MACHINE LEARNING AS A TOOL FOR PREDICTING COSMOLOGICAL STRUCTURE FORMATION ON LARGE AND SMALL SCALES

By

Abigail Petulante

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Astrophysics

December 17, 2022

Nashville, Tennessee

ACKNOWLEDGMENTS

I'd like to begin by thanking my amazing advisors, Andreas Berlind and Kelly Holley-Bockelmann. It has been truly awe-inspiring to see how deeply and quickly you both are able to think through any problem, and having you two as role models has made me a much better scientist and critical thinker. But much more importantly than that, your endless belief in me finally at some point turned into some belief in myself, and I can't thank you enough for that. I would also like to say thank you to all of my committee members, for your help, support, and feedback, which has been fundamental in getting me here. Manodeep Sinha, thank you for being an excellent collaborator and mentor.

To all members of the Large Scale Structure group, past and present, thank you for the comments, questions, and discussions during our group meetings which provided invaluable insights and new perspectives on all of my research problems. To all of my office mates, Victor and Laura especially, thank you for providing the perfect combination of laughter and in-the-zone work sessions to help me to be optimally productive but not too stressed. To all of the other graduate students and friends at Vanderbilt, thank you for the camaraderie you have provided over the years. This department is truly full of wonderful and kind people. Gillian and Mehnaaz especially, thank you for always being friends and colleagues through the unspeakable bond that is being new PhD students together.

To all of my family and friends, thank you for the constant support, comfort, advice, and for always reminding me that the stuff I work on is actually really cool by turning to me for your fun and wacky astronomy questions. More than anything, Steve, thank you for truly being my better half, my biggest supporter, and my best friend through each and every step of this journey. Anything and everything feels possible with you by my side.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The universe as we know it is made up of many mysterious constituents. One of these such constituents, dark matter, is fundamental to our understanding of how all structures in the universe come to be. In the standard cosmological model, dark matter makes up 27% of the total composition of the universe; however, it comprises around 84% of the total *matter* of the universe, with the other 16% being normal baryonic matter. While dark matter is unable to form the structures in the universe with which we are most familiar - planets, stars, galaxies, all of which are made up of baryonic matter - its gravitational effect on baryonic matter means that it is the hidden underyling driver of the formation of all observable structures in the universe.

In the early universe, shortly after the Big Bang, the universe lacked any significant structure. It was nearly uniform in its distribution of matter and energy. However, over the course of the next $\sim$14 billion years, small fluctuations in the initial density distribution would grow and become more extreme via gravity. Today, we can observe the imprint of these primordial density perturbations as the ”cosmic web” - large filamentary structures harbor massive clusters of galaxies in regions where the universe was once just slightly denser, and cosmic voids are now barren where they were once just slightly underdense. Because galaxies live in the denser regions of the universe, the effects of this density evolution are observable when we measure the distribution of galaxies that we seen in our own universe today. Measurements of this distribution, combined with models of cosmological structure formation through simulations, provide an extremely useful way for astrophysicists to compare observed and simulated structures to understand exactly what conditions the universe needs in order to form into the one in which we live (Bertschinger, 1998; Angulo and Hahn, 2022).

This makes cosmological simulations incredibly useful tools in modern astronomy - much of our understanding of how our universe came to be has been reliant on using simulations to model how universes with different cosmological parameters would develop. Despite their usefulness, it has been difficult for astrophysicists to have access to all of the simulations that they would ideally need. For one, it is often most useful to have many realizations of a simulation to accurately estimate and understand errors in calculated statistics (Manera et al., 2013; Avila et al., 2018). For some areas of study, simulations must be of very high resolution to accurately model the growth of structure at smaller scales (van den Bosch et al., 2018; Joyce et al., 2021). For other applications, simulations must be of large volumes to generate structures on all physical scales that we can measure (Greig et al., 2022). However, the computational expense of running a simulation is often prohibitively high, requiring millions of CPU hours. This makes running many, large, high-resolution simulations all but impossible, and has opened a large area of research in astrophysics to create estimations of behaviors that would otherwise be modeled via full simulations.

In recent years, the burgeoning field of machine learning has been of interest to scientists of all disciplines. In astronomy, machine learning has been used for a diverse set of topics, from detecting exoplanets (McCauliff et al., 2015; Shallue and Vanderburg, 2018; Schanche et al., 2019a; Cuéllar et al., 2022), to classifying galaxies (De La Calleja and Fuentes, 2004; Banerji et al., 2010; Khalifa et al., 2017), to identifying properties of clusters (Angora et al., 2020; Yan et al., 2020), to generating maps of the cosmic microwave background (Caldeira et al., 2019; Mishra et al., 2019; Guzman and Meyers, 2022). The ability of machine learning methods to model a mapping from some input to some output is ripe for the field of structure formation, which often relies on highly complex models (simulations) to predict structure growth. The goal of this dissertation is to investigate how modern machine learning methods can be used to model large-scale structure growth in astrophysics without the use of traditional simulations.

1.1 The Hierarchical Growth of Structure in an ΛCDM Universe

The most widely accepted cosmological model, ΛCDM (cold dark matter) states that the universe is made up of 68% dark energy, 27% dark matter, and 5% baryonic matter (Planck Collaboration et al., 2020). Dark energy acts as a negative vaccuum pressure on the universe - forcing its accelerated expansion. Baryonic matter makes up all visible components of our universe (e.g. planets, stars, galaxies, gas). Dark matter is some type of matter which is unable to interact through (at least) two of the four fundamental forces in the universe - neither electromagnetic, nor strong nuclear forces, making it impossible to directly observe or to clump into very tightly bound structures. However, dark matter is still able to interact through the gravitational force, and can attract mass from other dark matter and baryonic matter alike. The effect that this has on structures in the universe is profound. Although dark matter is not directly observable, its effects can be seen all over the universe, in galaxies through the measuring of rotation curves (Rubin et al., 1980), in galaxy clusters through dynamical measurements (Zwicky, 1933) and gravitational lensing (Soucail et al., 1987; Paczynski, 1987), and at the largest scales in the cosmic microwave background radiation (Spergel et al., 2003; Planck Collaboration et al., 2020), which has imprinted in it the density fluctuations that come to make up the ”cosmic web” of our universe.

A fundamental consequence of this cosmological model is the hierarchical growth of structure. Pockets of dark matter first coalesce into clouds which collapse into virialized structures, called ”halos”, once a region of dark matter is dense enough to become stable, at $\sim$200 times the mean density of the universe (Press and Schechter, 1974; Navarro et al., 1996). These halos attract each other via gravity, merging and forming larger halos. A larger halo is then able to attract more halos to merge with, so structures become progressively larger via the constant accretion of those smaller than them. This merging process is, importantly, not instantaneous. Small halos that fall into larger halos remain inside those larger halos as substructures - called ”subhalos” - for quite some time, slowly losing their mass via tidal stripping and dynamical friction (Tormen et al., 1998; Weinberg, 1989) to the host as they orbit.

1.2 Understanding Structure Formation with Cosmological Simulations

The hierarchical formation of the cosmic web is commonly studied with the assistance of cosmological N-body simulations. These simulations are also commonly "dark-matter only" simulations, meaning that normal baryonic matter and the formation of stars and galaxies through the physics of gas is not modeled, only the effects of gravity are modeled. Despite lacking any baryonic physics, on the largest scales gravity is responsible for the entirety of structure formation, so these models still accurately reproduce the cosmic web and detail the dynamics of dark matter halos through time. This allows us to probe how different cosmological parameters, initial conditions, and assumptions about the physics of our universe, affect the growth of structures through the statistical properties of dark matter halos (Springel, 2005).

These simulations model structure formation by placing collisionless particles inside a simulation box and moving them in accordance with the effects of gravity. In the early universe, the distributions of these particles, which represent large chunks of mass, are distributed fairly uniformly, as soon after the Big Bang the universe was still nearly homogeneous everywhere. However, as the simulation runs forward, and each particle is moved based on how much gravitational pull it feels from every other particle, these particles end up clumping together and forming structures. A simulation is run forward in timesteps, with at each timestep, each particle being assigned a new position and velocity which are calculated from the force it has experienced in the passing time interval. This is repeated until the age of the universe in the simulation is equal to the age of the universe today.

Figure 1.1 shows an example of the evolution of the density field in a dark matter only simulation. The top left panel shows the distribution of particles at the beginning of the simulation, 0.1 Gyr after the Big Bang. The top right panel shows the distribution of particles at 1.5 Gyr, the bottom left panel shows the distribution of particles at 5 Gyr, and the

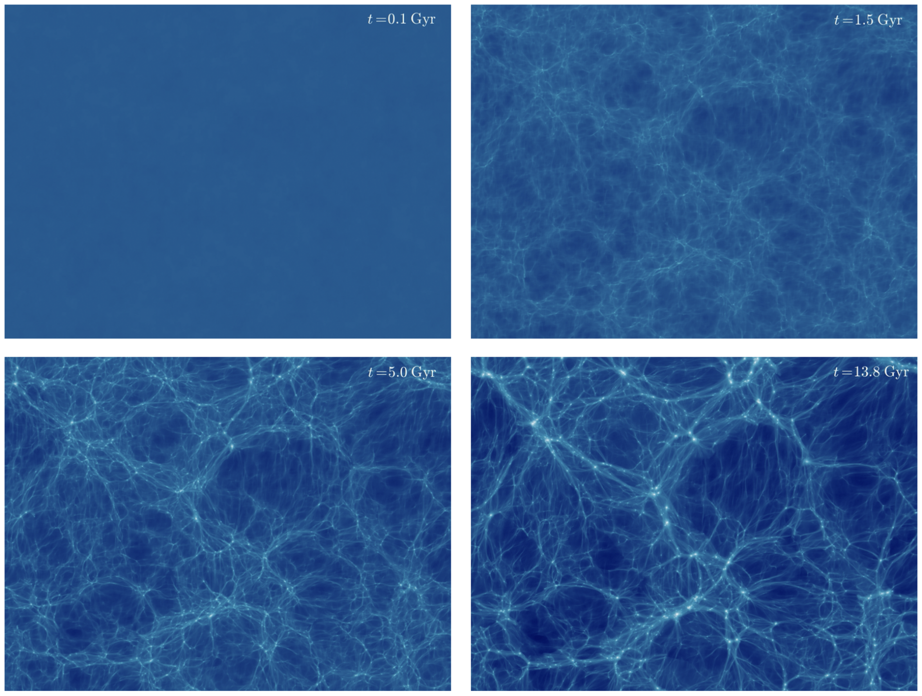


Figure 1.1: An example of a 179 x 134 Mpc x 18 Mpc (depth) simulation box from a dark-matter only simulation. The top left panel shows the distribution of particles at the beginning of a simulation, 0.1 Gyr after the Big Bang. The top right panel shows the distribution of particles at 1.5 Gyr, the bottom left panel shows the distribution of particles at 5 Gyr, and the bottom right panel shows the final distribution of particles at the end of the simulation. Brighter colors correspond to higher density. As the simulation evolves, overdense regions become progressively more dense (brighter) and underdense regions become progressively less dense (darker). Image credit: Screenshots come from a video created by Benedikt Diemer and Patt Mansfield. The individual frames were created using Phil Mansfield's visualization code gotetra (Mansfield, 2020). The simulation is described in detail in Diemer and Kravtsov (2014).

bottom right panel shows the final distribution of particles at the end of the simulation. At this simulation end point, the particles have clumped into structures which we can confirm statistically match our real universe. This is done with the assistance of galaxy surveys, which allow astronomers to observationally sample the distribution of mass in our universe (York et al., 2000).

