluded to in LeCun et al. (1989b). We also see a slight reduction in model size, but considering we now have shared weights, the real model reductions are realised when we combine this with Huffman encoding, which uses fewer bits for repeated values. Hence, we see a better reduction in model size using clustering *and* Huffman encoding compared to just using Huffman encoding on the baseline model alone. We would expect far greater compression rates should the technique be applied to more layers than just the final dense layer which goes from 448 unique values to 16 values. Models which have many thousands of unique values would greatly benefit from this procedure. In addition to space saving, we also see a small reduction in load latency as well as inference latency.

We now move to the application of weight-pruning. Recall pruning removes "unimportant" weights by setting them to zero, and specifically those weights with low magnitudes. We apply this technique network wide so all layers' weights are evaluated and trimmed down accordingly. Through fine-tuning of the clustered network, we soon discovered that any additional sparsity that was introduced negatively affected the classification scores. The results can be seen in Table 5.1, where even pruning to a level of 1.1% sparsity degraded performance. Further empirical studies are necessary to determine what level of sparsity would actually benefit this network. Notwithstanding, we complete our analysis by including Huffman encoding to this pruned network to witness any improvements in load time and inference latency. Indeed there is an improvement on both of these metrics, but at the cost of classification performance, we disregard using pruning any further.

We move towards a different approach for model optimisation

with a change in file format and framework as described in Section 5.3.4, as well as application of weight-quantization. These results are denoted by the dagger (†) symbol in Table 5.1. These are models that have originally been trained using the full TensorFlow framework but converted to a more optimised, efficient file format of `FlatBuffers`. The conversion also involves operator fusion, to combine primitive computations that appear as a single operation in the computational graph. With reduced code footprint through operator fusion, and efficient binary representation of data, we naturally see a large reduction in model size on disk. Around $10\times$ space savings can be achieved by simply converting the original model into a TFLite model. In addition to the impressive space saving gains that are made, with the model in this format we can take advantage of directly mapping the model into memory for a reduction in load latency of more than $13,000\times$ speed up compared to the exact same clustered model and almost $15,000\times$ that of the original baseline. As the model is loaded for each batch of data FINK processes, this should lead to a fair increase in potential throughput of alerts. While this would certainly help with throughput of alerts in the production system, the other key metric for success is inference latency. That is, the time go from alert packet ingestion to classification. It can be seen in Table 5.1 using the clustered model in the `FlatBuffer` format gives a speed up of around $5\times$ that of the same clustered model, and around $7\times$ speed up compared to the original baseline. Considering our model is expected to process millions of alerts per night, having inference latency gains of this magnitude is undoubtedly positive.

Finally, we apply the third technique described in Section 5.3.2 of quantization. To use this method, we leverage the functionality that comes with TFLite's model conversion tool, that allows for static quantization to 8-bit integers by examining the dynamic range of the weights when saving model to disk, and then upscaling to floating point approximation at inference time. By quantizing the weights of the clustered model, and saving in `FlatBuffer` format, we are able to shrink the model even further to now 60 kilo-

bytes, an 18× reduction when compared with the original baseline model, and an incredible load time improvement of 24,000× speed up. Moreover, inference latency is reduced slightly compared to the clustered model saved in `FlatBuffer` format, for an overall gain of nearly 8× against the baseline. An important point is to mention the preservation of model score with a logarithmic-loss of 0.834. Note the slight improvement in performance here compared to the clustered model without quantization. We suspect this discrepancy between the other clustered models to be due to the scaling approximation in Equation 5.3 and not due to the application of quantization itself.

The confusion matrix in Figure 5.8 shows the performance of this quantized-clustered model trained on only 2 passbands. We calculate a purity of 0.89 for class SNIa, with a core-collapse SNe cross-contamination of 8.15%, which is just outside the expected range of 8% described for DES (Vincenzi et al., 2021) and ∼ 5% described in Jones et al. (2018) for Pan-STARRS.

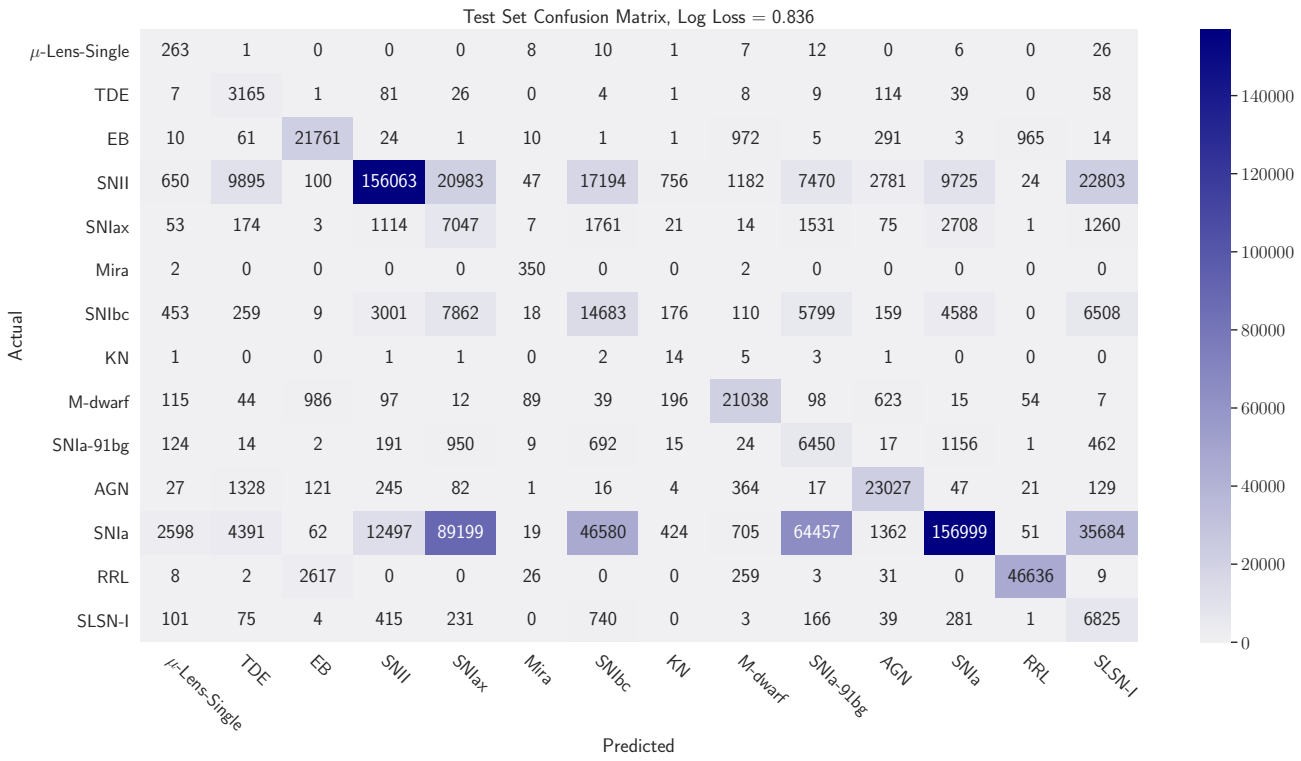


FIGURE 5.8: Raw count confusion matrix resulting from application of a *clustered* version of the time-series transformer (Allam, Jr. and McEwen, 2021), to the PLAsTiCC dataset in a representative setting using full light curves, with imbalanced classes, and only time-series information from g and r passband filters. This model achieves a logarithmic-loss of 0.836, using only the 2 passbands and no additional information. This yields a purity of 0.89 for SNIa and a cross contamination from core-collapse SNe of approximately 8%.

We have shown that application of compression techniques and use of appropriate file formats, substantial space and memory savings, alert processing throughput, and inference latency can be achieved. However, we acknowledge local tests of the pipeline, while on real data, may not be indicative of how well a model would perform in a real production systems, under real-time constrains. Therefore, in the next section we put forward our best performing model that uses a combination of clustering and quantization to be deployed in a live setting on the production system of FINK for tests of real-time classification.

TABLE 5.1

Comparative performance between the original time-series transformer model, referred as the baseline, and the respective compressed versions using a combination of weight quantization, weight clustering, weight pruning and Huffman encoding[17] . We present two sets of results in terms of models saved to disk in `ProtocolBuffer` format and those saved in `FlatBuffer` format, where the latter is denoted by a † symbol. Load latency refers to the time (in milliseconds) to simply read the model into memory, whereas inference latency (in seconds) tests the time to run predictions on a single ZTF alert packet. All tests were run on an Apple M1 Pro 32GB laptop.

Compression Method	Model Size (kb)	Load Latency (s^{-3})	Inference Latency (s)	Loss
Baseline	1100	6324.145	0.333	0.968
Baseline + Huffman	244	6015.565	0.224	0.968
Clustering	892	5559.868	0.227	0.836
Clustering + Pruning	688	5721.021	0.230	1.017
Clustering + Huffman	240	4991.857	0.223	0.836
Clustering + Pruning + Huffman	128	5251.288	0.228	1.017
†Clustering	92	0.426	0.046	0.836
†Clustering + Quantization	**60**	**0.271**	**0.043**	**0.834**

5.5 Production Results

In the previous section we spoke of creating a new model that can classify ZTF alert packets and hence be usable as a science module within FINK. Comparing that to the original time-series transformer of Allam, Jr. and McEwen (2021) which worked with 6 photometric passband of PLAsTiCC: u, g, r, i, z, y, as well as imputing additional features of photometric redshift, we train a model derived from the time-series transformer architecture that only takes in raw time-series from g and r bands, with no additional features. This is done to fit with the data that comes from ZTF alerts packets into FINK, which do not contain photometric redshift information but only time-series measurements for the two passbands of g and r filters. To give an easier visual comparison of the relative performance between the model described above, trained on only 2 passbands, with the model trained on 6 passbands plus additional information, we present the two confusion matrices of Figure 5.7 and Figure 5.8 in a normalised form in Figure 5.9. It was then this gr-only model that was used as the baseline, shown in Table 5.1. While the model is still able to make good classification scores across the board, removing the other passbands of *u, i, z* does cause an increase in cross-contamination by $\sim 4\%$. This is interesting in its own right, where an avenue of research could lead to investigate why training on only g and r passbands affect supernovae classification in this way compared to when we can use all 6 passbands. This may well be down to the lack of i-band specifically as this band along with r-band is typically given preference in times of good seeing and at low airmass (Abell et al., 2009), but for our purposes, we just note these results to keep in mind when assessing model validation.

[17] As described in Section 5.3.2, we actually use the DEFLATE algorithm (Deutsch, 1996) within `zlib`[18] which combines Huffman encoding with the LZ77 compression algorithm (Ziv and Lempel, 1977).

Then, when using a combination of weight-clustering along with weight-quantization saved in the more efficient format of `FlatBuffers`, we created the best performing model in terms of latency and space saving metrics when tested locally on ZTF alert packet data. It should be mentioned that we only train a model of this form to suit deployment into FINK (that at the time of writing, can only ingest ZTF data), for testing of our model as a real-time classifier. It is expected for LSST that a model more akin to that showcased in Figure 5.7 that uses all 6 passbands as well as redshift information but trained on discretised alerts would be deployed.

This section presents the results for deploying our *quantized-clustered* version of the time-series transformer model into the production system of FINK tested on the real ZTF alert stream. We compare the baseline model described in the previous section, that achieves 0.968 logarithmic-loss on ZTF-like data packets, with the compressed version, that achieves better logarithmic-loss of 0.834, in a now live production setting, and observe alert throughput and latency improvements that have been achieved when using the deep model compression techniques.

5.5.1 Model Validation

The first test for our deep learning model as a science module within FINK is to validate the classification scores that we achieve. Not only is it important for our model to operate in real-time under heavy work-load conditions, but it must clearly continue to report correct classification results when deployed. FINK validates models using the Transient Named Server (TNS) (Gal-Yam, 2021), which is a transient alert system that has spectroscopically confirmed objects in its database. By comparing predictions to that of what TNS lists for a given object, we can get an indication of how well a transient classifier is performing.