1.2.1 Simulations from Beginning to End

Highly accurate N-body simulations begin before the first gravitational calculations are ever made. Instead, they must begin with a set of initial conditions that accurately represent the density fluctuations of the early universe. The first step of this is to choose a cosmological model, which specifies an early-universe power spectrum that matches the cosmic microwave background. Then, in a universe that experiences a period of rapid expansion known as inflation, the initial field of density fluctuations is fully specified by sampling a Gaussian random field from this power spectrum (Guth and Pi, 1982; Brandenberger, 1985).

Collisonless dark matter particles are then laid down on a lattice, and perturbed in position and velocity by a displacement field to the starting redshift of the simulation (typically $z \sim 100$). This displacement field can be generated to first order with the Zel'dovich approximation (Zel'dovich, 1970), or more accurately to second order using second-order Lagrangian perturbation theory (2LPT) (Scoccimarro, 1998), with the perturbations being dependent on the initial field of density fluctuations. These approximations do not account for the highly nonlinear evolution that occurs at small physical scales, but are accurate enough to generate initial conditions to a very high redshift, when growth of structure in the universe was still nearly linear.

The simulation begins at this early redshift, and further particle displacements between timesteps are calculated according to the laws of Newtonian gravity. As this becomes a numerical integration problem, these timesteps must be chosen to be sufficiently small ($\sim 10^3$

timesteps per simulation). Additionally, the simulation box must have periodic boundaries to represent an indefinite universe. Finally, it's advantageous to have many particles (billions) of sufficiently small mass ($\sim 10^9$ $M_\odot$) to adequately and robustly resolve galaxy-sized structures.

1.2.2 Challenges for Large Simulations

The primary challenge for modern simulations is accurately calculating the forces for all N particles without the computational cost becoming so burdensome that the task is rendered impossible. In principle, each timestep requires N^2 calculations to iterate, for each particle, over every other particle in the simulation. However, some clever methods have been developed to significantly decrease this computational cost by taking advantage of the inverse-square nature of the gravitational force, combined with the large physical distances that are covered by a simulation box, allowing forces from particles at large distances to be approximated without significant loss of accuracy. This can be done, for example, with particle-mesh methods which approximate the force as a field on a grid (Centrella and Melott, 1983), or with tree methods that explicitly hierarchically bunch particles to estimate their force contributions in groups (Barnes and Hut, 1986). These methods reduce the number of calculations to $N_{grid}\log N_{grid}$ and $N\log N$, respectively. It is hugely advantageous to reduce the number of calculations required, as it can allow the creation of simulations that are more accurate and more useful.

These techniques allow the generation of high resolution simulations to be feasible but not free - running a simulation still requires millions of CPU hours and remains unachievable without the use of a supercomputer. Faster approximations of density evolution can be accomplished, but with significantly less accuracy. It remains an open challenge to both quickly and accurately model small scale structures.

1.2.2.1 Fast and Simple Approximations of Structure Growth

Some analytic estimations of structure growth have made it possible to quickly and crudely approximate an evolved density field. Perturbation theory estimations straightforwardly move particles towards overdensities. The Zel'Dovich approximation will move particles in a straight line towards an overdensity. Second-order Lagrangian perturbation theory (2LPT) keeps higher order terms, so will move particles in a mostly straight line, with some curvature. As we have discussed, both of these are suitable for generating initial conditions for simulations, as in the early universe the growth of structures was still quite linear. These approximations *can* be used to evolve the density field all the way to z=0, although will be highly inaccurate for small scale structures which are very nonlinear in their evolution. Quasi-N-body approximators have been developed which add additional models of evolution on top of perturbation theory estimations (Scoccimarro and Sheth, 2002; Monaco et al., 2013; Tassev et al., 2013), matching the fully non-linear power spectrum significantly better than perturbation theory alone, i.e with FastPM down $k < 0.5h/\mathrm{Mpc}$ (Feng et al., 2016). These fast approximations are useful for quickly generating simulations for which the larger-scale structures are what is of primary interest.

1.3 Hierarchical Growth in Simulations via Mergers

While the first dark matter halos are formed via collections of particles collapsing into virialized structures, halos grow to significant size by collecting mass via series of mergers. In this framework, when smaller halos are gravitationally attracted to larger halos, they fall into the larger halo, gradually losing their mass to the larger halo through the effects of dynamical friction and tidal stripping. Tidal stripping is the process whereby particles belonging to a diffuse object are differentially affected by a gravitational potential, shearing the object and shedding its outermost particles (Kampakoglou and Benson, 2007). Dynamical friction is the process whereby an orbiting object gradually moves more slowly through an object due to the cumulative drag force of particles from the host object that were left in its gravitational wake (Tormen et al., 1998; Weinberg, 1989). As a subhalo orbits within a host halo, these effects mean its orbit decays and it falls deeper into denser regions of the host halo. In response, it is susceptible to more mass loss, which will occur until the subhalo has lost so much mass that it is no longer identifiable as a unique object.

1.3.1 Halos and Subhalos from Simulations

Our current understanding of the formation of galaxies is that they reside in the interior portion of dark matter halos and subhalos (Kahn and Woltjer, 1959; Blaauw and Schmidt, 1965; Ostriker and Peebles, 1973). Thus, tracking the growth and merging of halos and subhalos within these cosmological simulations not only elucidates how dark matter structures form in detail, but allows a better understanding of how galaxies group and merge. Regions in the dark matter density field that have collapsed into sufficiently dense structures are identified using halo finders - post-processing codes that are run on a snapshot of a completed simulation to identify which particles have grouped into these stable, spherical-like structures (Springel et al., 2001; Behroozi et al., 2013a). The boundaries of these halos are not smooth or clearly defined in reality, so halo finders must make choices about how to identify halos and decide which particles belong to them. One such way to do this is with a friends-of-friends algorithm, where some linking length is decided, and a halo is formed by collecting particles that are within that length from each other until no more particles can be linked together by less than that distance (Davis et al., 1985). Another option is to grow spherical overdensities, which choose a central overdensity as a a starting point, then grow a radius outward from that center point until the enclosed density drops off to $200\rho_{\rm crit}$ (Press and Schechter, 1974). Once this density threshold has been reached, the halo is assumed spherical and particles that lay within the radius are assigned to that halo (Klypin and Holtzman, 1997; Gill et al., 2004a).

Figure 1.2 shows an example of a roughly Milky Way sized host halo and its substructure in the VISHNU simulation, found with the ROCKSTAR halo finder (Behroozi et al.,

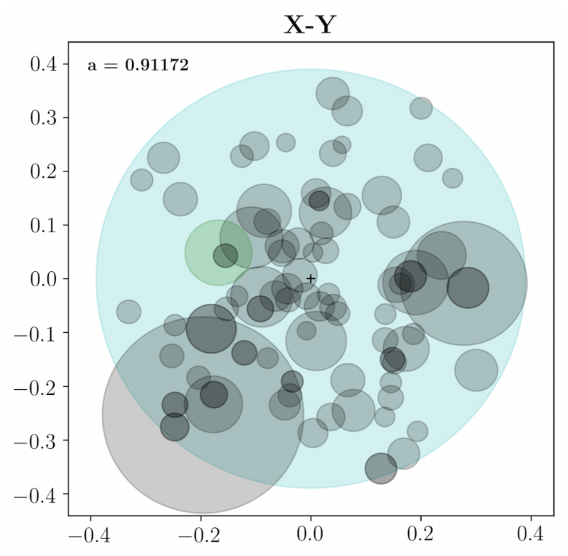


Figure 1.2: An example of a host halo of mass $\sim 10^{12}$ and its substructure, shown in the X-Y plane, which was identified using the ROCKSTAR phase-space halo finder on the VISHNU simulation. The blue circle shows the radius of the host halo, the black circles show the radii of all identified substructures belonging to that host halo at that snapshot, and the green circle shows a target subhalo which has been tracked through time.

2013a), at two different snapshots. The left panel shows an earlier snapshot (a $\sim$ 0.9; z $\sim$ 0.1; t $\sim$ 12.5Gyr) and the right panel shows the end of the simulation (a = 1; z = 0; t $\sim$ 13.8Gyr). The subhalo and host halo are drawn as circles with radius R_{200b}, with substructures being in black and the host halo being in blue. One such subhalo is highlighted in green to show its movement through the host as the simulation progresses, which can be tracked with the help of merger trees.

1.3.2 Merger Trees

As we have already discussed, structures in the universe form hierarchically through series of mergers. Thus, it is imperative to track the growth and merger history of a halo throughout the entirety of a simulation for its detailed evolution to be studied. After identifying halos for every snapshot of a simulation via a halo finder, a merger tree code can be applied to match halos and subhalos across different snapshots, identifying a structure as being the same in one snapshot to the next, even if its mass changes or it moves (Behroozi et al., 2013b). A resulting merger tree gives, for any halo that exists at the end of the simulation, a detailed account of how it collected mass over the last $\sim$14 billion years, and, for any subhalo which had merged, a detailed account of how it lost its mass during the merging process.

1.3.3 Semi-Analytic Models, Halo Models, and the Study of Galaxies without Modeling Baryons

Galaxies inhabit dark matter halos, so models of halo merging and evolution naturally provide insights to galaxy merging and evolution, far extending the usefulness of dark-matter only simulations and allowing for the study of galaxies without explicitly modeling baryons through hydrodynamic simulations. Semi-analytic models, which typically apply analytic approximations of the physical processes of galaxy evolution onto merger trees constructed from N-body simulations, are one such way to study galaxies with dark-matter only simulations (White and Frenk, 1991; Kauffmann et al., 1993; Cole et al., 1994). These modeled physical processes often include dynamical friction, tidal stripping, and tidal heating, which estimate how a subhalo evolves as it falls into its host, along with approximations for how the galaxy is affected by the subhalo's evolution. The umbrella of ”semi-analytic models” encompasses models which approximate many different physical processes, but merging timescales (Taylor and Babul, 2001; Boylan-Kolchin et al., 2008), mass loss (Taylor and Babul, 2005), and final distributions of galaxies (Diaferio et al., 1999; Somerville et al., 2008) have been studied in detail and approximated using semi-analytic frameworks. Another approach to studying galaxy distributions is through halo models, which are parameterized models that dictate how to place galaxies inside of dark matter halos according to properties of those halos (Neyman and Scott, 1952; Peebles, 1974). The generated galaxy distribution can then be compared to observations, allowing for a fully empirical model of galaxy clustering down to small scales.

1.4 Understanding the Distribution of Structure on Different Physical Scales

It is of vital importance to be able to statistically characterize the distribution of structures in our universe in order to determine if theoretical distributions generated by simulations are truly representative of it. Typically, these statistics aim to describe the frequency and strength of clustering of different sized structures. On the observational side, it is common to probe the clustering of galaxies using the spatial two-point correlation function, which measures the excess probability of finding a pair of galaxies at some separation distance, as compared to a random distribution (Landy and Szalay, 1993). The frequency of the formation of galaxies of certain sizes can be probed using for instance the galaxy luminosity function, which measures the number density of galaxies in a certain luminosity bin (Schechter, 1976). In a dark matter only simulation, a halo mass function is a similar measurement, which instead counts the number of halos in the simulation that belong to a given mass bin (Jenkins et al., 2001). These measurements allow for theoretical distributions of structure from simulations to be compared to both each other and to the real universe in a straightforward and consistent way.

1.4.1 The Power Spectrum

One of the most crucial and basic cosmological observables to measure for density fields is the power spectrum. Under the assumption that the density field is traced by the number density distribution of dark matter particles, the power spectrum can be calculated from the dark matter particle distribution. Formally, the power spectrum is the Fourier transform of the two-point correlation function. It describes the statistical distribution of the amplitudes of density fluctuations as a function of the wavelength of those fluctuations. Functionally, the power spectrum of a density field can be calculated by computing a Fourier transform on the gridded density field to find underlying sine wave modes with wavenumber k, and measuring, in bins of k, the amplitude of those modes as P(k) (Feldman et al., 1994; Hand et al., 2018).

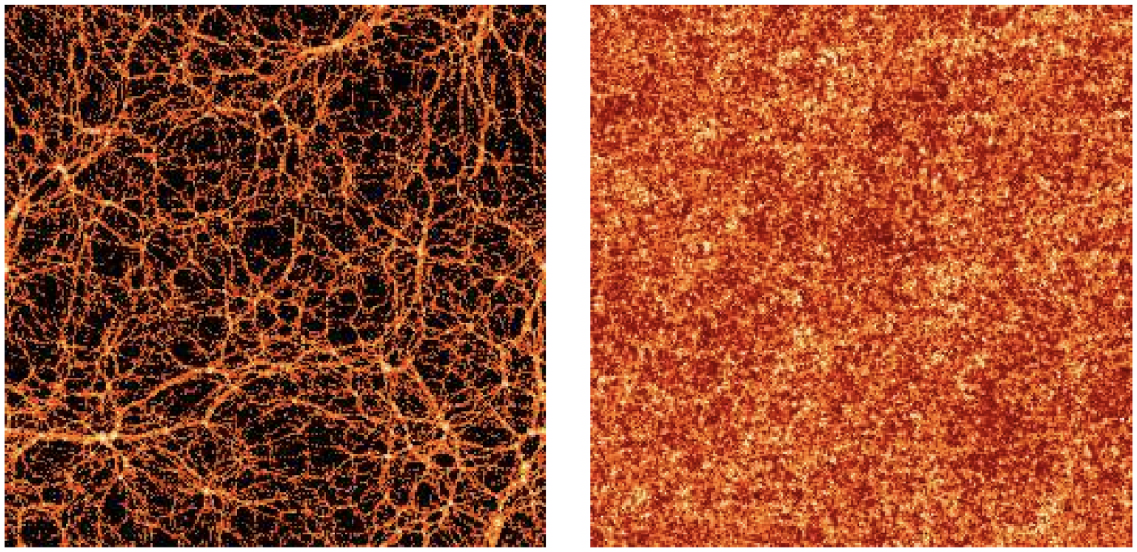


Figure 1.3: Two density field maps, with identical power spectra, but visually very different distributions of structure. The image on the left comes from a real N-body simulation. The image on the right is generated by randomly reshuffling the phases between Fourier modes of the image on the left. This figure is taken directly from Coles (2001), where it is shown as Figure 2.

The power spectrum measures the distribution of fluctuations, but crucially, not the *arrangement* of those fluctuations as phase information is not embedded in P(k). Therefore, an accurate power spectrum is necessary but not sufficient to ensure that a density field matches our universe. This is illustrated in Figure 1.3, which shows two maps with identical power spectra which are morphologically very different. On the left is an image of a real N-body simulation. The image on the right is constructed by randomly shuffling the phases between Fourier modes of the N-body simulation. This uncorrelates the phases, but the coupling of phases in later-stage evolution of the universe is precisely what is responsible for the sharply defined structures we see in the image on the left (Coles and Chiang, 2000).

1.4.2 Higher Order Statistics: The Bispectrum, Void Probability Function, and Cross-Power

Higher moments of the density field can act as more sensitive probes of the non-linearity, and many other statistics exist to examine specific properties of the structure distribution. The bispectrum - which is a Fourier transform of the three-point correlation function - is capable of encoding directional information that is lost by the power spectrum. In turn, it can be a better probe of the typical shapes and orientations of structures (Lazanu et al., 2017). Of course in theory, N-point correlation functions can be calculated, but are much more arduous to calculate and have not found to be generally as useful (Sharp et al., 1984). The void probability function (White, 1979), which describes the probability that a randomly placed sphere of radius R does not contain any galaxies or halos, allows for more specific probing of less highly non-linear regions of the density field and is tied to higher-orders of the correlation function (Fry, 1986; Perez et al., 2021). The cross-power spectrum measures the covariance between two different density maps, and can be used to quantify correlations in their phases (Stirling and Peacock, 1996). The aforementioned statistics, and others, can not only be useful when attempting to describe the patterns of structures in the universe, but also when determining if and how a simulated universe matches or deviates from its desired configuration when comparing simulations to the real universe or to each other.

1.5 Machine Learning as a Predictive Approach

Machine learning has the advantage of being able to model complicated, abstract relationships without any a priori knowledge about the target problem. Instead, the creation of successful machine learning models relies on the supposition that a model which has sufficiently large capacity can be trained to approximate any complex function. In astrophysics, predictive models like this are useful for a wide span of analyses. For some astrophysical modeling tasks, such as simulating the highly non-linear effects of running complete simulations, machine learning may provide robust mappings between input and output where otherwise analytical estimators are too difficult to specify precisely. For other astrophysical problems, we have a priori knowledge about how the physics of certain systems work, but machine learning may be able to learn complex additional effects that are not captured by our current analytic models alone.

1.6 Summary

The goal of the work presented in this dissertation is to investigate to what degree machine learning methods might mitigate some modern problems in astronomy that are a consequence of the need to use simulations. In Chapter 2, I use machine learning to explore structure formation on smaller scales, investigating the merging of subhalos into their host halos to probe the potential of modeling mergers without detailed simulations and the validity of the assumptions we presently make about these behaviors. In Chapter 3, I move to a more macro scale, and investigate the use of deep convolutional neural networks to predict the results of a cosmological N-body simulation and how these methods may help astrophysicists move toward bypassing running these simulations altogether. Finally, in Chapter 4, I provide a summary of the work presented here and discuss potential for future work.

CHAPTER 2

Machine Learning the Fates of Dark Matter Subhalos: A Fuzzy Crystal Ball

This chapter was previously published in the June 2021 edition of Monthly Notices of the Royal Astronomical Society[1] *(Petulante et al., 2021), and is reproduced here, with minor formatting changes, with the permission of the publisher and my co-authors, Andreas A. Berlind, J. Kelly Holley-Bockelmann, and Manodeep Sinha.*

The evolution of a dark matter halo in a dark matter only simulation is governed purely by Newtonian gravity, making a clean testbed to determine what halo properties drive its fate. Using machine learning, we predict the survival, mass loss, final position, and merging time of subhalos within a cosmological N-body simulation, focusing on what instantaneous initial features of the halo, interaction, and environment matter most. Survival is well predicted, with our model achieving 94.25% out-of-bag accuracy using only three model inputs (redshift, subhalo-to-host-halo mass ratio, and the impact angle of the subhalo into its host) taken at the time immediately before the subhalo enters its host. However, the mass loss, final location, and merging times are much more stochastic processes, with significant errors between true and predicted quantities for much of our sample. Only five inputs (redshift, impact angle, relative velocity, and the masses of the host and subhalo) determine almost all of the subhalo evolution learned by our models. Generally, subhalos that enter their hosts at a mid-range of redshifts (z = 0.67-0.43) are the most challenging to make predictions for, across all of our final outcomes. Subhalo orbits that come in more perpendicular to the host are easier to predict, except for in the case of predicting disruption, where the opposite appears to be true. We conclude that the detailed evolution of individual subhalos within N-body simulations is difficult to predict, pointing to a stochasticity in the

[1]Because this chapter was originally published in a journal in the UK, British English spelling conventions are used in this chapter.

merging process. We discuss implications for both simulations and observations.

2.1 Introduction

According to the standard ΛCDM model of cosmology, dark matter structures in the universe form hierarchically through a series of mergers, with larger halos continuously growing from the accretion of smaller halos. Once independent halos themselves, these ”subhalos” sink to the center of their ”host” halos, losing mass to their hosts along their orbits due to tidal effects and dynamical friction, a process which has been studied in detail (Tormen et al., 1998; Weinberg, 1989; van den Bosch et al., 1999; Hayashi et al., 2003; Taffoni et al., 2003; Gan et al., 2010; van den Bosch, 2017). A significant number of such subhalos retain some of their mass, surviving as substructures within their hosts today. The study of these substructures has been fundamental to our understanding of many areas of astrophysics, from large-scale structure (Zentner and Bullock, 2003; Knebe et al., 2002; Zentner et al., 2005; Watson et al., 2011) to the formation and evolution of galaxies (Hayashi and Chiba, 2009; Kazantzidis et al., 2009; Simha and Cole, 2017), which rely on both accurate final subhalo populations and the evolution of these populations (Diemand et al., 2007; Giocoli et al., 2008).