Figure 5.10 shows our models predictive performance on one full year of real ZTF alerts against the spectroscopically confirmed

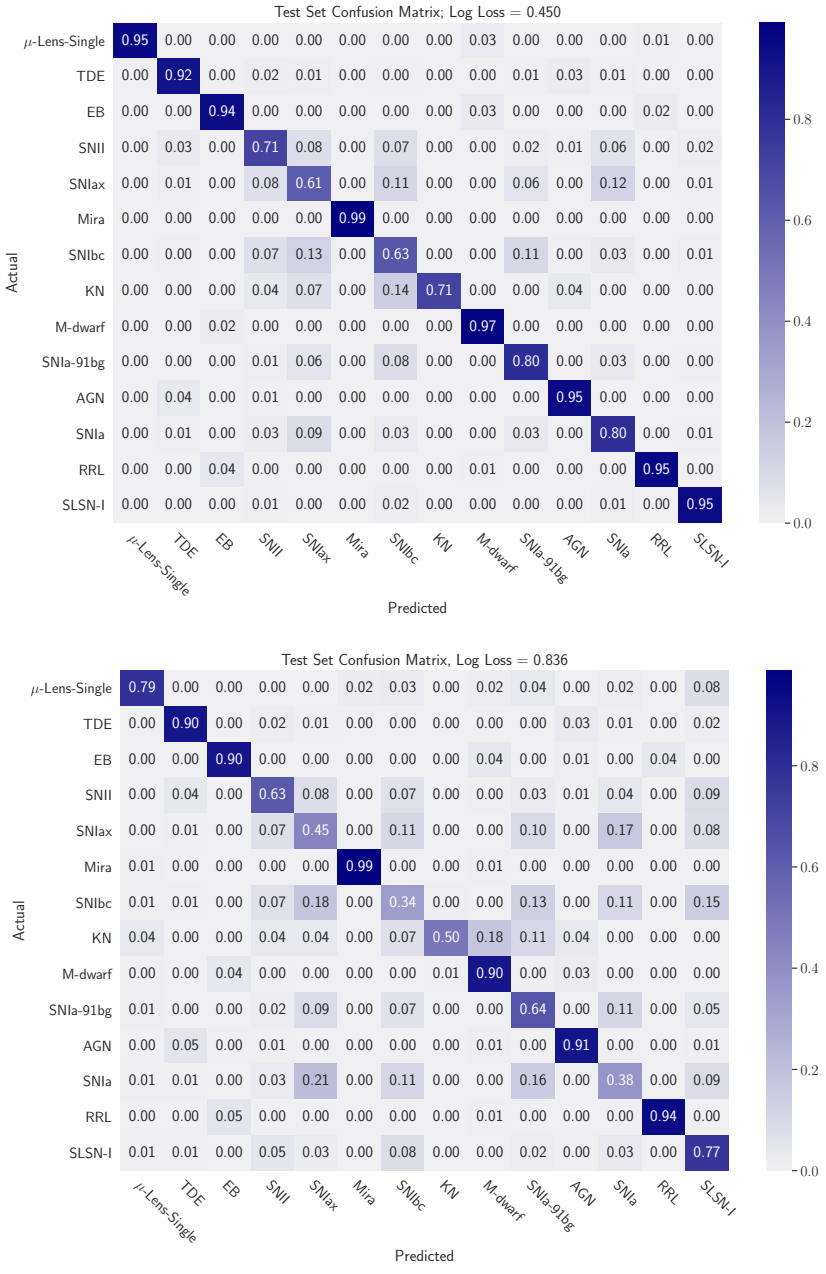


FIGURE 5.9: Normalised confusion matrices for Figure 5.7 above and Figure 5.8 below.

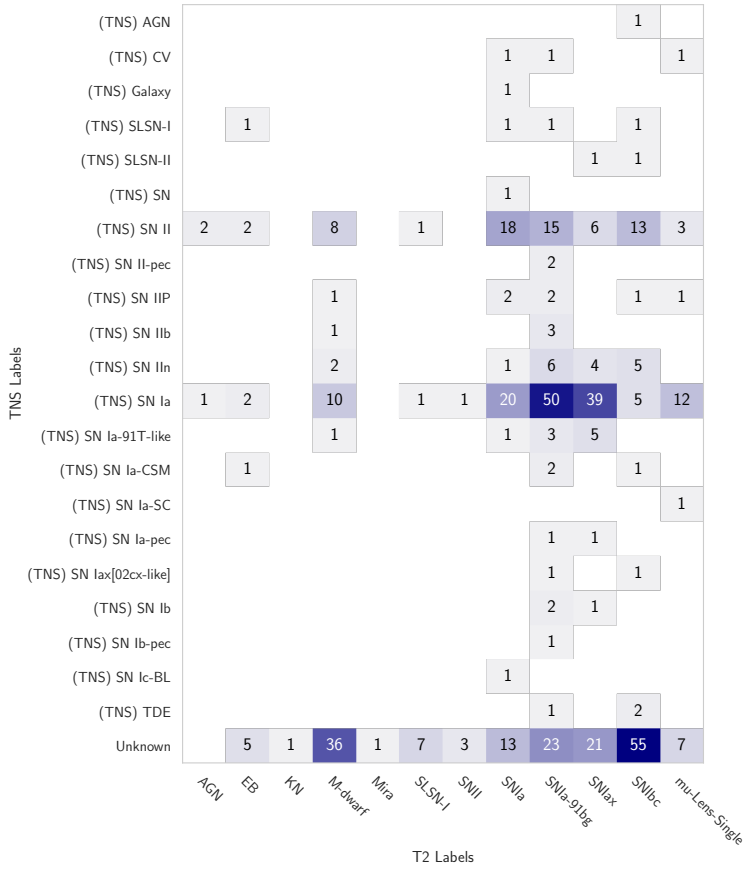


FIGURE 5.10: Comparison of spectroscopically confirmed labels in the Transient Named Server (TNS) database against the top-1 predictions for the compressed time-series transformer. The data used for this test comprised of one full year (2022) of real ZTF data with specific quality applied to the light curve history data requiring a minimum of 2 points and maximum of 90 points on the light curve since the first alert emission date. The set of alerts are also reduced to filter out objects known to be a Solar System object from the MPC database or Galactic object when cross-matched against the SIMBAD database.

objects in the TNS database (extra-galactic objects). In addition to the quality cuts common for all modules, we apply further criterion of at least 2 points and at most 90 points on the light curve since the first alert emission date, as well as the object to not be a Solar System object from the Minor Planet Center (MPC) database or Galactic object according to the SIMBAD database (Wenger et al., 2000). Recall in 5.4.2 our baseline model has been trained using full light curves. Since real ZTF alert packets contain only 30 days history, only partial light curves are observable most of the time. This results in a model being trained in non-representitive setting where it expects full light curves but is tested on discretised light curves. Yet, our model is able to correctly identify the majority of SNIa objects, as well as other classes. Though it should be noted there is greater misclassification of SNIa and core-collapse SNe beyond the predicted cross-contamination of $\sim 8\%$ described in the previous section, and this is likely due to the non-representative nature of the alert data. As such, we would not consider our model in this form to be suitable for a fine-grain transient classifier, and ideally would need to be trained on the discretised data of alert packets to be more representitive, which is planned for future work. Considering these results, we can instead frame our model as to be a general transient classifier that is able to identify SNe more broadly. Indeed, when we evaluate the model against the ensemble of predictions from all other classifiers in FINK we can see our model is able to correctly identify SN candidates, shown in Figure 5.11.

Therefore, while there are some misclassification within SNe classes, the compressed time-series transformer is able to successfully classify supernova objects in general, and when compared to existing science modules in FINK, correctly identifies supernova candidates. It is also able to go further, where FINK science modules labels certain objects as "Unknown", our model is able to accurately suggest these as supernovae *candidates* (see Figure 5.11). A clear value-add for the brokering system which can be used to update and enrich the database.

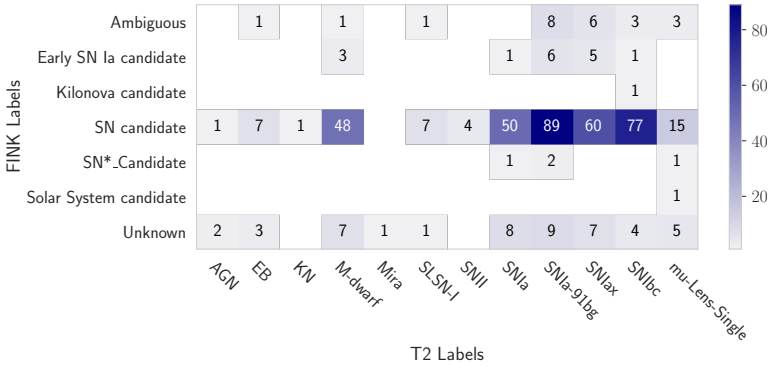


FIGURE 5.11: Comparison of aggregated FINK classifiers' predictions against the top-1 predictions from the compressed time-series transformer. One full year (2022) of real ZTF alert data is filtered according to the respective classifiers quality cuts as well as the criteria for a minimum of 2 points and maximum of 90 points on the light curve since the first alert emission date. We also ensure to disregard alerts that correspond to be a Solar System object from the MPC database or Galactic object when referenced against the SIMBAD database.

5.5.2 Alert Throughput/Latency Performance

With confirmation that the compressed model is operating correctly, we now come to test the throughput and latency of the classifier, ultimately deciding the usefulness of our model for real-time classifications.

We first look at how the original time-series transformer, which we refer to as the baseline model in Table 5.1, fairs up against other existing science modules in FINK. For this, we take one full nights worth of ZTF alerts, amounting to approximately $200,000$ alert packets, and compare the throughput and latency performances of all the science modules currently implemented in `fink-science` [19]as of `v2.0.0`. For an overall comparison, we show the "on-sky" throughput performance that passes all alerts through all science modules, ignoring any pre-processing filters that would normally be applied.

Over an average of 20 processing runs, the mean alerts per second per core for each science module is calculated. These results are most succinctly presented in Figure 5.12, where our model is the only deep learning model of such complexity offering up a vector of probabilities for the classification scores. Other science modules such as Solar-System Object (SSO) and The Centre de Donnés astronomiques de Strasbourg (CDS) cross matching service are examples of table lookups whose performance is determined by the execution of a query plan, and the only other deep learning model of `SuperNNova` (SNN) (Möller and Boissière, 2020) offers only a binary classification for SNIa.

The baseline model, with no compression or optimisations made, is actually able to sit amongst the other science modules and deliver real-time classifications. While this seems to have already achieved our desired goal of deploying a science module capable of real-time classification, it is important to consider that the model that is deployed in FINK is not done so in isolation, but rather all science modules within FINK will be operating in tandem. Cor-

[19] github.com/astrolabsoftware/fink-science

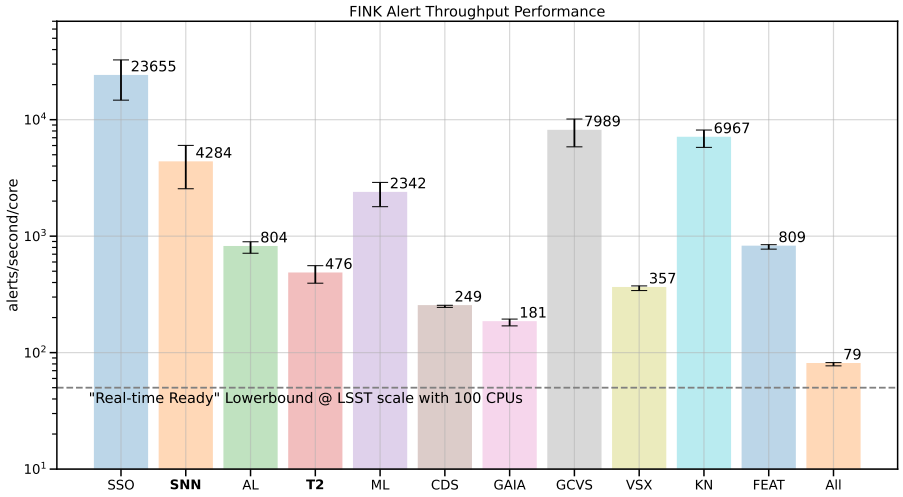


FIGURE 5.12: Alert throughput of FINK science modules, as of version `fink-science-v2.0.0`, tested on one full nights worth of ZTF alerts (approximately 200, 000). Our time-series transformer model, referred to as the baseline model in Table 5.1, and labelled here as "T2" is able to achieve a throughput of $\sim$ 500 alerts per second per core. As described in Möller et al. (2021), we trace a horizontal line to indicate the threshold for a single science module within FINK to be considered *real-time ready*, assuming 100 CPU cores. Under the data rates estimated for LSST, FINK will receive 10, 000 alerts every 37 seconds, and such a threshold would allow for approximately a dozen science modules to provide classification scores serially. For full details of the inner workings of the other science modules shown here, the reader is advised to explore the `fink-science` package. It should be noted that the results presented here are to be considered "on-sky" performance, where all alerts are processed blindly by all modules. This would not be the case typically, for example only a small fraction of alerts would be processed by the ML (microlensing) (Godines et al., 2019) module since various inbuilt physics filters would determine if an alert is suitable for processing beforehand. Highlighted in bold are the two deep learning models, with one being our own time-series transformer listed as (T2) and the other being that of (`SuperNNova`) (Möller and Boissière, 2020) listed as SNN. With a key difference being that SNN is a binary classifier whereas T2 provides a set of probabilities scores across many classes.

respondingly, since the outgoing enriched alert packet is only sent after *all* science modules have finished processing the data packet, each individual science module can have a large impact on the real-time science capability of FINK as a whole. To not delay other modules that may be time-critical such as for Gamma Ray Burst (GRB) detections, science modules need to optimise throughput wherever possible, ultimately benefiting the entire system.