Theoretical models of galaxy formation and evolution are commonly put to the test through analytic frameworks of subhalo evolution (Taylor and Babul, 2004; Zentner et al., 2005; van den Bosch et al., 2005; Penarrubia and Benson, 2005; Jiang and van den Bosch, 2016). These techniques typically rely on merger trees constructed from N-body simulations, along with analytic treatments of the physical processes that cause their evolution, which allow for an in-depth study of the individual effects of dynamical friction, tidal stripping, and tidal heating. Semi-analytic models often perform quite well, producing galaxy populations that match hydrodynamic simulations (Croton et al., 2006; De Lucia and Blaizot, 2007; Somerville et al., 2008; Henriques and Thomas, 2010; Hirschmann et al., 2012; Mitchell et al., 2018). Other works have focused on developing analytical models that reproduce specific properties of subhalos, such as disruption and mass loss rates, spatial distributions within host halos, and merging timescales, to better understand our assumptions about these driving physical processes.

Although there has been significant focus on constructing semi-analytic models of subhalo evolution, the *reliability* of tuning these models to N-body simulations has remained relatively unexplored. N-body simulations do produce consistent subhalo mass functions, the evolution of which has been thoroughly studied (Gao et al., 2004; Onions et al., 2012; Jiang and van den Bosch, 2017; Chua et al., 2017) even down to 10^7 solar masses (Munshi et al., 2019). However, though the ensemble of subhalos is well-characterized, it is not clear that the evolution of an *individual* subhalo within these simulations is a truly deterministic process. Subhalo evolution models that are tuned to N-body simulations (Penarrubia and Benson, 2005; Gan et al., 2010; Hiroshima et al., 2018) reflect the noise and uncertainty within the simulation. It may be that subhalo evolution within N-body simulations is somewhat stochastic, such that nearly identical interactions evolve differently, adding unforeseen complications when using these models.

In this work, we use halo merger tree data generated from the dark matter only simulation VISHNU, described in Johnson et al. (2019), to quantify the evolution and fate of subhalos. We attempt to predict the final subhalo state using initial conditions at the time the subhalo enters its host. Using a large range of physically-motivated quantities at the time of a subhalo's entry, we train machine learning algorithms to predict final properties for the subhalo, in the hopes of investigating to what degree its fate is determined by these parameters and to what degree the interaction is stochastic and cannot be predicted. If the amount of mass loss, for example, is deterministic, a machine learning algorithm should be able to successfully map subhalo initial conditions to its final mass. On the other hand, if there is a level of stochasticity in subhalo fate, there will remain large prediction errors, even when using a complete set of inputs that describe its initial state.

Machine learning has emerged as a powerful tool in astrophysics with a variety of applications, such as galaxy classification (Barchi et al., 2020; Nolte et al., 2019), exoplanet detection (Schanche et al., 2019b), and gravitational wave noise removal (Cavaglia et al., 2019), and has recently been used with cosmological simulations to predict galaxy properties from halo properties (Kamdar et al., 2016; Moster et al., 2020), populate halos with galaxies (Agarwal et al., 2018; Jo and Kim, 2019), connect initial conditions to final halos (Lucie-Smith et al., 2018, 2019), predict the formation of large scale structure (Feder et al., 2020; Li et al., 2020), and predict the halo masses of galaxies (Calderon and Berlind, 2019) and clusters (Ntampaka et al., 2015). The ability of machine learning models to approximate any function with a large set of parameters provides a useful means of revealing complex correlations when a direct analytic function cannot be found. Notably, Nadler et al. (2018) recently used machine learning to predict the survival or disruption of subhalos in a hydrodynamic simulation, using the initial conditions of their counterparts in a dark matter only simulation. They were quite successful, accurately predicting the results of 85% of their test set of subhalos. Works like this are encouraging that machine learning can be used to fit these complicated interactions.

Here, we focus on dark matter only simulations and aim to predict not only survival, but more quantitative metrics such as the amount of mass loss, the final position, and the time to final merger for a subhalo. By using a comprehensive set of model inputs that describe the physical state of the subhalo to make these predictions, we hope to reveal what properties of the subhalo are the most closely tied to – and thus what physical processes most strongly drive – this evolution. While we expect these additional quantities to be more difficult to predict, even in a dark-matter only simulation with simpler physics, than a binary prediction for disruption, the ability or inability of machine learning models to make predictions in the first place can inform us about the determinism of a model. Predictions that are not successful, despite having a complete set of physical descriptors available to them, may indicate that subhalos in N-body simulations do not evolve in a straightforward, predictable way. This could be due to a number of factors, ranging from resolution effects, to halo catalog or merger tree errors, to an inherent chaotic nature of these interactions.

The topic of subhalo disruption has a particularly rich body of work, which will guide us in selecting our initial suite of halo properties to use as model inputs. Accretion redshift has repeatedly been found to be overwhelmingly important in determining the survivability of subhalos, with the majority of surviving subhalos being accreted more recently than z=1 (Ghigna et al., 2000; Diemand et al., 2004; Gao et al., 2004; Zentner et al., 2005; Penarrubia and Benson, 2005; Diemand et al., 2007). The abundance of subhalos is also found to be lower for host halos of fixed mass that have higher concentrations. Because higher concentration halos on average form earlier, they have less surviving substructure because their substructure is accreted earlier and spends more time orbiting inside the host (Gao et al., 2004; Giocoli et al., 2010; Gao et al., 2011; Mao et al., 2015). Trends with the orbits of subhalos find that many subhalos that are destroyed do not complete even one pericenter passage, but many subhalos that do survive have completed more than one pericenter passage. However, it has also been found that surviving subhalos tend to have more eccentric orbits (Klimentowski et al., 2010) . Slower subhalos with low orbital energies are preferentially destroyed, resulting in a positive velocity bias in the subhalo distribution within clusters (Diemand et al., 2004). The fraction of surviving subhalos can be well modeled as a function of the subhalo-to-host mass ratio (Tormen et al., 1998), and subhalos with larger mass ratios have been found to more rapidly disrupt (Tormen et al., 1998). It has also been found that there is a weak dependence of the subhalo disruption rate on the mass of the host halo (Gill et al., 2004b).

As subhalos typically disrupt after losing a significant fraction of their mass (Taylor and Babul, 2005), we expect many of the properties that determine subhalo survival to also be of significant importance to subhalo mass loss. For populations of subhalos across a wide variety of host halos, subhalo mass loss does not appear to strongly depend on the mass of the host halo (Gao et al., 2004). In a static host potential, the eccentricity of the subhalo orbit and the subhalo concentration are the dominant determinants of mass loss, with subhalos losing a significant portion of their mass during each pericenter passage, resulting in more radial orbits losing mass more quickly (Taylor and Babul, 2004). Average mass loss rates of subhalos using only the redshift and subhalo-to-host mass ratio appear to have good agreement with subhalo mass functions from simulations (van den Bosch et al., 2005). Additionally, many analytical models of subhalo positions and internal structure at each timestep have been created to model subhalo evolution (Taylor and Babul, 2001; Hayashi et al., 2003; Kampakoglou and Benson, 2007; Gan et al., 2010; Han et al., 2016). Although these works give deeper insights to the relative importance of the physical processes at work on these subhalos, our machine learning models do not explicitly model the evolution over time of our subhalos, so the properties used by those works are less relevant here.

The final distributions of subhalos within their hosts have also been closely studied. Radial distributions of subhalos within their hosts do not appear to depend on host halo mass or redshift, but do depend on the subhalo-to-host mass ratio (Angulo et al., 2009). Subhalos that merge with the central parts of their host halos also tend to have larger subhalo-to-host mass ratios than those merging with the outer parts of the host halo, which may suggest that mass ratio can help predict the final location of a subhalo (Nipoti et al., 2018). Distributions of subhalo spins show lower spins closer to the host center, suggesting that lower spin halos may have more success at surviving to z=0 when orbiting at small fractions of the host radius (Reed et al., 2005). If this is due to higher spin subhalos being more susceptible to tidal stripping, this trend could also be important for our other predicted quantities.

Analytic predictions of merging timescales for subhalos have been found by a number of previous works. A function to determine the time until satellite removal after accretion can be successfully fit using only the subhalo-to-host mass ratio (Wetzel and White, 2010). The effects of dynamical friction can be accurately modeled to determine galaxy merging timescales, using the subhalo-to-host mass ratio, the circularity and energy of the subhalo orbit, the virial radius of the host halo, and the dynamical time at the host halo's virial radius (Boylan-Kolchin et al., 2008; Jiang et al., 2008; McCavana et al., 2012). Although the dependencies on these parameters are different in these different works, the parameters dominating the merging timescale remain the same.

In Section 2.2, we describe the simulation and input data. In Section 2.3, we cover the machine learning methods we use to create our predictive models. In Section 2.4.1, we discuss the parameter selection methods we use to gain intuition and decide on which parameters are the most important for each model to make predictions. In the rest of Section 2.4, we share the results of our models for predicting each of our outcomes, including their performance and which parameters were needed as inputs to the model. Finally, in Section 2.5 we discuss implications of our results for both observation and theory.

2.2 Description of the Data

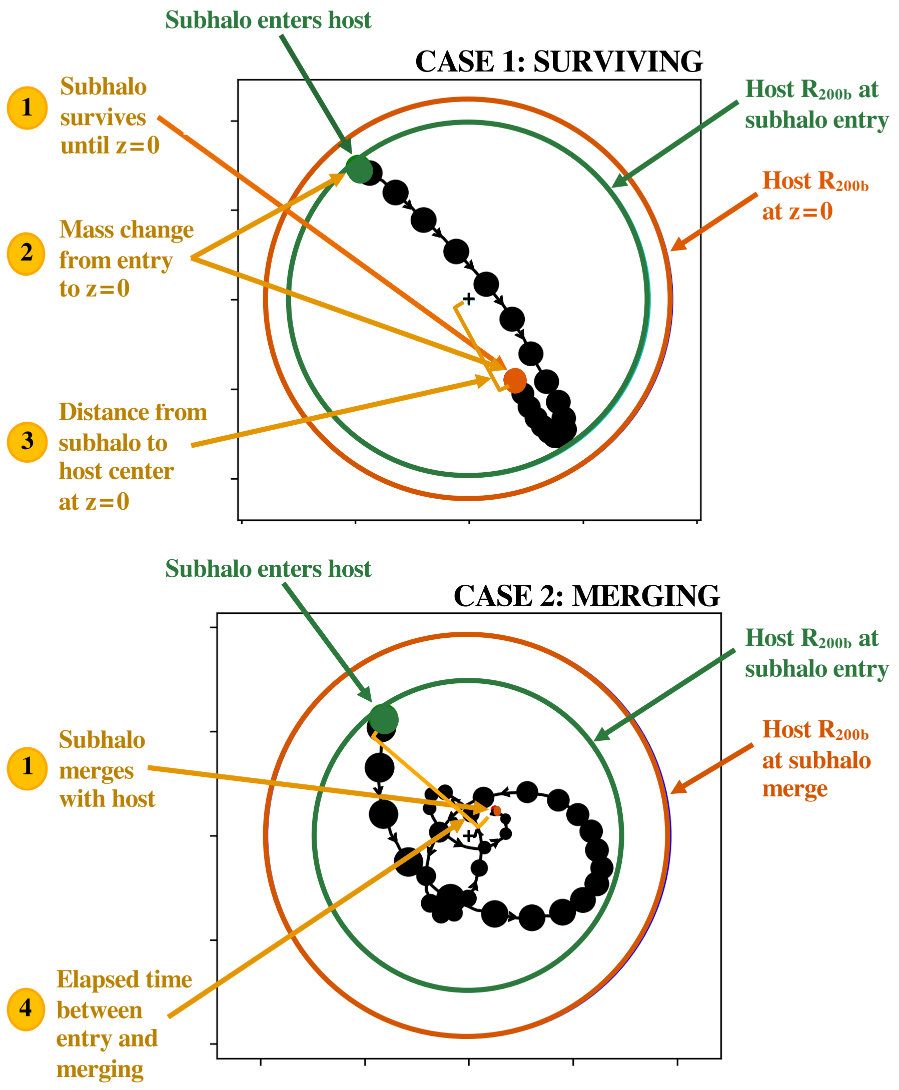
Our analysis makes use of VISHNU, a cosmological N-body simulation with 1000 snapshots for exquisite time resolution; no snapshot is separated by more than 3.2×10^7 years. VISHNU contains 1680^3 dark matter particles of mass $m_p = 3.215 \times 10^7 \mathrm{h}^{-1}\mathrm{M}_\odot$ in a box of size 130 h^{-1}Mpc and uses WMAP-1 cosmology (Spergel et al., 2003); $\Omega_\mathrm{m} = 0.25$, $\Omega_\Lambda = 0.75$, $\Omega_\mathrm{b} = 0.04$, $\sigma_8 = 0.8$, $n_s = 1.0$, $h = 0.7$). The initial positions and velocities of the particles at redshift z = 599 were then determined using the 2LPT code (Scoccimarro, 1998). The simulation was evolved to z = 0 using the GADGET-2 N-body TreeSPH code (Springel, 2005), adopting a force resolution of 2.2 h^{-1}kpc. The ROCKSTAR halo finder was used to identify halos and subhalos (Behroozi et al., 2013a), adopting a spherical overdensity halo definition with a threshold density equal to 200 times the background density of the universe. We denote the mass and radius of such halos with $\mathrm{M}_{200\mathrm{b}}$ and $\mathrm{R}_{200\mathrm{b}}$. Finally, merger trees were constructed using the code `Consistent-Trees` (Behroozi et al., 2013b).

Starting with halos at z=0, we use the merger trees to identify the most massive progenitors of all host halos within the simulation, and track the subhalos within. These subhalos are allowed to host further substructure but cannot, at any point during their infall, become sub-substructure themselves. To help mitigate resolution uncertainties, we select only subhalos with a minimum of 1000 particles (total mass $3.215 \times 10^{10}\text{h}^{-1}$ $M_{\odot}$) at their time of accretion. We define the accretion time as the last snapshot before a subhalo enters its host. This mass cut reduces our sample from over 1,250,000 to 121,343 subhalos.

In addition to this resolution cut, some interactions were removed due to their unphysical behavior, likely as a result of errors in the merger tree generation or halo finder. Specifically, 1474 subhalos were removed that more than tripled their mass during infall, likely due to swapping identities with another halo. In addition, 327 subhalos were removed because their initial mass was larger than that of their supposed hosts. Taken together, these cuts culled about 1.5% of halos, leaving 119,543 subhalo-halo interactions in our final sample.

Once a subhalo has been accreted by a host, it must have one of two fates: disrupt within the host (*merge*), or remain a bound, identified subhalo within that host until today (*survive*). We define the merger time as the last snapshot at which a subhalo is identified as its own entity. However, subhalos are highly sensitive to artificial disruption after they have lost a significant amount of their mass, as has been shown by a number of studies that have shown that "orphan satellites" are required to match the small-scale clustering of galaxies in semi-analytic models (Wang et al., 2006; Guo and White, 2014; Campbell et al., 2018). Following uncertainties in subhalo mass loss shown by van den Bosch and Ogiya (2018), we attempt to preempt this issue by also considering a subhalo to be merged when it has lost more than 90% of its mass, provided it remains underneath this threshold for the remainder of the simulation. This cut changes the fates of 21,929 subhalos, around 18% of our total sample, but ensures a more consistent definition of merging that is less sensitive to resolution errors.

Figure 2.1 illustrates the two possible subhalo fates using actual examples from our data set. The top panel shows a surviving interaction, where the subhalo orbits for some time,



Subhalo enters host
CASE 1: SURVIVING
1 Subhalo survives until z=0
2 Mass change from entry to z=0
3 Distance from subhalo to host center at z=0
Host R_{200b} at subhalo entry
Host R_{200b} at z=0
Subhalo enters host
CASE 2: MERGING
1 Subhalo merges with host
4 Elapsed time between entry and merging
Host R_{200b} at subhalo entry
Host R_{200b} at subhalo merge

Figure 2.1: An example of a surviving (top) and merging (bottom) interaction between a subhalo and host halo. The large circles show the radii of the host halos at the beginning and end of the interaction. The orbits of the subhalos are shown with the series of filled circles, plotted at each eighth timestep, and with a point size corresponding to subhalo mass along the orbit. The green point and circle show initial quantities, at the timestep right before the subhalo enters its host. The orange point and circle show final quantities, at either the timestep right before the subhalo dissolves in the merging case, or at the final timestep in the simulation in the surviving case. Predicted quantities (gold) are labeled and numbered in the order we will present them throughout the paper.

losing mass but not dissolving before z=0, while the bottom panel demonstrates a merger. In all, 76,442 (64%) are mergers and 43,101 (36%) survive. The four quantities we predict are shown in yellow and numbered by the order that they will be presented in Section 2.4.

Note that the binary fate, survival or merger, uses the whole sample, but mass loss and final position considers only surviving halos and merging time applies only to those halos which merge.

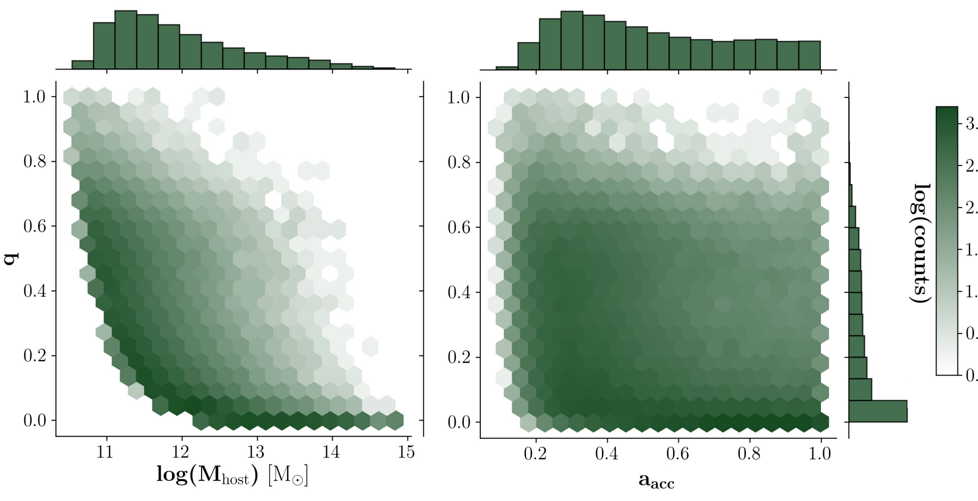


Figure 2.2: Demographics of our sample of interactions. The left panel shows the distribution of mergers as a function of host mass and mass ratio, defined at the time of accretion. The cosmological scale factor of accretion versus mass ratio is shown on the right. The color represents number of mergers in each hexagonal pixel on a logarithmic scale, as denoted on the far right. Histograms show the one-dimensional distribution of the variable on the corresponding axis. Most mergers occur at lower mass ratios and for smaller host halos, but the spaces are still well-spanned over a range of interactions in both panels. The chosen cut of 1000 particles ($M = 3.215 \times 10^{10} h^{-1}\ M_\odot$) for subhalos upon entry is clearly shown in the left panel. Imposing this cut and requiring that host halos be larger than their subhalos means that, for very small host halos, the only subhalos within our dataset are those with masses more similar to their hosts.

Distributions of our final sample with respect to host masses, mass ratios, and the scale factor of the time of entry are shown in Figure 2.2. The most common interactions are those of unequal masses that occurred more recently. However, our sample also spans the space of more equal mass and higher redshift interactions, with hundreds of interactions shown in many of the bins in Figure 2.2. The effects of our chosen particle cut can also be clearly seen in Figure 2.2. Because subhalos must have 1000 particles at their time of entry, this results in a minimum initial mass of $3.215 \times 10^{10} h^{-1}\ M_\odot$ for both the subhalo and host halo, given that a host halo must also be at least as large as its subhalo. In the left panel of Figure 2.2, this results in an area of no data with low mass hosts and unequal masses, because most subhalos of low mass hosts are too low mass to be included in our sample. We also do not have many interactions between very large hosts halos and similarly large subhalos. This is because there are relatively few massive halos in the simulation, so interactions between them are expected to be very rare.

2.2.1 Used Interaction Parameters

Here we describe our set of physically-motivated parameters to characterize an interaction. Our parameters comprise four categories: 1) Global interaction parameters give information about the interaction that is not specific to the particular system; 2) Internal halo parameters that are properties of the individual halos themselves and describe their size, shape, or structure; 3) Orbital parameters provide information about the initial trajectory of the subhalo's infall path; and 4) Environmental parameters describe the influence of larger scale environment around the subhalo and its host. In total, this yields 26 parameters, which we list and define here. All of these quantities are measured at the time of accretion, which is defined as the last snapshot before a subhalo enters its host.

2.2.1.1 Global Interaction Parameters

- $\mathbf{a_{acc}}$: the scale factor of the universe when the subhalo is accreted, defined as the snapshot right before the subhalo is flagged as a subhalo of its host.

- **q**: the ratio of subhalo to host halo masses.

2.2.1.2 Internal Halo Parameters

- $\mathbf{M_{sub}}$: Mass of the subhalo (just before it becomes a subhalo) at 200 times the background mass density of the universe, M_{200b}. This is defined using only those particles that were assigned to the subhalo and the sub-substructure within it.

- $\mathbf{M_{host}}$: M_{200b} of the host halo, as described above. Note that this mass is calculated by including the particles of all substructure within the halo.