Therefore, by going further and looking at how our best performing compressed model manages to deal with the alert throughput in a live setting, we can see in Figure 5.13 a sizeable improvement. While our local processing tests gave up to 8× speed up compared to the baseline model for inference latency, in a real production environment, we achieve an impressive 5× in a live setting. It is suspected that a decline in speed up compared to what was achieved in a local processing context can be attributed to communication overheads in the cluster, where networking bandwidth becomes the bottleneck in place of computations.

This substantial throughput performance, thanks to low-latency inference via deep compression techniques, greatly benefits the overall FINK system. As science modules are run serially in FINK, our models ability to quickly complete processing not only ensures there is no delay to other time-critical science modules, but also permits more science modules to co-exist within the total computational time budget afforded to FINK. Finally, by improving latency in this way, we lay out a guide for other existing deep learning models, and those under current development, for how to use model optimisations for improved performance.

5.6 Conclusions

We have shown through deep model compression, complex models such as the time-series transformer can be made super-lightweight

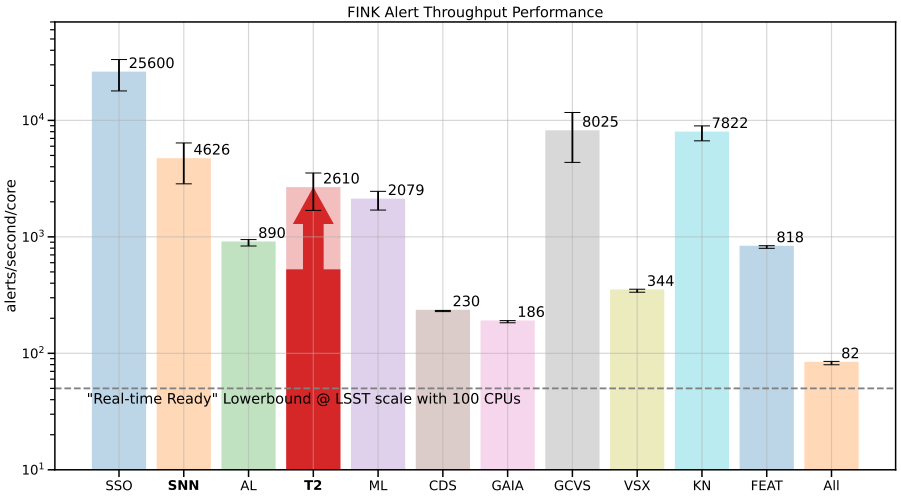


FIGURE 5.13: Our time-series transformer model (T2), is originally able to achieve a throughput of $\sim$ 500 alerts per second per core, but following application of deep compression techniques achieves an increased throughput of approximately 5$\times$ of the number of alerts to now $\sim$ 2600 alerts per second per core. In this same plot we have shown the baseline performance, and the new compressed version of the time-series transformer architecture using an arrow to indicate the increase in throughput. This version of the time-series transformer that uses weight-clustering and weight-quantization along with TFLite fused operations achieves performance far beyond the requirements for real-time classification of alerts at LSST scale. This has knock-on benefits for all other science modules within FINK, and encourages use of these methods for other science modules going forward. We highlight in bold along with T2 the only other deep learning model of SNN (`SuperNNova`) (Möller and Boissière, 2020). The increase in throughput performance brings it within the same order-of-magnitude of alerts able to be processed, yet we are able to provide probability scores across all classes as opposed to a single binary classification.

for real-time inference. The already efficient architecture benefits even further from weight-clustering and weight-quantization to provide low-latency, high-throughput classification scores, all the while preserving the accuracy of the results. Our study of weight-pruning showed good reduction in model size but proved to be detrimental to performance. Clearly even the low magnitude weights of the network carry information critical for good classification.

We have shown through careful choice of file formats, major speed ups can be achieved, which in turn dramatically improves a deep learning model's ability to process inputs and operate in real-time, in a live production setting. Our compressed version of the time-series transformer now resides in FINK, providing nightly classifications for the incoming ZTF alert stream. We have showcased our models suitability for providing robust classifications at a fraction of the original model size and runtime. By scaling out computations, we have brought retraining down to within the time frame required for nightly updating on new alert data.

As described in Section 5.2.2, the ZTF alerts stream, although $1/10^{\text{th}}$ of the expected LSST data rates, is a good precursor for modelling the suitability of models and infrastructure to how well they will handle future data streams. Consequently, we used FINK to emphasise our model's ability to handle such large volumes of data and have presented results that showcase its ability to cope with LSST scale, and beyond.

It is hoped that the work here, which introduces deep compression to the field of real-time transient classifiers, will be harnessed to enable existing architectures to be deployed as real-time classifiers easily into other brokering systems, as well as to inspire those currently being developed that real-time capability is within reach if techniques like those described here are applied.

6

Conclusions

"*We can only see a short distance ahead, but we can see plenty there that needs to be done.*"

— **Alan Turing.**

6.1 Research Summary

This thesis has introduced two novel deep learning architectures for photometric classification of astronomical transients, that are also applicable to the task of general multivariate time-series classification. Specifically designed to be fast and efficient, with computational cost and algorithmic complexity in mind from the outset, we presented architectures that go beyond the current state-of-the-art for photometric classification in terms of classification performance and greatly improves computational efficiency.

Our first architecture, the astronomical-transient xception network, `atx`, (Chapter 3), introduced the depthwise-separable convolution to the field of transient classification for a low-rank factorisation of the normal convolution operator. This brought computational costs down compared to other CNN architectures by $\mathcal{O}(1/N + 1/w)$, where N is the number of output channels and w is the width of the kernel in a convolution, as well as reducing number of parameters per layer from $\mathcal{O}(w \cdot d^2)$ to $\mathcal{O}(w \cdot d + d^2)$, with d representing the dimensionality of the output. The idea for

such a network was built upon from the investigations by Fawaz et al. (2019) which showed how state-of-the-art architectures typically applied to computer vision tasks on 5-dimensional images (2 spatial dimensions with 3 channel dimensions), can be naturally applied to time-series data which is simply 1-dimensional "spatio" data with number of channels equal to the number of features.

As put forward by Chollet (2017), taking an Inception module to the *extreme*, one can achieve significant computational savings using the depthwise-separable convolution in place of normal convolution. For that reason, we took the InceptionTime architecture developed in Fawaz et al. (2019) as inspiration to form an "XceptionTime" architecture centred around depthwise-separable convolutions for efficient multivariate time-series classification. When specifically adapted for the task of photometric classification this architecture became the network described in Chapter 3 as `atx`. Using this architecture we achieved a logarithmic-loss of 0.739, where the leading result in the field under the same metric is 0.468 (Boone, 2019).

To go even further and improve our achievements with `atx`, we changed our focus of inspiration from one deep learning subdomain of computer vision, to another of natural language processing (NLP). By making the connection between sequence modelling of NLP to that of photometric light curve data, we were able to leverage major advancements in architectural design from the NLP world to help bring computational cost and parameter count down even further than before, all the while improving classification performance. The time-series transformer, `t2`, described in Chapter 4 is a fast and efficient architecture, that reduced model size by 45× along with 3× reduction in per epoch training time, when compared to our already computationally efficient astronomical-transient xception network.

Our work of Allam, Jr. and McEwen (2021), that drew inspiration from the breakthrough paper of Vaswani et al. (2017), introduced transformers and the multi-head self-attention mechanism to the field of photometric classification. We also pioneered the use

of a new *convolutional embedding* that can project a photometric light curve, or more generally a multivariate time-series, into a vector-space representation suitable for modern machine learning architectures. The smaller and more lightweight model was able to achieve even better performance on the PLAsTiCC dataset with near state-of-the art logarithmic-loss of 0.507.

Our final contribution to the field laid out in this thesis has been the introduction of deep compression techniques for the deployment of deep learning models for real-time transient classification. With an implementation of these methods on top of our best performing architecture of the time-series transformer, we demonstrate our deep learning models' ability to not only handle live streaming data from the Zwicky Transient Facility (ZTF) under stress, but to accurately make predictions in real-time.

Through performance engineering and model optimisation methods, we were able to bring our already performant deep learning model of `t2` into the realm of high-throughput low-latency inference necessary for LSST scale. Now with a model size of a mere 60 kilobytes on disk, we are able to achieve a throughput capacity of around 2600 alerts per second per CPU core, ultimately improving the throughput capacity of the entire brokering system as well. Moreover, by leveraging data parallelism and modern hardware accelerators, we bring retraining of our deep learning model to within the critical window for nightly updates on new data, fitting with one of the unique selling points of FINK.

Our tests show that our compressed version of the time-series transformer is already capable of handling the influx of data that is expected from LSST when it comes online. We also show that by using all six passbands that will eventually be available, along with photometric redshift and redshift error, a quantized and clustered model can achieve beyond state-of-the art multi-class classification of astronomical transients with a logarithmic loss of 0.450 compared to 0.468 of the previously leading model (Boone, 2019).

6.2 Future Work

As is the nature of research: *we see there is always plenty to be done*, but in this section we identify avenues of particular research that would either naturally follow from the work presented here in this thesis, or offer an alternative approach to achieving good classification performance at a reduced computational cost, as well as improved downstream efficiency of resource allocation.

6.2.1 ELAsTiCC

Such was the success of the Photometric LSST Astronomical Time-Series Classification Challenge (PLAsTiCC), the dataset for which all our classification results are based, a renewed effort with improved data is coming in the form of the *Extended* LSST Astronomical Time-series Classification Challenge (ELAsTiCC) (Naryan et al., 2022). However, changing focus slightly, the purpose of ELAsTiCC is not only to attract novel machine learning classifiers from the community at large, but to formulate a dataset suitable for testing end-to-end real-time pipelines of brokering systems in preparation for LSST. To do so, the ELAsTiCC team has begun to simulate 5 million detected events that are discretised into 50 million alerts, which will be streamed from LSST to the brokers listed in 5.2.1. They will use this stream of alerts to stress-test infrastructure and provide their packet enrichment services along with a classification of each alert back to LSST

Participation in ELAsTiCC by way of our integration with the FINK brokering system would be an obvious next step for our research. What is more, depending on how the ELAsTiCC dataset is constructed, *i.e.* whether the training set will be representative or not, this may be an opportunity to explore use and integration of data augmentation techniques within our models. We have already shown that by using data that is in a similar format to LSST with Zwicky Transient Facility (ZTF) alerts, our deep learn-

ing models can be adapted to handle real-world data arriving at high velocity in a streaming fashion. We empirically showed that using all six passbands available from LSST along with photometric redshift and the associated error, our lightweight version of the time-series transformer can achieve beyond state-of-the-art classification scores, and at a throughput level above the critical line for LSST data scale.

As the goal of this challenge is not to rank brokers on their classification performance, nor to showcase any classifiers in particular, we do not expect the results of this challenge to praise a particular architecture. Having said that, a classifier that can prove to be reliable under the same conditions as will be in place when LSST would further strengthen our research presented in Chapter 5, and hopefully highlight the benefits our methods have to offer to the transient classification community.

6.2.2 Probabilistic Machine Learning

In this thesis, we have stressed the importance for reliable classification, especially when it comes to further resource allocation of follow up observations and estimation of cosmological parameters. While we currently output a vector of probabilities normalised across the classes, with the classification label being assigned to the arguments of the vector maxima, one would ideally want a distribution over the classes that contains an intrinsic measure of uncertainty for better cosmological parameter estimation and decision making. As these photometric classifications are used for cosmology directly, a measure of uncertainty of each prediction can help to better constrain competing models of dark energy. Furthermore, probabilistic predictions can also effectively determine whether a particular object is worth following up or not, irrespective of it simply being the maximum class among the other classes. Consequently, we could reduce expensive false-positive follow-ups by enforcing a threshold of classification confidence before any further resources are sought.

So far, with all architectures we have described, we have used point estimates for the weights in the networks. To achieve the uncertainty measures we would like, a probabilistic extension to the deep learning architecture would be necessary. By moving away from the point estimates of our weights to instead drawing the weight values from a distribution we can better quantify the uncertainty of our predictions.