- **$\mathbf{R_{sub}}$**: Radius of the subhalo at the point where the mean subhalo density is 200 times the background mass density of the universe, R_{200b}. Because this radius is calculated as the spherical overdensity radius of a halo with mass M_{200b}, this value contains identical information to M_{sub}.

- **$\mathbf{R_{host}}$**: R_{200b}, as described above, of the host halo. This value contains identical information to M_{host}.

- **$\mathbf{c_{sub}}$**: the concentration of the subhalo, defined as R_{200b}/R_s, where R_s is the scale radius of the subhalo.

- **$\mathbf{c_{host}}$**: the concentration of the subhalo, defined as R_{200b}/R_s, where R_s is the scale radius of the host halo.

- **$\lambda_{\mathbf{sub}}$**: Bullock spin parameter of the subhalo, defined as in Bullock et al. (2001).

- **$\lambda_{\mathbf{host}}$**: Bullock spin parameter of the host halo.

- **$\mathbf{T_{sub}}$**: the triaxiality parameter of the subhalo. Calculated from the definition given in Franx et al. (1991):
$$T = \frac{1-(b/a)^2}{1-(c/a)^2} \tag{2.1}$$
where b/a is the minor/major axis ratio and c/a is the intermediate/major axis ratio.

- **$\mathbf{T_{host}}$**: the triaxiality parameter of the host halo, calculated as above.

- **$\mathbf{max(M_{subs,sub})}$**: M_{200b} of the most massive sub-subhalo within the subhalo. 0 if subhalo has no sub-substrucutre.

- **$\mathbf{max(M_{subs,host})}$**: M_{200b} of the most massive subhalo already within the host halo at the time of the selected subhalo's entry. Does not include the selected subhalo. 0 if no other subhalos are present.

- **$\mathbf{N_{subs,sub}}$**: the total number of sub-subhalos within the subhalo.

- $\mathbf{N_{subs,host}}$: the total number of subhalos within the host halo. Does not include the selected subhalo.

2.2.1.3 Orbital Parameters

- $\mathbf{d_{rel}}$: distance between the centers of the subhalo and host halo.

- v_{rel}: magnitude of the relative velocity between subhalo and host halo, calculated in the reference frame of the subhalo.

- ε: eccentricity of subhalos initial orbit. Calculated as described in Wetzel (2011):

$$\varepsilon = \sqrt{1 + \frac{2EL^2}{(GM_{\text{host}}M_{\text{sub}})^2\mu}} \tag{2.2}$$

  In this definition, ε = 1 is a perfectly elliptical orbit, and orbits that are initially unbound have eccentricities greater than 1.

- ϕ: impact angle of subhalos initial orbit. Calculated as $L_{\text{total}}/L_{\text{max}}$, the ratio between the total angular momentum of the subhalo orbit and the angular momentum of an orbit with the same velocity magnitude and orbital radius, but with the entire velocity component in the direction perpendicular to the direction of the host center. We refer to this as an impact angle because:

$$\frac{L_{\text{total}}}{L_{\text{max}}} = \frac{m(\vec{v} \times \vec{r})_{\text{total}}}{m(\vec{v} \times \vec{r})_{\text{max}}} = \frac{mvrcos(\theta)}{mvr} = cos(\theta) \tag{2.3}$$

  In this definition, ϕ = 1 is an orbit coming in perfectly perpendicular to the axis between the host and subhalo. We call this a grazing impact angle. Alternately, $\phi = 0$ would be an orbit coming in perfectly along the axis between subhalo and host halo, which we call a plunging impact angle. We note that, while an orbit with L_{max} would be instantaneously circular, this is different from the typical definition of circularity, $L_{\text{total}}/L_{\text{circ}}$, which compares the total angular momentum of the subhalo orbit and the angular momentum of a stable circular orbit with the same energy. As circularity is simply mathematically related to our definition of ε, it would contain identical information.

2.2.1.4 Environmental Parameters

- $\mathbf{F_{tid}}$: Magnitude of the tidal force on the subhalo, approximated as the tidal force from the neighboring halo within d = 4h^{-1}Mpc that contributes the most to the tidal force. Excludes the subhalo's own host. This value is calculated using the full halo catalog at the relevant a_{acc}.

$$F_{\mathrm{tid}} = \frac{M_{\mathrm{neighbor}}}{{d_{\mathrm{neighbor}}}^3} \tag{2.4}$$

- $\rho_{\mathbf{1Mpc}}$: The density due to neighboring halos in a surrounding sphere with radius r = 1h^{-1}Mpc from the subhalo center, not including the subhalo's own host halo. This value is calculated using the full halo catalog at the relevant a_{acc}.

$$\rho_{\mathrm{1Mpc}} = \sum_i \frac{M_{\mathrm{i}}}{\frac{4}{3}\pi r^3} \tag{2.5}$$

- $\rho_{\mathbf{2Mpc}}$: The density due to other halos in a surrounding sphere with radius r = 2h^{-1}Mpc from the subhalo center not including the subhalo's own host halo.

- $\rho_{\mathbf{4Mpc}}$: The density due to other halos in a surrounding sphere with radius r = 4h^{-1}Mpc from the subhalo center not including the subhalo's own host halo.

2.2.2 Predicted Quantities

We aim to use machine learning to predict the following for each subhalo:

1. **survival**: a (categorical, binary) indication of whether or not a subhalo survives until z=0. 0 or 1, depending on whether the subhalo exists above the required mass threshold at z=0 (survives, 1) or if the subhalo has fallen below the mass threshold at some

time before z=0 (dissolves, 0). This mass threshold is defined as 10% of the subhalos mass upon accreting into the host (M_{sub}). Predicted for all subhalos.

2. **$M_{sub,f}$**: the M_{200_b} (as described above for M_{sub}, calculated using only particles which belong to the subhalo) of the subhalo at z=0. Only predicted for surviving subhalos. When compared to the initial subhalo mass, this shows the amount of mass loss that the subhalo experiences.

3. **$d_{rel,f}$**: relative absolute total distance between subhalo and host halo centers at z=0, normalized by the radius of the host halo (R_{host}, as described above) at z=0. Only predicted for surviving subhalos.

4. **t_{merge}**: the elapsed time between the accretion of a subhalo into the host and its dissolution within the host. Only predicted for dissolving subhalos.

2.2.3 Data Normalization

Machine learning, requires data to *train* and *test*; we randomly split our data into subsamples, with 80% for training and 20% as a final holdout test set. The training set will then further be split into training and validation sets when tuning our models. We scale and normalize the data using `StandardScaler` from the `scikit-learn preprocessing` package such that each quantity, X, if assumed Gaussian, is distributed with zero mean and unit variance:

$$X_{\text{norm}} = \frac{X - \mu}{\sigma} \tag{2.6}$$

where μ is the mean of the unscaled data and σ is the standard deviation. This scaling is necessary for many machine learning models, as large variations dynamic range over a set of observables can affect model accuracy.

2.3 Machine Learning Methods

The machine learning algorithms we use come from the `scikit-learn` package for python. Subhalo survival is a classification problem, so we use a random forest algorithm. On the other hand, predictions of the amount of mass loss, the final position, and merging time are all classic regression problems, so we use the gradient boosting regressor algorithm. Although we refer the reader to the `scikit-learn` documentation for a full description of these algorithms, we briefly describe these methods below.

2.3.1 Random Forest

Random forest classifiers use an ensemble of decision trees to reach consensus on a prediction. These decision trees repeatedly split the data into bins based on the values of its input parameters, resulting in gradually smaller subsets of data belonging to each bin. The goal of the algorithm is to find bins that span a section of the input parameter space where almost all members of the bin have the same output value. Then, the assumption is that test data points with input parameters that fall in a certain bin will usually have the same output value as other members of that bin. In a random forest, many individual decision trees are trained on random subsets of the training dataset. Because random forests are a type of bagging - or bootstrap aggregating - method, the classification for an object is then the majority vote of all of the trees. This means that the trees that make up the ensemble are distinct, making an independent prediction for the classification of an object in the testing set.

There are several *hyperparameters* of the algorithm that we tune in order to get the best-fitting model. These hyperparameters are parameters of the model itself, that determine properties such as the complexity of the model or the way that it learns. Their values are fixed before the model is trained, and are not adjusted during the training of the model. The hyperparameters that we set for the random forest classifier are as follows. The *n_estimators* hyperparameter sets the number of estimators, in this case decision trees, used in the final consensus. Too few decision trees removes the power of using multiple trees. In general, using too many trees is not a concern, though it does increase the runtime of the algorithm. The *max_depth* hyperparameter sets the maximum number of decisions in each tree. Effectively, this sets the maximum number of input parameters each decision can use, since each depth splits on one parameter. The *max_leaf_nodes* hyperparameter sets the maximum number of nodes at a given depth. We note that, if this value is too small, the decision tree may split on the same parameter at multiple depths. We keep other hyperparameters at default values (see `scikit-learn` documentation).

One of the main advantages of random forest algorithms is that they are less prone to overfitting, especially compared to a single decision tree. Because each decision tree works with a subset of the data and considers a random group of parameters, the ensemble is more robust to unseen data. This is important in cases like ours in which a large number of training examples determine a small number of phenomena. Random forests are also useful in their ability to deal with correlations between input parameters, such as halo mass and concentration. Unfortunately, the results of a random forest are much less straightforward to interpret than that of single decision trees, and any given decision tree within the forest may be a very poor predictor. Nonetheless, random forest classifiers robustly rank the importance of each input parameter, based on their frequency and proximity to the top of the decision trees within the forest. However, strong correlations between input parameters can make this ranking difficult to straightforwardly interpret as well, which we discuss in more detail in Section 2.4.1.

Subhalo survival is particularly amenable to binary classification; we assign 0 to a subhalo for dissolving before z=0, and a value of 1 for surviving until z=0. We train multiple models, using the same hyperparameters, on increasingly smaller subsets of input parameters to confirm the minimum number needed to make accurate predictions. To determine the order of parameter removal, and for presentation purposes in our figures, we use the custom parameter selection algorithm outlined in Section 2.4.1 to determine the relative importance of our parameters.

2.3.2 Gradient Boosting Regressors

Gradient boosting regressors, like random forests, rely on an ensemble of decision trees to make predictions. However, unlike random forests, gradient boosting regressors construct trees that are dependent on the results of all the trees trained before them. New trees are fit to the errors of the current ensemble; the purpose of each new tree is to learn the errors of the current model and to iteratively hone in on the prediction.