However, the motivation of this thesis has been to develop novel, efficient deep learning architectures that can provide reliable classification in *real-time*. At conflict with this premise is the notorious computational cost of running probabilistic networks. But, there is light on the horizon. In this golden age of computer architecture and hardware accelerators, coupled with the major advancements in fast optimisation algorithms, there is a resurgence of efforts that leverage Probabilistic Programming Languages (PPLs), that help one define a probabilistic model, on top of computationally efficient hardware. Development of probabilistic architectures with PPLs that use fast algorithms on specialised hardware for the approximation of integrals through optimisation are giving hope for possible real-time use cases in the future. It is certainly felt worth exploring this space to gain better uncertainty estimates on the predictions that are made.

6.2.3 Probabilistic Data Structures

Another alternative avenue of research that could be explored, which also gives a notion of uncertainty, is use of a probabilistic data structure (PDS). PDSs are data structures that typically use a hashing function to provide an approximated answer to a query, but with provable bounds on the errors. The rise of PDS research came in the recent big-data era when fast approximated answers became more appealing than slow deterministic ones when dealing with large data streams (Gakhov, 2019).

The big-data stream that will flow from LSST warrants data structures and algorithms that can run on a single pass of the data

i.e. online, and that can provide a summary (or *sketch*) of the data stream that can be used for data monitoring, general data statistics or even stream filtering of alerts.

We see a potential use case for a PDS that uses locality-sensitive-hashing (LSH) (Indyk and Motwani, 1998) for fast similarity search of embeddings created by our deep learning architectures. Work by Yoon et al. (2015) showed how a time-series *fingerprint* can be generated using a pipeline of transforms (see Figure 5. of Yoon et al. (2015)) and then similar time-series are grouped together into the same hash bucket with high probability using LSH. We see our fast time-series transformer as a drop in replacement for the pipeline of transforms with LSH used to probabilistically group similar light curves. We see this as a fast and scalable way for efficient detection of possible transient events that could sit early in a broker's processing pipeline. For instance, this could act as an early stage supernova candidate finder that is able to flag an alert for more specialised downstream supernova classifiers. Another example may be for anomaly detection, where we would be able to test whether the embedding we create is similar to anything we have ever detected before. If the similarity score is below a threshold we define, we can also flag for the alert to be considered a potential anomaly.

The enticing aspect of using a PDS is the extremely fast query lookups to the data that run in constant time, $\mathcal{O}(1)$ (assuming well constructed hash-maps) and provide an estimated error of the result. We can then use this to inform further time-critical downstream tasks which need a notion of uncertainty when making decisions. Their ability to scale to ever increasing data, yet in sub-linear space make for an exciting prospect for further research, and perhaps a desirable feature for brokering systems to have early in their processing pipelines.

6.3 Closing Remarks

With the Vera C. Rubin Observatory's first-light approaching, preparations for the next generation survey of LSST are ramping up. The work contained in this thesis is part of that effort, putting forward fast and efficient architectures for photometric classification that can operate in real-time. The computational cost savings achieved by our methods, while conserving state-of-the-art classification performance are already benefiting real-world astronomical systems such as FINK (Möller et al., 2021), and it is hoped upon first-light, our compressed time-series transformer will be a standout science module capable of providing accurate classification scores in an efficient and cost effective way.

The machine learning methods described, while specifically designed for the task of photometric classification, can be more generally applied to other domains that have multivariate time-series data. It is therefore expected that the techniques that are set out here will be leveraged by those who wish to not only classify astronomical transients events, but to perform generic time-series classification at low computational costs. Furthermore, we hope to inspire other practitioners developing time-series classification architectures, that real-time deployment of those models are possible through a compression blueprint explained in the previous chapter.

Use of these methods will positively impact the scientific community by improving the automated labelling of transient alerts, where deep learning methods are now critical, and by that virtue help constrain theories of the Universe. Hopefully shedding light onto the age old questions of *where did it all begin, and how will it all end?*.

A

MTS Benchmark Results

The MTS is a set of 12 multivariate time-series datasets released by Baydogan (2015) to encourage efforts towards the development of better time-series classification methods. A review of the state-of-the-art circa 2016 is given in Bagnall et al. (2017), with a more recent overview outlining modern deep learning approaches presented in Fawaz et al. (2019). Shown in Table A.1, A.2 and A.3 are the comparative results for `atx` and `t2` against other leading architectures described in Fawaz et al. (2019) with regards to accuracy, precision and recall metrics respectively. With the task of photometic classification in mind, we compare our architectures to MTS to gauge suitability of such architectures applied to general multivariate time-series data found in the wild.

TABLE A.1

Classification accuracy for 12 multivariate time-series datasets (see Bagnall et al., 2017; Bagnall et al., 2018; Baydogan, 2015, for details) against architectures discussed in Fawaz et al. (2019)

	T2	ATX	CNN	ENCODER	FCN	MCDCNN	MCNN	MLP	RESNET	TLENET	TWIESN
ArabicDigits	97.32	98.50	95.77	98.07	99.42	95.88	10.00	96.91	99.55	10.00	85.28
AUSLAN	92.91	87.09	72.55	93.84	97.54	85.38	1.05	93.26	97.40	1.05	72.41
CharacterTrajectories	94.57	97.97	96.00	97.06	98.98	93.82	5.36	96.90	99.04	6.68	92.04
CMUsubject16	100.00	93.10	97.59	98.28	100.00	51.38	53.10	60.00	99.66	51.03	89.31
ECG	84.00	76.00	84.10	87.20	87.20	50.00	67.00	74.80	86.70	67.00	73.70
JapaneseVowels	97.30	97.03	95.65	97.57	99.30	94.43	9.24	97.57	99.16	23.78	96.54
KickvsPunch	90.00	70.00	62.00	61.00	54.00	56.00	54.00	61.00	51.00	50.00	67.00
Libras	82.78	74.44	63.72	78.33	96.39	65.06	6.67	78.00	95.44	6.67	79.44
NetFlow	86.14	77.90	88.95	77.70	89.06	62.96	77.90	55.04	62.72	72.32	94.49
UWave	84.53	90.95	85.88	90.76	93.43	84.50	12.50	90.06	92.59	12.51	75.44
Wafer	89.40	89.40	94.81	98.56	98.24	65.76	89.40	89.40	98.85	89.40	94.90
WalkvsRun	100.00	75.00	100.00	100.00	100.00	45.00	75.00	70.00	100.00	60.00	94.38

TABLE A.2

Classification precision for 12 multivariate time-series datasets (see Bagnall et al., 2017; Bagnall et al., 2018; Baydogan, 2015, for details) against architectures discussed in Fawaz et al. (2019), where negative results indicate an numerical instability in the calculation.

	T2	ATX	CNN	ENCODER	FCN	MCDCNN	MCNN	MLP	RESNET	TLENET	TWIESN
ArabicDigits	96.79	98.51	95.84	98.10	99.43	95.95	1.00	96.97	99.56	1.00	86.16
AUSLAN	86.19	88.46	76.12	94.72	97.92	87.87	0.01	94.41	97.79	0.01	75.00
CharacterTrajectories	87.14	97.84	96.18	97.11	98.86	93.86	0.27	96.98	98.91	0.33	92.94
CMUsubject16	27.59	93.03	97.50	98.23	100.00	30.60	26.55	39.46	99.71	25.52	89.59
ECG	77.39	41.33	81.87	85.55	85.31	25.00	33.50	65.05	84.91	33.50	70.96
JapaneseVowels	96.09	96.84	95.56	97.33	99.14	94.22	1.03	97.33	99.00	2.64	96.75
KickvsPunch	79.17	69.05	68.19	62.39	52.12	28.00	27.00	58.21	55.19	25.00	67.98
Libras	84.32	74.77	64.15	79.12	96.69	67.17	0.44	79.66	95.84	0.44	81.62
NetFlow	80.58	38.95	84.61	42.78	85.77	45.80	38.95	34.93	69.33	36.16	94.19
UWave	-999900.00	90.46	86.19	90.99	93.42	85.05	1.56	90.70	92.59	1.56	77.38
Wafer	-999900.00	-999900.00	87.89	98.27	96.09	32.88	44.70	44.70	97.95	44.70	97.20
WalkvsRun	37.50	37.50	100.00	100.00	100.00	22.50	37.50	35.00	100.00	30.00	93.05

TABLE A.3

Classification recall for 12 multivariate time-series datasets (see Bagnall et al., 2017; Bagnall et al., 2018; Baydogan, 2015, for details) against architectures discussed in Fawaz et al. (2019), where negative results indicate an numerical instability in the calculation.

	T2	ATX	CNN	ENCODER	FCN	MCDCNN	MCNN	MLP	RESNET	TLENET	TWIESN
ArabicDigits	96.77	98.50	95.77	98.07	99.42	95.88	10.00	96.91	99.55	10.00	85.28
AUSLAN	84.63	87.09	72.55	93.84	97.54	85.38	1.05	93.26	97.40	1.05	72.41
CharacterTrajectories	86.63	97.69	95.66	96.77	98.86	93.48	5.00	96.62	98.91	5.00	91.44
CMUsubject16	50.00	93.03	97.81	98.37	100.00	50.31	50.00	58.13	99.62	50.00	89.23
ECG	77.39	49.23	83.14	85.60	86.53	50.00	50.00	72.27	85.15	50.00	66.53
JapaneseVowels	95.70	96.96	96.21	97.89	99.28	94.26	11.11	97.71	99.23	11.11	97.21
KickvsPunch	79.17	66.67	65.83	62.50	55.00	50.00	50.00	61.25	55.00	50.00	68.33
Libras	82.78	73.33	63.72	78.33	96.39	65.06	6.67	78.00	95.44	6.67	79.44
NetFlow	77.45	50.00	82.59	50.41	81.05	50.21	50.00	50.77	66.20	50.00	89.49
UWave	-999900.00	90.25	85.88	90.76	93.43	84.50	12.50	90.06	92.59	12.50	75.44
Wafer	-999900.00	-999900.00	83.41	94.05	94.56	50.00	50.00	50.00	95.97	50.00	75.99
WalkvsRun	50.00	50.00	100.00	100.00	100.00	50.00	50.00	50.00	100.00	50.00	95.42

B

PLAsTiCC Data Samples

Here we present example light curves built from the The Photometric LSST Astronomical Time-Series Classification Challenge (PLAsTiCC) using techniques laid out in Chapter 3. Each colour represents one of the 6 passband filters that will be used for LSST; *u, g, r, i, z, y.*