As with the random forest classifier, several hyperparameters can be tuned to create the best model. The *learning rate* weights the significance of a new tree within the ensemble; small learning rates significantly increase the number of trees that need to be added to the model, but too large a learning rate may result in corrections that perpetually overshoot the prediction. As with the random forest classifier algorithm, we also tune the *n_estimators*, *max_depth* and *max_leaf_nodes*. Other hyperparameters are kept as their default values.

The advantage of gradient boosting regressors is again the reduction of overfitting because it is an ensemble method. Additionally, given that we also expect relatively few parameters to be important in making our regression predictions, a model that is able to avoid selecting unimportant parameters is favorable. The main advantage that a gradient boosting ensemble has over a random forest is that the trees work together to make a prediction. Because each tree added to a gradient boosting model corrects errors from the previous iteration of the ensemble, data points that were difficult to make predictions for are preferentially corrected for in later trees, making a gradient boosting ensemble better at dealing with outliers. This is particularly important for our regression problems given the large ranges of outcomes for all of our predicted quantities. As with the random forest classifier, the gradient boosting regressor class in `scikit-learn` also contains a function to report relative feature importances. However, interpreting these results again presents difficulties due to strong correlations between parameters.

We use gradient boosting regressors to predict mass loss, final position, and merge time individually. As with the survival classification problem, due to the difficulty of interpreting the reported feature importances of the algorithm, we use our custom algorithm, outlined in Section 2.4.1 to determine the order and relative importance of each parameter.

2.3.3 Model Training

When training any machine learning model, the choice of hyperparameters is critical for the model's ability to learn. So we begin by finding a set of hyperparameters for each of our models that leads to the best fit for our data. We do this by repeatedly creating models with different hyperparameters and evaluating their performance using training and validation sets that are random subdivisions of our total training set, using an 80%/20% split. We then check if the model performs well without overfitting too strongly. From our complete dataset, these repeated divisions mean that 20% of our data is used only for final testing and never given to a model while tuning hyperparameters, while 80% of our data set is used to tune the model by creating different training and validation sets. The use of these validation sets allows us to fine tune the properties of the model so that it best fits the training set, while leaving the testing set untouched to ensure that the model performance at testing time accurately shows the model's ability to generalize to new data. We use 5-fold cross-validation when determining a new set of hyperparameters' performance. The number of different hyperparameter values we need to try before arriving at a set of hyperparameters that best fits the data is different for each quantity we predict.

We begin this process by using `scikit-learn`'s `GridSearchCV` function, which accepts arrays of values for the desired hyperparameters to be tuned, then repeatedly trains the model on all combinations of the chosen hyperparameter values and evaluates each combination's performance. This function includes a `best_params_` attribute, which will return the combination of hyperparameters that leads to the best performance. We then, by inspection, fine-tune the hyperparameters within small ranges around this set. For example, we may begin with a depth hyperparameter of 5 for our model, then check if either raising the depth to 6 or lowering it to 4 with other hyperparameters held constant will give us improvement. We repeat this process for all hyperparameters, until we find a model that achieves high accuracy on both the training and validation sets, indicating that the model is well fit to the data, but with the smallest differences between training and validation accuracy, indicating that the model is minimally overfit. We take this extra step of fine-tuning the hyperparameters by hand to ensure that the model is well fit to our specific accuracy metrics for each prediction, which we motivate from physical quantities about the subhalo and host halo. These specific accuracy metrics that we use for each of our predicted quantities are described further in Section 2.4.

The best set of hyperparameters for each of our models can be found in Table 2.1. We note that, while we find these hyperparameters to create models that fit the data well, different choices of the hyperparameters can yield equally good models. Typically, slight changes to these hyperparameters did not lead to significant performance differences for our models. In the case of *max_leaf_nodes*, we found that changing its value, even significantly, did not have strong effects on the model performance for any of our models, and thus we kept it fixed to its default value (`None`) for all of our models. However, large changes to the other hyperparameters can cause significant changes to the model's ability to fit the data, so using a selection of a well-fitting set of hyperparameters is imperative to getting the best results. The choice of loss function to optimize is another free parameter. We found the best results using the mean squared error criterion for the random forest regressor and the Huber loss for the gradient boosting regressors.

2.4 Results

In the following subsections, we discuss our exploration of the parameter space and the performance of each of our machine learning models. In Section 2.4.1, we detail our parameter selection methods, where we find a preliminary order of the importance of our parameters

Table 2.1: The best fit hyperparameters for each of our models. Definitions of these hyperparameters and how they were selected are detailed in Section 2.3.

	Survival	Mass Loss	Position	Merge Time
method	RF	GBR	GBR	GBR
n_estimators	50	600	1800	500
max_depth	7	3	6	5
max_leaf_nodes	None	None	None	None
learning rate	N/A	.07	.008	.05

for predicting our final quantities. Then, in the subsections that follow, we detail the results of our machine learning models, including a discussion of our metrics for determining the accuracy of our predictions, how well each of the models perform, and which parameters were the most important for making the predictions for each model. In Section 2.4.2, this is discussed in detail for predicting the survival quantity. In Section 2.4.3, we discuss this in detail for the mass loss quantity. In Section 2.4.4 we discuss the final position quantity, and in Section 2.4.5 we discuss the merging time quantity. In Section 2.4.6, we investigate the frequency of subhalo interactions that we find within our sample and their potential effects on our predictions.

2.4.1 Feature Selection

We select 26 features to describe each subhalo/host halo interaction. These parameters are selected to encompass information about the orbit, environment, and individual properties of both the host and subhalo. Although we begin with this large set of features for thoroughness, we expect that not all of them will be important for predicting our desired quantities. To determine which features most strongly affect the predicted quantities, we use a feature selection method to select four features from the complete set, for each predicted quantity, which are responsible for the most variation in that quantity. Because our set of features has strong correlations between several values, we also aim to use a selection method that minimizes correlations in the selected set. We emphasize that we do not remove any features using these methods. Instead, we use the order of the selected set to determine the order in which we will add features to our models and for display purposes in our figures. During training, all models still use all features, to ensure that no information is missed.

When building decision trees, higher ranked features are both the most important, and those that cause the most variance in the final quantity. Our aim, then, is to order the features by the amount of variance, while binning to remove correlations with other features; this leads to an independent ordering that appears to add information to the prediction most quickly. We describe the feature selection method using the example of predicting the final mass of surviving subhalos. We begin by binning the data by each of our features, say impact angle, into bins with equal numbers of subhalos, and calculating the mean final mass in each bin. We then determine the range of these binned mean values as a measure of the strength of the correlation between final mass and impact angle. We adopt the feature with the largest range as the most important. This algorithm selects the subhalo radius as the most important feature for predicting the final subhalo mass. To select the next most important feature, we bin the data in two dimensions, where the one dimension is our adopted primary feature and for the second dimension we try each of the remaining features. In each two-dimensional bin we calculate the mean final mass. We can now measure the strength of correlation between final mass and each feature *at fixed* subhalo radius. We do that by calculating the range of mean final mass values across all bins of the secondary feature, while staying in the same bin of subhalo radius. Finally, we calculate the mean such range, averaging over all the bins of subhalo radius. We adopt the secondary feature with the largest mean range as the second most important feature. This algorithm selects the scale factor of the halo-subhalo interaction as the second most important feature for predicting the final subhalo mass. We continue with this process until we have extracted four features; the sample size does not permit further binning of the data beyond four dimensions without having too few bins to be useful.

The four most important features for each of the predicted quantities are shown in Table 2.2. The numbers in parentheses represent the strength of correlation between the

Table 2.2: The ranking order of most important features for predicting each of the desired quantities. The second column displays the normalized maximum range in target quantity that results from binning on that feature, where higher values (maximum 1) indicate the feature is more strongly responsible for changes in the prediction outcome. The final row shows mean and standard deviation in the normalized maximum range at the fourth bin level, due to a random parameter.

rank	Survival	Mass Loss	Position	Merge Time
1st	a_{acc} (.997)	R_{sub} (.243)	a_{acc} (.538)	a_{acc} (.283)
2nd	q (.254)	a_{acc} (.132)	q (.214)	q (.127)
3rd	ϕ (.145)	ϕ (.047)	ϕ (.177)	ϕ (.091)
4th	v_{rel} (.093)	M_{host} (.042)	v_{rel} (.153)	v_{rel} (.058)
random	0.027±0.002	0.012±0.0004	0.035±.009	0.017±0.002

predicted quantity and each selected feature. This is determined by using the ranges, as described above, that were maximized to select the most important features. For ease of comparison, we normalize these ranges by the full range of the predicted quantity in the data. For example, in the case of predicting the final position of the subhalo, the range of mean final positions in bins of scale factor is 53.8% of the total range of final positions. Furthermore, the mean range of final positions in bins of mass ratio, at fixed scale factor, is 21.4% of the total range of final positions. The mean range of final positions in bins of impact angle, at fixed scale factor *and* mass ratio, is 17.7% of the total range of final positions, and so forth. We note that in the case of mass loss, we convert to log space before reporting the normalized ranges.

From Table 2.2, we see that the same features appear repeatedly for all of our quantities. In particular, survival, final position, and merge time have the same four features, in the same relative order, chosen as most important for making their predictions. The initial scale factor of entry, ranked as most important for all of those quantities, is also ranked as second most important for predicting mass loss. The impact angle of the subhalos orbit also appears as important for predicting mass loss. These results are already encouraging, as many of these features are those that we were motivated to select because they were known to be important from previous works, as discussed in Section 3.1. It is interesting that, although the eccentricity of a subhalo has been shown to affect subhalo evolution in a number of ways (Taylor and Babul, 2004; Boylan-Kolchin et al., 2008; Jiang et al., 2008; Klimentowski et al., 2010; McCavana et al., 2012), the impact angle feature is chosen over the eccentricity, meaning that just the initial direction that the subhalo enters the host is more influential than the actual orbit that the subhalo is on. While these two parameters are fairly well correlated and contain similar information, there is a large amount of scatter in their relationship.

In the last row in Table 2.2, we include the mean and standard deviation of the normalized ranges calculated from a set of uniformly distributed random numbers at the 4th ranking level over 100 runs. Comparing the ranges due to our features to these random ranges, we can tell that all four of our chosen features for all of our predicted quantities hold some information more than noise. Furthermore, from the relative ranges associated with these top four features, we can see that some of our predicted quantities may need more than four features to make accurate predictions, while others may be able to reach maximum accuracy with fewer features. For example, in the case of mass loss, the range due to the fourth feature, $\mathrm{M}_{\mathrm{host}}$, is already small, and closer to the range obtained by a uniform random number than in the case of, for instance, final position, where the range of the 4th feature remains relatively high, and well above noise.