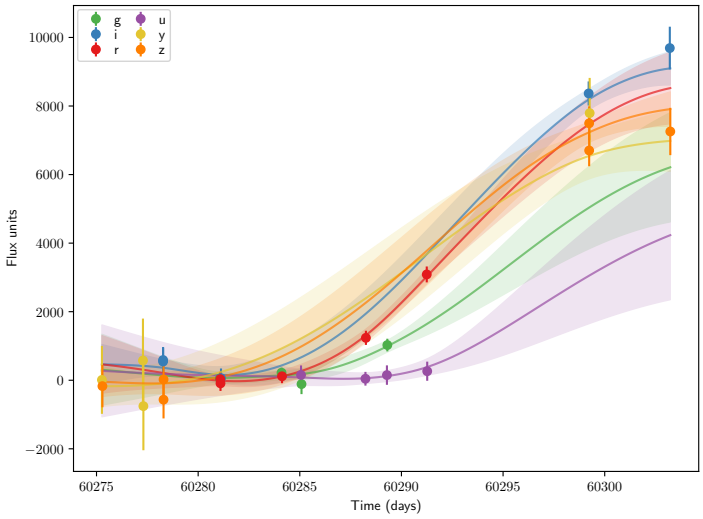


FIGURE B.1: SNIa

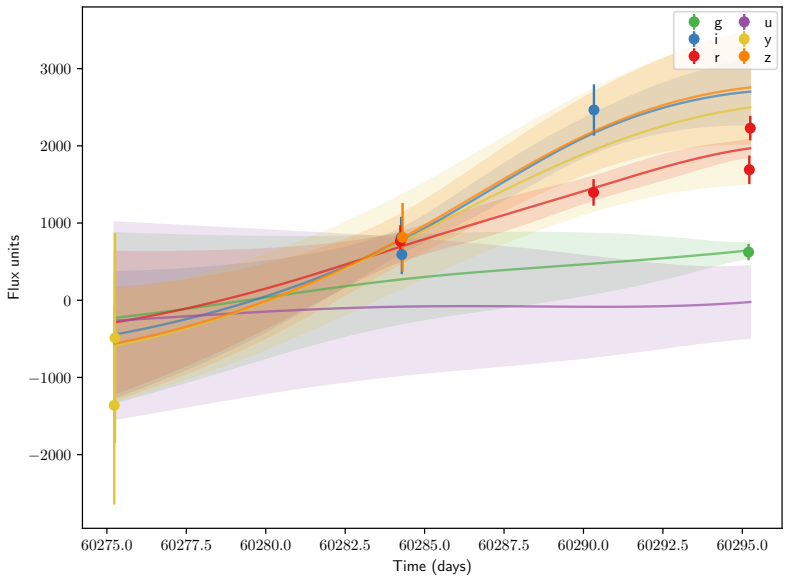


FIGURE B.2: SNIbc

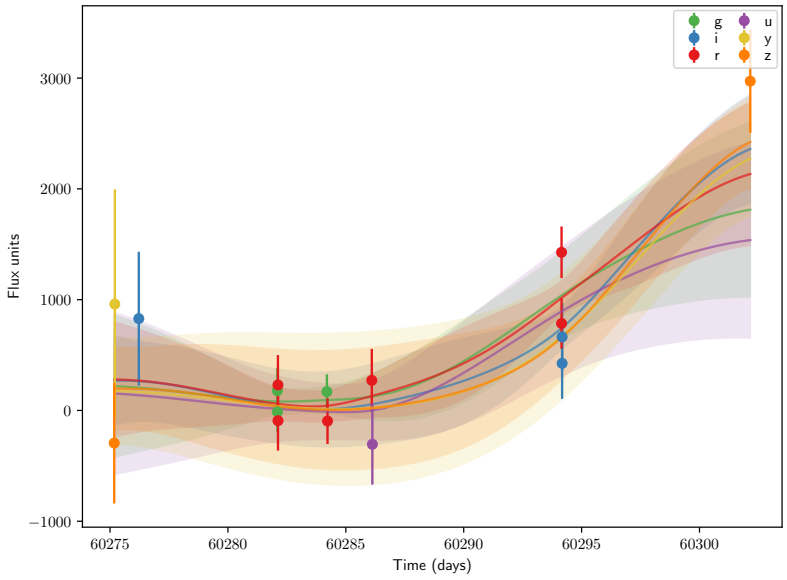


FIGURE B.3: SNII

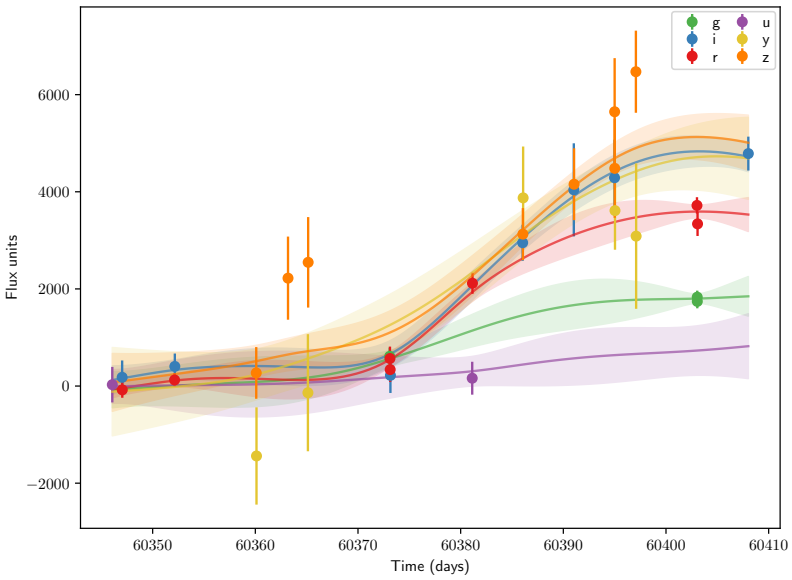


FIGURE B.4: SNIax

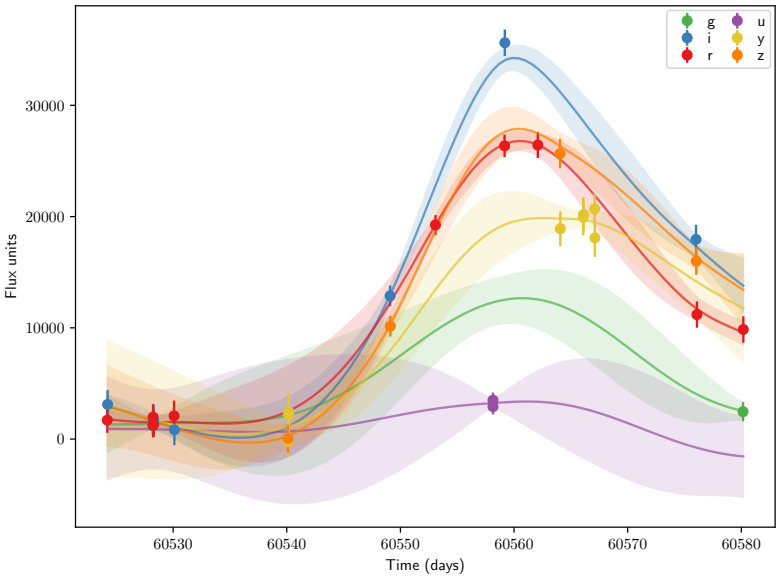


FIGURE B.5: SNIa-91bg

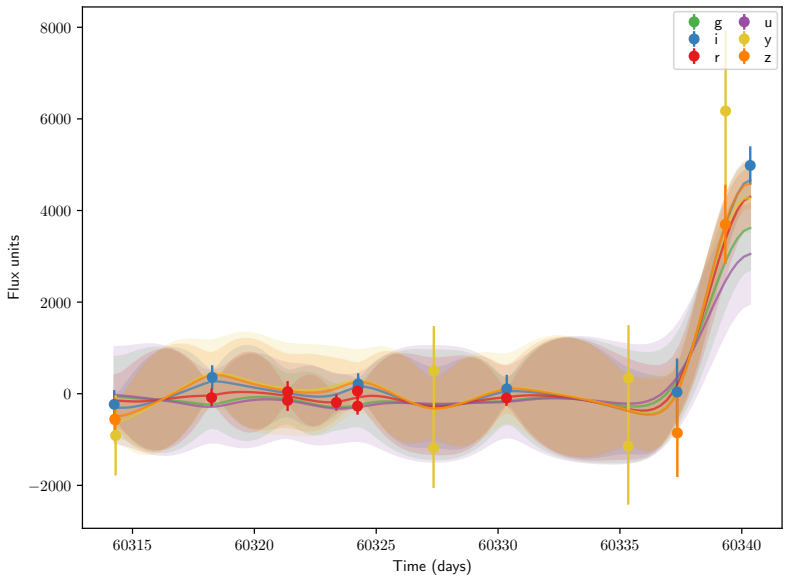


FIGURE B.6: Kilonovae

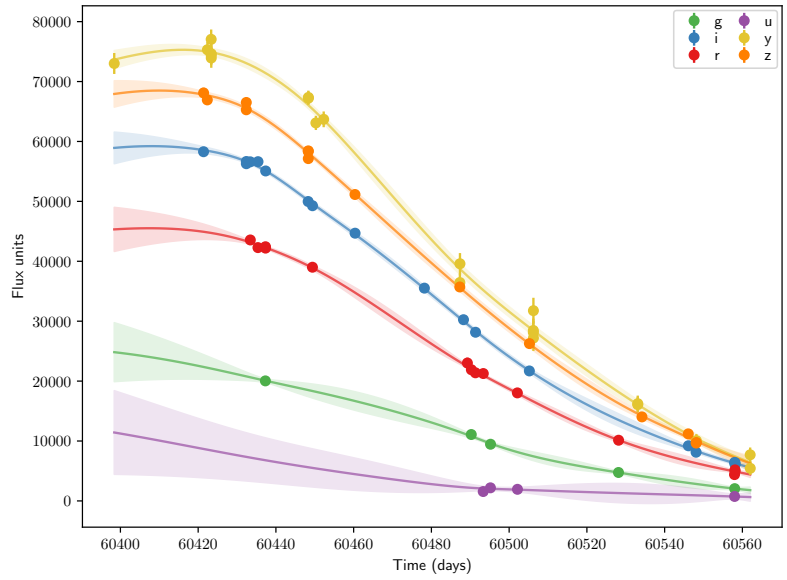


FIGURE B.7: μ-Lens

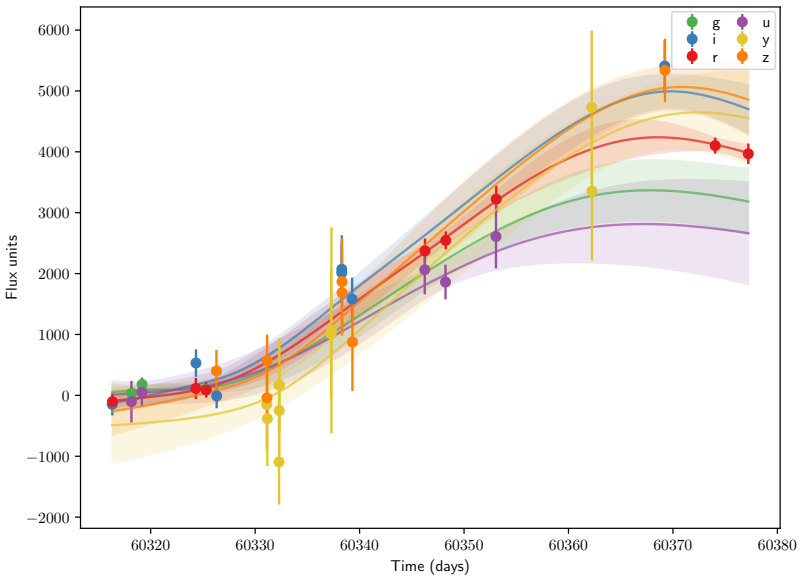


FIGURE B.8: SLSN

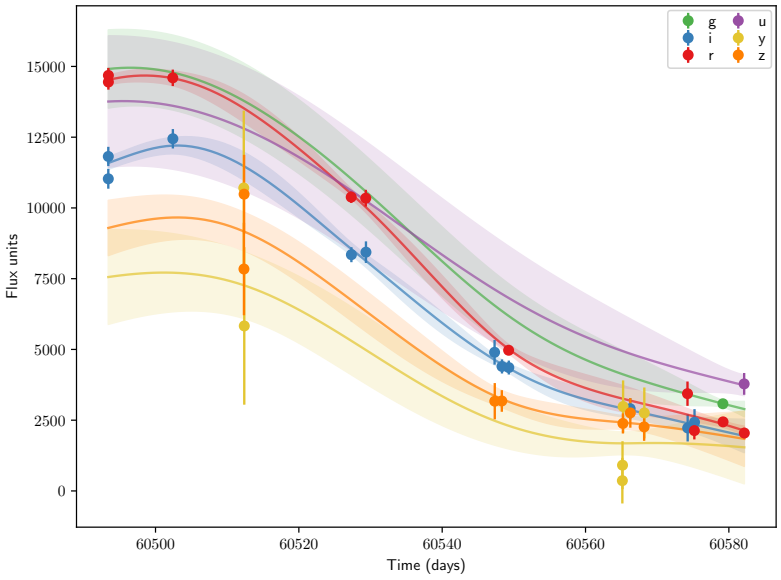


FIGURE B.9: TDE

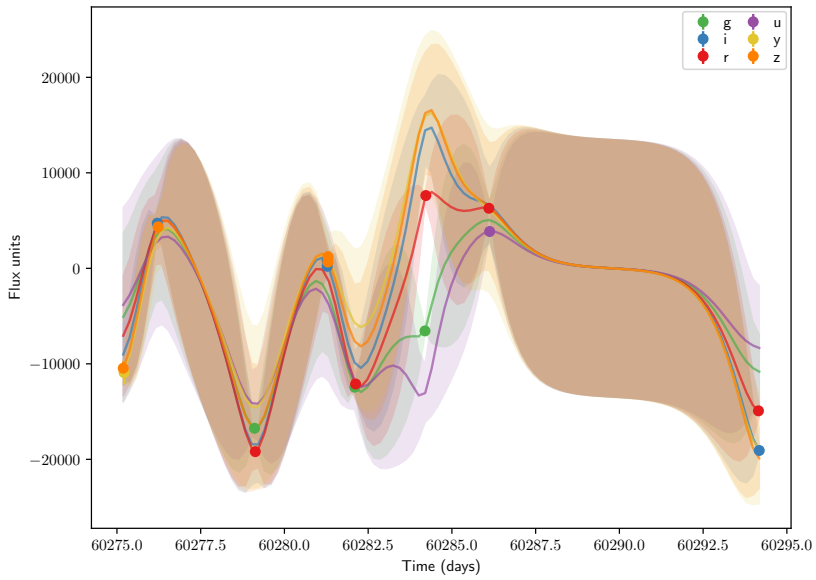


FIGURE B.10: RR Lyrae

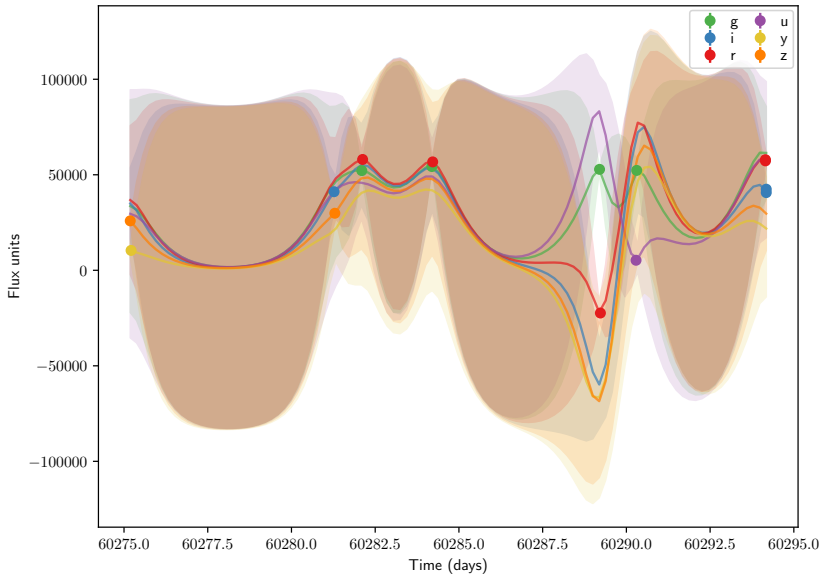


FIGURE B.11: EB

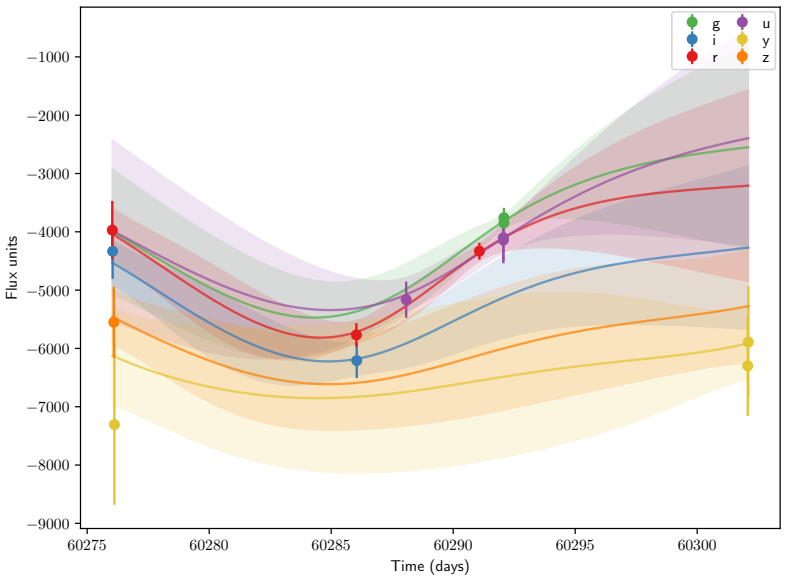


FIGURE B.12: AGN

Glossary

Activtion Function The activation function within a neuron is the function that defines the output of that neuron relative to the input. Typically these are non-linear functions that allow for the network to learn complex patterns in the data.

Epoch One full forward pass and one full backward pass of all the examples in the training set.

Hyperparameters The set of parameters that define the network structure or method of training. Examples include number of hidden layers, or the number or size of filters to use in a convolutional neural network.

Layer A layer in a neural network setting is a collection of neurons. A layer is defined as the set of neurons that recieve the same inputs. A neural network is considered a deep network if there are many layers which pass information through.

Neuron A neuron is mathematical approximation of a biological neuron in the brain. A single neuron is described by a set of inputs, a set of weights, and an activation function. The neuron translates these inputs into a single output, which can then be used as input for another layer of neurons.

Parameters Collection of weights and biases that define the output of a neural network.

Scale Factor, a A mathematical quantity that describes the changing separation of two points as the Universe expands.

Sketch A summary or approximation of data that is too large to query in a reasonable time-frame.

Bibliography

Abbott, Benjamin P et al. (2017). *Multi-messenger observations of a binary neutron star merger*. In: *Astrophysical Journal Letters* 848.2.

Abbott, Timothy et al. (2016). *The Dark Energy Survey: more than dark energy–an overview*. In: *Monthly Notices of the Royal Astronomical Society* 460.2, pp. 1270–1299.

Abbott, TMC et al. (2019). *First cosmology results using type Ia supernovae from the dark energy survey: constraints on cosmological parameters*. In: *The Astrophysical Journal Letters* 872.2, p. L30.

Abell, Paul A et al. (2009). *Lsst science book, version 2.0*. In: *arXiv preprint arXiv:0912.0201*.

Ade, Peter AR et al. (2016). *Planck 2015 results-xiii. cosmological parameters*. In: *Astronomy & Astrophysics* 594, A13.

Agarap, Abien Fred (2018). *Deep learning using rectified linear units (relu)*. In: *arXiv preprint arXiv:1803.08375*.

Aghanim, Nabila et al. (2020). *Planck 2018 results-VI. Cosmological parameters*. In: *Astronomy & Astrophysics* 641, A6.

Akiba, Takuya, Sano, Shotaro, Yanase, Toshihiko, Ohta, Takeru, and Koyama, Masanori (2019). *Optuna: A next-generation hyperparameter optimization framework*. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631.

Akrami, Yashar et al. (2020). *Planck 2018 results-VII. Isotropy and statistics of the CMB*. In: *Astronomy & Astrophysics* 641, A7.

Alam, Shadab et al. (2017). *The clustering of galaxies in the completed SDSS-III Baryon Oscillation Spectroscopic Survey: cos-*

mological analysis of the DR12 galaxy sample. In: *Monthly Notices of the Royal Astronomical Society* 470.3, pp. 2617–2652.

Alexander Friedmann (1922). *Über die krümmung des raumes*. In: *Z. Phys.* 10, pp. 377–386.

Allam Jr, T., Biswas, R., Hlǒzek, R., Lochner, M., J. D. McEwen, Peiris, H. V., and Setzer, C. (2019). *Optimising the LSST observing strategy for Supernova light curve classification with machine learning*. In: Biomedical and Astronomical Signal Processing Frontiers (BASP).

Allam, Jr., Tarek and McEwen, Jason D (2021). *Paying Attention to Astronomical Transients: Photometric Classification with the Time-Series Transformer*. In: *arXiv preprint arXiv:2105.06178*. arXiv: 2105.06178 [astro-ph.IM].

Alves, Catarina S, Peiris, Hiranya V, Lochner, Michelle, McEwen, Jason D, Allam, Tarek, Biswas, Rahul, Collaboration, LSST Dark Energy Science, et al. (2022a). *Considerations for Optimizing the Photometric Classification of Supernovae from the Rubin Observatory*. In: *The Astrophysical Journal Supplement Series* 258.2, p. 23.

Ambikasaran, Sivaram, Foreman-Mackey, Daniel, Greengard, Leslie, Hogg, David W, and O'Neil, Michael (2015). *Fast direct methods for Gaussian processes*. In: *IEEE transactions on pattern analysis and machine intelligence* 38.2, pp. 252–265.

Armbrust, Michael, Das, Tathagata, Torres, Joseph, Yavuz, Burak, Zhu, Shixiong, Xin, Reynold, Ghodsi, Ali, Stoica, Ion, and Zaharia, Matei (2018). *Structured streaming: A declarative api for real-time applications in apache spark*. In: *Proceedings of the 2018 International Conference on Management of Data*, pp. 601–613.

Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E (2016). *Layer normalization*. In: *arXiv preprint arXiv:1607.06450*.

Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. (2017). *The Great Time Series Classification Bake Off: a Review and Experimental Evaluation of Recent Algorithmic Advances*. In: *Data Mining and Knowledge Discovery* 31 (3), pp. 606–660.

Bagnall, Anthony, Dau, Hoang Anh, Lines, Jason, Flynn, Michael, Large, James, Bostrom, Aaron, Southam, Paul, and Keogh, Eamonn (2018). *The UEA multivariate time series classification archive, 2018*. In: *arXiv preprint arXiv:1811.00075*.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua (2014). *Neural machine translation by jointly learning to align and translate*. In: *arXiv preprint arXiv:1409.0473*.

Baumann, Daniel (2022). *Cosmology*. Cambridge University Press.

Baydogan, Mustafa Gokce (2015). *Multivariate Time Series Classification Datasets*. http://www.mustafabaydogan.com.

Bellm, Eric (2014). *The Zwicky transient facility*. In: *The Third Hot-wiring the Transient Universe Workshop*. Vol. 27.

Bellm, Eric, Blum, Robert, Graham, Melissa, Guy, Leanne, Ivezic, Zeljko, O'Mullane, William, Patterson, Maria, Swinbank, John, and LSST Project, Beth Willman for the (2019a). *LDM-612, Plans and Policies for LSST Alert Distribution*. https://ls.st/ldm-612.

Bellm, Eric, Blum, Robert, Graham, Melissa, Guy, Leanne, Ivezic, Zeljko, O'Mullane, William, Patterson, Maria, Swinbank, John, and LSST Project, for the (2020). *LDM-723, Call for Proposals for Community Alert Brokers*. https://ls.st/ldm-723.

Bellm, Eric, Blum, Robert, Graham, Melissa, Guy, Leanne, Ivezic, Zeljko, O'Mullane, William, and Swinbank, John (2019b). *LDM-682, Call for Letters of Intent for Community Alert Brokers*. https://ls.st/ldm-682.

Bergstra, James, Bardenet, Rémi, Bengio, Yoshua, and Kégl, Balázs (2011). *Algorithms for hyper-parameter optimization*. In: *25th annual conference on neural information processing systems (NIPS 2011)*. Vol. 24. Neural Information Processing Systems Foundation.

Betoule, M et al et al. (2014). *Improved cosmological constraints from a joint analysis of the SDSS-II and SNLS supernova samples*. In: *Astronomy & Astrophysics* 568, A22.

Bhuwalka, Pulkit et al. (2020). *Quantization Aware Training with TensorFlow Model Optimization Toolkit - Performance with Ac-*

curacy. URL: https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html.

Boone, Kyle (2019). *Avocado: Photometric classification of astronomical transients with gaussian process augmentation*. In: *The Astronomical Journal* 158.6, p. 257.

Branco, Paula, Torgo, Luis, and Ribeiro, Rita (2015). *A survey of predictive modelling under imbalanced distributions*. In: *arXiv preprint arXiv:1505.01658*.

Brausch, Lukas, Hewener, Holger, and Lukowicz, Paul (2022). *Classifying Muscle States with One-Dimensional Radio-Frequency Signals from Single Element Ultrasound Transducers*. In: *Sensors* 22.7, p. 2789.

Brownlee, Jason (2020). *Tour of Evaluation Metrics for Imbalanced Classification*. URL: https://bit.ly/3sxqx9Y (visited on 03/10/2021).

Brunel, Anthony, Pasquet, Johanna, PASQUET, Jérôome, Rodriguez, Nancy, Comby, Frédéric, Fouchez, Dominique, and Chaumont, Marc (2019). *A CNN adapted to time series for the classification of Supernovae*. In: *Electronic imaging* 2019.14, pp. 90–1.

Butkevich, AG, Berdyugin, AV, and Teerikorpi, P (2005). *Statistical biases in stellar astronomy: the Malmquist bias revisited*. In: *Monthly Notices of the Royal Astronomical Society* 362.1, pp. 321–330.

Chai, Yekun (2020). *Counting the Number of Parameters in Deep Learning*. URL: https://bit.ly/3tCcsJ2 (visited on 04/21/2021).

Chandrasekhar, Subrahmanyan (1931). *The highly collapsed configurations of a stellar mass*. In: *Monthly Notices of the Royal Astronomical Society* 91, pp. 456–466.

Charnock, Tom and Moss, Adam (2017). *Deep recurrent neural networks for supernovae classification*. In: *The Astrophysical Journal Letters* 837.2, p. L28.

Chen, Jiarong, Lu, Zongqing, Xue, Jing-Hao, and Liao, Qingmin (2020). *XSepConv: Extremely separated convolution*. In: *arXiv preprint arXiv:2002.12046*.

Cheng, Jianpeng, Dong, Li, and Lapata, Mirella (2016). *Long short-term memory-networks for machine reading*. In: *arXiv preprint arXiv:1601.06733*.

Chollet, François et al. (2015). *Keras*. https://keras.io.

Chollet, François (2017). *Xception: Deep learning with depthwise separable convolutions*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258.

Cohen-Tannoudji, Gilles (2018). *Lambda, the fifth foundational constant considered by Einstein*. In: *Metrologia* 55.4, p. 486.

David, Robert et al. (2021). *Tensorflow lite micro: Embedded machine learning for tinyml systems*. In: *Proceedings of Machine Learning and Systems* 3, pp. 800–811.

De Boor, Carl and De Boor, Carl (1978). *A practical guide to splines*. Vol. 27. springer-verlag New York.

De Cia, Annalisa et al. (2018). *Light curves of hydrogen-poor Superluminous Supernovae from the Palomar Transient Factory*. In: *The Astrophysical Journal* 860.2, p. 100.

De Putter, Roland (2010). *Probing dark energy with theory and observations*. University of California, Berkeley.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li (2009). *Imagenet: A large-scale hierarchical image database*. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Deutsch, Peter (1996). *Rfc1951: Deflate compressed data format specification version 1.3*.

Duev, Dmitry and Graham, Matthew (2022). *BABAMUL*. https://github.com/babamul/babamul.

Duvenaud, David (2014). *Automatic model construction with Gaussian processes*. PhD thesis. University of Cambridge.

Eisenstein, Daniel J et al. (2005). *Detection of the baryon acoustic peak in the large-scale correlation function of SDSS luminous red galaxies*. In: *The Astrophysical Journal* 633.2, p. 560.

Enderich, Lukas, Timm, Fabian, and Burgard, Wolfram (2021). *Holistic filter pruning for efficient deep neural networks*. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2596–2605.

Fawaz, Hassan et al. (2020). *Inceptiontime: Finding alexnet for time series classification*. In: *Data Mining and Knowledge Discovery* 34.6, pp. 1936–1962.

Fawaz, Hassan Ismail, Forestier, Germain, Weber, Jonathan, Idoumghar, Lhassane, and Muller, Pierre-Alain (2019). *Deep learning for time series classification: a review*. In: *Data Mining and Knowledge Discovery* 33.4, pp. 917–963.

Filippenko, Alexei V (1997). *Optical spectra of supernovae*. In: *Annual Review of Astronomy and Astrophysics* 35.1, pp. 309–355.

Flammarion, C. (1894). *Popular Astronomy: A General Description of the Heavens*. Chatto & Windus. URL: https://books.google.co.uk/books?id=fDgLAAAAYAAJ.

Förster, F et al. (2021). *The Automatic Learning for the Rapid Classification of Events (ALeRCE) Alert Broker*. In: *The Astronomical Journal* 161.5, p. 242.

Gabruseva, Tatiana, Zlobin, Sergey, and Wang, Peter (2020). *Photometric light curves classification with machine learning*. In: *Journal of Astronomical Instrumentation* 9.01, p. 2050005.

Gakhov, A. (2019). *Probabilistic Data Structures and Algorithms for Big Data Applications*. Books on Demand. ISBN: 9783748190486. URL: https://books.google.co.uk/books?id=PWOHDwAAQBAJ.

Gal-Yam, A. (Jan. 2021). *The TNS alert system*. In: *Bulletin of the AAS* 53.1. https://baas.aas.org/pub/2021n1i423p05.

Georges Lemaître (1927). *Un Univers homogène de masse constante et de rayon croissant rendant compte de la vitesse radiale des nébuleuses extra-galactiques*. In: *Annales de la Société scientifique de Bruxelles*. Vol. 47, pp. 49–59.

Gilmer, Matthew S, Kozyreva, Alexandra, Hirschi, Raphael, Fröhlich, Carla, and Yusof, Norhasliza (2017). *Pair-instability*

supernova simulations: progenitor evolution, explosion, and light curves. In: *The Astrophysical Journal* 846.2, p. 100.

Godines, Daniel, Bachelet, E, Narayan, Gautham, and Street, RA (2019). *A machine learning classifier for microlensing in wide-field surveys*. In: *Astronomy and Computing* 28, p. 100298.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Grosse, Roger and Ba, Jimmy (Winter 2019). *Neural Networks and Deep Learning*. URL: https://bit.ly/2R4wLjW.

Guy, Julien et al. (2007). *SALT2: using distant supernovae to improve the use of type Ia supernovae as distance indicators*. In: *Astronomy & Astrophysics* 466.1, pp. 11–21.

Han, Song, Mao, Huizi, and Dally, William J (2015a). *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. In: *International Conference on Learning Representations (ICLR)*.

Han, Song, Pool, Jeff, Tran, John, and Dally, William (2015b). *Learning both weights and connections for efficient neural network*. In: *Advances in neural information processing systems* 28.

Harris, Charles R. et al. (Sept. 2020). *Array programming with NumPy*. In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

He, Haibo and Ma, Yunqian (2013). *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian (2016). *Deep residual learning for image recognition*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

He, Yihui, Zhang, Xiangyu, and Sun, Jian (2017). *Channel pruning for accelerating very deep neural networks*. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397.

Hložek, R et al. (2020b). *Results of the Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC).* In: *arXiv preprint arXiv:2012.12392*. arXiv: 2012 . 12392 [astro-ph.IM].

Hochreiter, Sepp, Bengio, Yoshua, Frasconi, Paolo, Schmidhuber, Jürgen, et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.*

Hochreiter, Sepp and Schmidhuber, Jürgen (1997). *Long short-term memory.* In: *Neural computation* 9.8, pp. 1735–1780.

Hofmann, Heike, Kafadar, Karen, and Wickham, Hadley (2011). *Letter-value plots: Boxplots for large data.* In: *The American Statistican.*

Holz, Daniel E, Hughes, Scott A, and Schutz, Bernard F (2018). *Measuring cosmic distances with standard sirens.* In: *Physics today* 71.12, pp. 34–40.

Howard, Andrew G, Zhu, Menglong, Chen, Bo, Kalenichenko, Dmitry, Wang, Weijun, Weyand, Tobias, Andreetto, Marco, and Adam, Hartwig (2017). *Mobilenets: Efficient convolutional neural networks for mobile vision applications.* In: *arXiv preprint arXiv:1704.04861.*

Hubble, Edwin (1929). *A relation between distance and radial velocity among extra-galactic nebulae.* In: *Proceedings of the national academy of sciences* 15.3, pp. 168–173.

Huffman, David A (1952). *A method for the construction of minimum-redundancy codes.* In: *Proceedings of the IRE* 40.9, pp. 1098–1101.

Huterer, Dragan and Shafer, Daniel L (2017). *Dark energy two decades after: Observables, probes, consistency tests.* In: *Reports on Progress in Physics* 81.1, p. 016901.

Indyk, Piotr and Motwani, Rajeev (1998). *Approximate nearest neighbors: towards removing the curse of dimensionality.* In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613.

Ioffe, Sergey and Szegedy, Christian (2015). *Batch normalization: Accelerating deep network training by reducing internal covari-*

ate shift. In: *International conference on machine learning*. PMLR, pp. 448–456.

Ishida, Emille EO and Souza, Rafael S de (2013). *Kernel PCA for Type Ia supernovae photometric classification*. In: *Monthly Notices of the Royal Astronomical Society* 430.1, pp. 509–532.

Ivezić, Željko et al. (2019). *LSST: from science drivers to reference design and anticipated data products*. In: *The Astrophysical Journal* 873.2, p. 111.

Jha, Saurabh W (2017). *Type Iax Supernovae*. In: *arXiv preprint arXiv:1707.01110*.

Jones, DO et al. (2018). *Measuring Dark Energy Properties with Photometrically Classified Pan-STARRS Supernovae. II. Cosmological Parameters*. In: *The Astrophysical Journal* 857.1, p. 51.

Jordan, Michael I (1997). *Serial order: A parallel distributed processing approach*. In: *Advances in psychology*. Vol. 121. Elsevier, pp. 471–495.

Jurić, Mario, Axelrod, Tim, Becker, Andrew, Becla, Jacek, Bellm, Eric, Bosch, Jim, and Ciardi, David (2022). *Vera C. Rubin Observatory Systems Engineering Data Products Definition Document*. https://lse-163.lsst.io/.

Kaiser, Lukasz, Gomez, Aidan N, and Chollet, Francois (2017). *Depthwise separable convolutions for neural machine translation*. In: *arXiv preprint arXiv:1706.03059*.

Kaiser, Nicholas et al. (2002). *Pan-STARRS: a large synoptic survey telescope array*. In: *Survey and Other Telescope Technologies and Discoveries*. Vol. 4836. SPIE, pp. 154–164.

Karpenka, Natalia V, Feroz, F, and Hobson, MP (2013). *A simple and robust method for automated photometric classification of supernovae using neural networks*. In: *Monthly Notices of the Royal Astronomical Society* 429.2, pp. 1278–1285.

Kazemnejad, Amirhossein (2019). *Transformer Architecture: The Positional Encoding*. URL: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/ (visited on 03/25/2021).

Kessler, R et al. (2019). *Models and simulations for the photometric LSST astronomical time series classification challenge (PLAsTiCC)*. In: *Publications of the Astronomical Society of the Pacific* 131.1003, p. 094501.

Kessler, Richard, Conley, Alex, Jha, Saurabh, and Kuhlmann, Stephen (2010a). *Supernova photometric classification challenge*. In: *arXiv preprint arXiv:1001.5210*.

Kessler, Richard et al. (2009). *SNANA: A public software package for supernova analysis*. In: *Publications of the Astronomical Society of the Pacific* 121.883, p. 1028.

Kessler, Richard et al. (2010b). *Results from the supernova photometric classification challenge*. In: *Publications of the Astronomical Society of the Pacific* 122.898, p. 1415.

Kingma, Diederik P and Ba, Jimmy (2014). *Adam: A method for stochastic optimization*. In: *arXiv preprint arXiv:1412.6980*.

Koehrsen, Will (2018). *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*. URL: https://bit.ly/3cAyhTd (visited on 03/10/2021).

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E (2012). *Imagenet classification with deep convolutional neural networks*. In: *Advances in neural information processing systems* 25.

Kwapisz, Jennifer R, Weiss, Gary M, and Moore, Samuel A (2011). *Activity recognition using cell phone accelerometers*. In: *ACM SigKDD Explorations Newsletter* 12.2, pp. 74–82.

Lahav, Ofer (1996). *Artificial neural networks as a tool for galaxy classification*. In: *arXiv preprint astro-ph/9612096*.

Las Cumbres Observatory (2022). *SpaceBook: Cepheid Variable Stars, Supernovae and Distance Measurement*. https://lco.global/spacebook/distance/cepheid-variable-stars-supernovae-and-distance-measurement/.

Lazarevich, Ivan, Kozlov, Alexander, and Malinin, Nikita (2021). *Post-training deep neural network pruning via layer-wise calibration*. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 798–805.

Leavitt, Henrietta S and Pickering, Edward C (1912). *Periods of 25 Variable Stars in the Small Magellanic Cloud*. In: *Harvard College Observatory Circular* 173, pp. 1–3.

LeCun, Yann (July 6, 2021). *ConvNet Patent*. https://twitter.com/ylecun/status/1412544692219269121. eprint: Twitter.

LeCun, Yann, Boser, Bernhard, Denker, John S, Henderson, Donnie, Howard, Richard E, Hubbard, Wayne, and Jackel, Lawrence D (1989a). *Backpropagation applied to handwritten zip code recognition*. In: *Neural computation* 1.4, pp. 541–551.

LeCun, Yann, Denker, John, and Solla, Sara (1989b). *Optimal brain damage*. In: *Advances in neural information processing systems* 2.

LeCun, Yann, Jackel, Lionel D, Boser, Brian, Denker, John S, Graf, Henry P, Guyon, Isabelle, Henderson, Don, Howard, Richard E, and Hubbard, William (1989c). *Handwritten digit recognition: Applications of neural network chips and automatic learning*. In: *IEEE Communications Magazine* 27.11, pp. 41–46.

Li, Shuangfeng (2020). *Tensorflow lite: On-device machine learning framework*. In: *J. Comput. Res. Dev* 57, p. 1839.

Lin, Min, Chen, Qiang, and Yan, Shuicheng (2013). *Network in network*. In: *arXiv preprint arXiv:1312.4400*.

Liu, Shuying and Deng, Weihong (2015). *Very deep convolutional neural network based image classification using small training sample size*. In: *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*. IEEE, pp. 730–734.

Lochner, M. et al. (2018b). *Optimizing the LSST Observing Strategy for Dark Energy Science: DESC Recommendations for the Wide-Fast-Deep Survey*. In: *arXiv*. eprint: arXiv:1812.00515.

Lochner, Michelle, McEwen, Jason D, Peiris, Hiranya V, Lahav, Ofer, and Winter, Max K (2016). *Photometric supernova classification with machine learning*. In: *The Astrophysical Journal Supplement Series* 225.2, p. 31.

Lochner, Michelle et al. (July 2021). *snmachine: Photometric supernova classification*. ascl: 2107.006.

Luminet, Jean-Pierre (2015). *Lemaitre's Big Bang*. In: *arXiv preprint arXiv:1503.08304*.

Lundberg, Scott M and Lee, Su-In (2017). *A unified approach to interpreting model predictions*. In: *Advances in neural information processing systems* 30.

Luong, Minh-Thang, Pham, Hieu, and Manning, Christopher D (2015). *Effective approaches to attention-based neural machine translation*. In: *arXiv preprint arXiv:1508.04025*.

Madsen, Andreas (2019). *Visualizing memorization in RNNs*. In: *Distill*. https://distill.pub/2019/memorization-in-rnns. DOI: 10.23915/distill.00016.

Mallat, S. (2008). *A Wavelet Tour of Signal Processing: The Sparse Way*. Elsevier Science. ISBN: 9780080922027. URL: https://books.google.co.uk/books?id=5qzeLJljuLoC.

Malz, AI et al. (2019b). *The Photometric LSST Astronomical Time-series Classification Challenge PLAsTiCC: Selection of a Performance Metric for Classification Probabilities Balancing Diverse Science Goals*. In: *The Astronomical Journal* 158.5, p. 171.

Marshall, Phil et al. (2017). *Science-driven optimization of the LSST observing strategy*. In: *arXiv preprint arXiv:1708.04058*.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Matheson, Thomas et al. (2021). *The ANTARES astronomical time-domain event broker*. In: *The Astronomical Journal* 161.3, p. 107.

McKinney, Wes (2010). *Data Structures for Statistical Computing in Python*. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey (2013). *Efficient estimation of word representations in vector space*. In: *arXiv preprint arXiv:1301.3781*.

Möller, Anais and Boissière, Thibault de (2020). *SuperNNova: an open-source framework for Bayesian, neural network-based supernova classification*. In: *Monthly Notices of the Royal Astronomical Society* 491.3, pp. 4277–4293.

Möller, Anais et al. (2021). *Fink, a new generation of broker for the LSST community*. In: *Monthly Notices of the Royal Astronomical Society* 501.3, pp. 3272–3288.

Muthukrishna, Daniel, Narayan, Gautham, Mandel, Kaisey S, Biswas, Rahul, and Hložek, Renée (2019). *RAPID: early classification of explosive transients using deep learning*. In: *Publications of the Astronomical Society of the Pacific* 131.1005, p. 118002.

Nair, Vinod and Hinton, Geoffrey E (2010). *Rectified linear units improve restricted boltzmann machines*. In: *Icml*.

Narayan, Gautham et al. (2018). *Machine-learning-based Brokers for Real-time Classification of the LSST Alert Stream*. In: *The Astrophysical Journal Supplement Series* 236.1, p. 9.

Naryan, Gautham et al. (2022). *The DESC ELAsTiCC Challenge*. https://portal.nersc.gov/cfs/lsst/DESC_TD_PUBLIC/ELASTICC/.

Nguyen, Thao NN, Veeravalli, Bharadwaj, and Fong, Xuanyao (2021). *Connection Pruning for Deep Spiking Neural Networks with On-Chip Learning*. In: *International Conference on Neuromorphic Systems 2021*, pp. 1–8.

Nordin, J et al. (2019). *Transient processing and analysis using AMPEL: alert management, photometry, and evaluation of light curves*. In: *Astronomy & Astrophysics* 631, A147.

Oord, Aaron van den, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew, and Kavukcuoglu, Koray (2016). *Wavenet: A generative model for raw audio*. In: *arXiv preprint arXiv:1609.03499*.

Patterson, Maria, Bellm, Eric, and Swinbank, John (2019). *h-093: Design of the LSST Alert Distribution System*. https://dmtn-093.lsst.io/.

Patterson, Maria T et al. (2018). *The zwicky transient facility alert distribution system*. In: *Publications of the Astronomical Society of the Pacific* 131.995, p. 018001.

Peloton, Julien, Ishida, Emille, and Möller, Anais (2020). *First FINK Workshop*. https://indico.in2p3.fr/event/20222/.

Perlmutter, Saul et al. (1999). *Measurements of Ω and Λ from 42 high-redshift supernovae*. In: *The Astrophysical Journal* 517.2, p. 565.

Petsiuk, Vitali, Das, Abir, and Saenko, Kate (2018). *Rise: Randomized input sampling for explanation of black-box models*. In: *arXiv preprint arXiv:1806.07421*.

Phillips, Mark M (1993). *The absolute magnitudes of Type IA supernovae*. In: *The Astrophysical Journal* 413, pp. L105–L108.

Rasmussen, Carl Edward (2004). *Gaussian processes in machine learning*. In: *Advanced lectures on machine learning*. Springer, pp. 63–71.

Revsbech, Esben A, Trotta, Roberto, and Dyk, David A van (2018). *STACCATO: a novel solution to supernova photometric classification with biased training sets*. In: *Monthly Notices of the Royal Astronomical Society* 473.3, pp. 3969–3986.

Riess, Adam G et al. (1998). *Observational evidence from supernovae for an accelerating universe and a cosmological constant*. In: *The Astronomical Journal* 116.3, p. 1009.

Riess, Adam G et al. (2001). *The farthest known supernova: support for an accelerating universe and a glimpse of the epoch of deceleration*. In: *The Astrophysical Journal* 560.1, p. 49.

Riess, Adam G et al. (2004). *Type Ia supernova discoveries at z¿ 1 from the Hubble Space Telescope: Evidence for past deceleration and constraints on dark energy evolution*. In: *The Astrophysical Journal* 607.2, p. 665.

Robitaille, Thomas P et al. (2013). *Astropy: A community Python package for astronomy*. In: *Astronomy & Astrophysics* 558, A33.

Ruiz, Alejandro Pasos, Flynn, Michael, Large, James, Middlehurst, Matthew, and Bagnall, Anthony (2021). *The great multivariate time series classification bake off: a review and experimental*

evaluation of recent algorithmic advances. In: *Data Mining and Knowledge Discovery* 35.2, pp. 401–449.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J (1986). *Learning representations by back-propagating errors*. In: *nature* 323.6088, pp. 533–536.

Ryden, Barbara (2017). *Introduction to cosmology*. Cambridge University Press.

Scolnic, D. M. et al. (2018b). *Optimizing the LSST Observing Strategy for Dark Energy Science: DESC Recommendations for the Deep Drilling Fields and other Special Programs*. In: *arXiv*. eprint: arXiv:1812.00516.

Selvaraju, Ramprasaath R, Cogswell, Michael, Das, Abhishek, Vedantam, Ramakrishna, Parikh, Devi, and Batra, Dhruv (2017). *Grad-cam: Visual explanations from deep networks via gradient-based localization*. In: *Proceedings of the IEEE international conference on computer vision*, pp. 618–626.

Shokoohi-Yekta, Mohammad, Hu, Bing, Jin, Hongxia, Wang, Jun, and Keogh, Eamonn (2017). *Generalizing DTW to the multi-dimensional case requires an adaptive approach*. In: *Data mining and knowledge discovery* 31.1, pp. 1–31.

Sifre, Laurent and Mallat, Prof Stéphane (2014). *Rigid-motion scattering for image classification author*. In: *Ecole Polytechnique, CMAP PhD thesis*.

Singh, Mridweeka et al. (2021). *The Fast-evolving Type Ib Supernova SN 2015dj in NGC 7371*. In: *The Astrophysical Journal* 909.2, p. 100.

Smith, Ken (2019). *Lasair: the transient alert broker for LSST: UK*. In: *The Extragalactic Explosive Universe: the New Era of Transient Surveys and Data-Driven Discovery*, p. 51.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan (2014). *Dropout: a simple way to prevent neural networks from overfitting*. In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V (2014). *Sequence to sequence learning with neural networks*. In: *arXiv preprint arXiv:1409.3215*.

Szegedy, Christian, Ioffe, Sergey, Vanhoucke, Vincent, and Alemi, Alexander A (2017). *Inception-v4, inception-resnet and the impact of residual connections on learning*. In: *Thirty-first AAAI conference on artificial intelligence*.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew (2015a). *Going deeper with convolutions*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jonathon, and Wojna, Zbigniew (2015b). *Rethinking the inception architecture for computer vision. CoRR abs/1512.00567 (2015)*.

Tay, Yi, Dehghani, Mostafa, Bahri, Dara, and Metzler, Donald (2020). *Efficient transformers: A survey*. In: *arXiv preprint arXiv:2009.06732*.

The PLAsTiCC team et al. (2018b). *The photometric LSST astronomical time-series classification challenge (PLAsTiCC): Data set*. In: *arXiv preprint arXiv:1810.00001*.

Tong, David (2019). *Cosmology: University of Cambridge Part II Mathematical Tripos*. http://www.damtp.cam.ac.uk/user/tong/cosmo/cosmo.pdf.

Valcin, David, Bernal, José Luis, Jimenez, Raul, Verde, Licia, and Wandelt, Benjamin D (2020). *Inferring the age of the universe with globular clusters*. In: *Journal of Cosmology and Astroparticle Physics* 2020.12, p. 002.

Varughese, Melvin M, Sachs, Rainer von, Stephanou, Michael, and Bassett, Bruce A (2015). *Non-parametric transient classification using adaptive wavelets*. In: *Monthly Notices of the Royal Astronomical Society* 453.3, pp. 2848–2861.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Lukasz, and Polo-

sukhin, Illia (2017). *Attention is all you need*. In: *arXiv preprint arXiv:1706.03762*.

Vesto Slipher (1917). *Radial velocity observations of spiral nebulae*. In: *The Observatory* 40, pp. 304–306.

Vincenzi, Maria et al. (2021). *The Dark Energy Survey supernova programme: modelling selection efficiency and observed core-collapse supernova contamination*. In: *Monthly Notices of the Royal Astronomical Society* 505.2, pp. 2819–2839.

Wang, Dun, Hogg, David W, Foreman-Mackey, Daniel, and Schölkopf, Bernhard (2017). *A pixel-level model for event discovery in time-domain imaging*. In: *arXiv preprint arXiv:1710.02428*.

Weiss, Gary M, Yoneda, Kenichi, and Hayajneh, Thaier (2019). *Smartphone and smartwatch-based biometrics using activities of daily living*. In: *IEEE Access* 7, pp. 133190–133202.

Weng, Lilian (2018). *Attention? Attention!* In: *lilianweng.github.io/lil-log*. URL: http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html.

Wenger, Marc et al. (2000). *The SIMBAD astronomical database-The CDS reference database for astronomical objects*. In: *Astronomy and Astrophysics Supplement Series* 143.1, pp. 9–22.

Wood-Vasey, Michael, Daher, Christine Mazzola, Perrefort, Daniel, and Raen, Troy (2022). *Pitt-Google Broker*. https://github.com/mwvgroup/Pitt-Google-Broker.

Ye, Lexiang and Keogh, Eamonn (2011). *Time series shapelets: a novel technique that allows accurate, interpretable and fast classification*. In: *Data mining and knowledge discovery* 22.1, pp. 149–182.

Yoon, Clara E, O'Reilly, Ossian, Bergen, Karianne J, and Beroza, Gregory C (2015). *Earthquake detection through computationally efficient similarity search*. In: *Science advances* 1.11, e1501057.

York, Donald G et al. (2000). *The sloan digital sky survey: Technical summary*. In: *The Astronomical Journal* 120.3, p. 1579.

Zaharia, Matei et al. (2016). *Apache spark: a unified engine for big data processing*. In: *Communications of the ACM* 59.11, pp. 56–65.

Zeiler, Matthew D and Fergus, Rob (2014). *Visualizing and understanding convolutional networks*. In: *European conference on computer vision*. Springer, pp. 818–833.

Zhou, B., Khosla, A., A., Lapedriza., Oliva, A., and Torralba, A. (2015). *Learning Deep Features for Discriminative Localization*. In: *CVPR*.

Ziv, Jacob and Lempel, Abraham (1977). *A universal algorithm for sequential data compression*. In: *IEEE Transactions on information theory* 23.3, pp. 337–343.

Zyla, P.A. et al. (2020). *Review of Particle Physics*. In: *PTEP* 2020.8. and 2021 update, p. 083C01. DOI: 10.1093/ptep/ptaa104.

Index