Figure 2.3 shows predicted quantities as a function of the two most important features. For example, the top left panel shows the mean survival fraction as a function of both initial scale and subhalo to host mass ratio. Note the strong trends in predicted quantities in these 'best feature planes'. For survival, the plane is clearly divided, suggesting that the survival of a subhalo is already well-determined by only two features. For the other quantities, this gradient is less defined, suggesting that either more features are needed to make good predictions, or the process is stochastic enough that general trends with respect to our input features are harder to find. In particular, the panel for merge time (lower right) in Figure 2.3 shows little variation in outcome across the entire plane of initial scale and subhalo-to-host mass ratio, despite the merging time having the strongest trend of variation with those

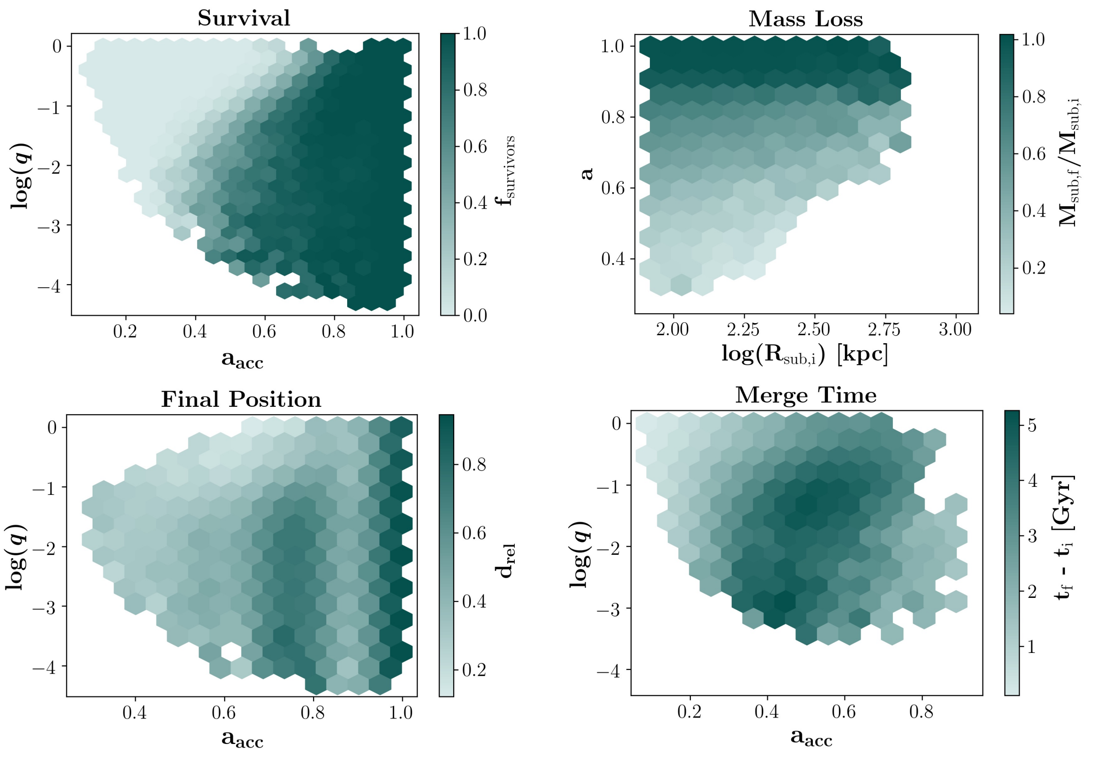


Figure 2.3: Distributions of the predicted quantities of interest with respect to the two parameters that are most responsible for each of their variations. In each panel, the parameter that causes the most variation is shown on the x-axis, and the parameter that causes the second most variation is shown on the y-axis. In addition, in each panel the colorbar shows the average value of the quantity of interest within a hexagonal bin. The top left panel shows the fraction of surviving subhalos. The top right panel shows the fraction of subhalo mass that remains for surviving subhalos. The bottom left panel shows the fractional distance of surviving subhalos from their host's center. The bottom right panel shows the elapsed time for a subhalo to merge. In each instance, some pattern of color striation can be seen to represent the importance of the two parameters shown. However, it is clear that the survival of a subhalo is by far the most drastically divided and well-defined by this two-dimensional space.

features. From these 'best feature planes', it is clear that some of these final quantities, such as the binary outcome of survival or disruption, will be much more straightforward to predict than others, such as merge time. The lower left panel of Figure 2.3 shows some structure in the final relative distance with regards to scale factor, which likely is due to the orbital period of subhalos. Subhalos that enter their hosts at, for instance, $a_{\rm acc} = 0.75$, are likely near the outer edge of their host halos because they have completed one full orbit around their host. Indeed, this corresponds to a time of around 3.5 Gyr, which is roughly the dynamical time of a host halo.

We have designed our feature selection algorithm to find the set of features that adds the most new information the fastest. However, the `scikit-learn` algorithms that we use also provide a `feature_importances_` method, which ranks features according to their relative placement in the decision trees, where features with higher rankings being able to split more of the data more effectively. We find this ranking to be less useful than our feature importance ranking, as we are interested in finding the smallest set of features that can be used to make accurate predictions rather than a complete set of features that holds relevant information.

2.4.2 Survival

For the entire sample of subhalos, we predict whether or not a halo will survive until z=0 (assigned a 1) or dissolve within the host (assigned a 0) using a random forest classifier. The details of this model are outlined in Section 2.3.3. Our accuracy is defined straightforwardly as the percentage of halos correctly classified. Figure 2.4 displays the accuracy when one feature at a time is added to the model in the order shown. We emphasize that, although we add features to the model in an ordered way, the random forest does not use the ordering of input features when training a model; each new subset of features requires re-training the model. Thus, if the ordering of our features had been completely random, the maximum model accuracy would not change, although the slope of information gain in Figure 2.4

would.

As is clearly shown in Figure 2.4, after adding the four most important features, we reach a maximum accuracy for both the testing and training sets of 94.4% and 94.6%, respectively. The gap between these accuracies is small, suggesting low overfitting. The decrease in accuracy (less than 1%) when adding additional features beyond these four suggests a lack of information gained by adding any feature thereafter. Any small increase or decrease in the accuracy beyond the first four features are within noise. Since the model is free at any iteration to use as many of the provided features as needed, the fact that maximum accuracy is reached after the addition of only these four features suggests that they are the only features that are necessary to predict the survival of a subhalo.

The four critical features are: $a_{\rm acc}$, q, ϕ, and $v_{\rm rel}$. The initial scale of the subhalo entry is overwhelmingly the most important of these features - 90% of our test sample is accurately predicted with this feature alone. With the addition of each of the remaining features, a 2.7%, 1.2%, and .27% gain in accuracy occurs. In Figure 2.5, we show trends in accuracy with respect to each of these features, by showing the average accuracy score in bins, normalized by the average accuracy score of the entire sample. Here, we see distinct trends in accuracy with respect to each of our features. The initial scale of entry, which has the most predictive power for our sample, also has the most drastic trend with accuracy. All subhalos that enter their hosts at times either before $a_{\rm acc}$ = 0.3 or after $a_{\rm acc}$ = 0.9 are predicted correctly. This is consistent with what we see in the top left panel of Figure 2.3, where all subhalos entering after $a_{\rm acc}$ = 0.9 survive, and almost all halos entering before $a_{\rm acc}$ = 0.3 merge. Subhalos with entry times between $a_{\rm acc}$ = 0.5-0.7 are by far the hardest to make predictions for, with the peak of this uncertainty occurring at $a_{\rm acc}$ = 0.6. There is also a strong trend with respect to q, where in general the larger the subhalo-to-host mass ratio is, the better survival is predicted. The one exception to this trend is at the smallest subhalo-to-host mass ratios, with $q < 0.002$, where the percent of accurate predictions returns to about average. This is likely because most of the subhalos below this mass ratio do not survive, making them easier to predict. Subhalos on orbits with the highest ϕ (most grazing orbits) are worse predicted than those on orbits with $\phi < 0.7$, and prediction accuracy appears to generally decrease for orbits that are both more plunging and more grazing than ϕ = 0.3-0.4. A similar trend occurs in $\log(v_{\rm rel})$, where subhalos with both larger and smaller relative velocities being worse predicted than subhalos with $\log(v_{\rm rel})$ = 2.1-2.4. For both ϕ and $\log(v_{\rm rel})$, the difference between the best and worst average accuracy points are much smaller than for $a_{\rm acc}$ or q, so these trends are also less significant.

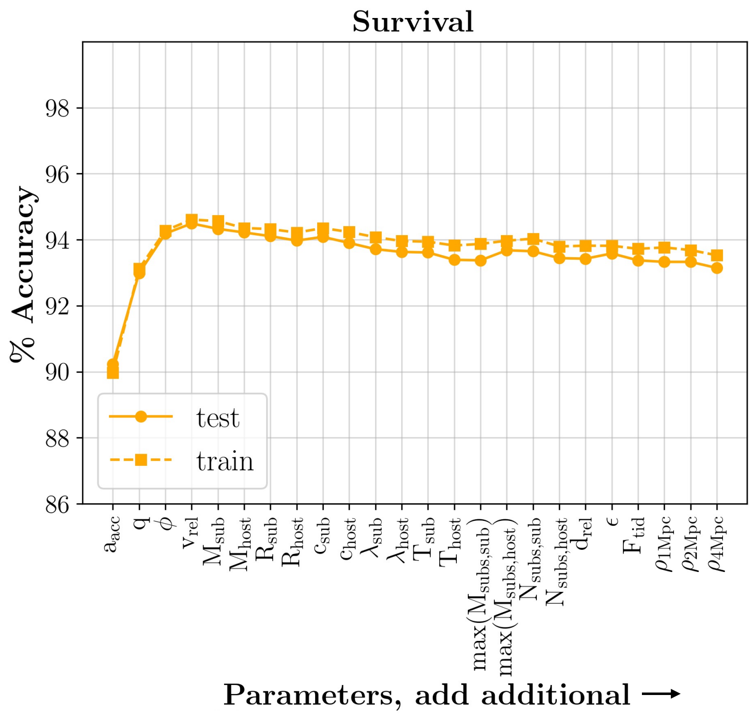


Figure 2.4: Accuracy of model predictions, for both the training and test sets, in the case of predicting subhalo survival. On the y-axis, we show the accuracy, defined as the percentage of the subhalo sample that is predicted correctly. On the x-axis, we show the features used to train the model. For each point, the model was trained using all features to the left of and including that point on the x-axis. The solid line and circles show the accuracy of the test set, while the dashed line and square points show the accuracy of the set the model was trained on. The choice and order of features for the first four features in the x-axis is determined by our feature selection algorithm described in Section 2.4.1, while the order of the remaining features is arbitrary.

In Figure 2.5, we also show the distributions of our correctly and incorrectly predicted subhalos for our model trained with the four most important features. There is a clear ten-

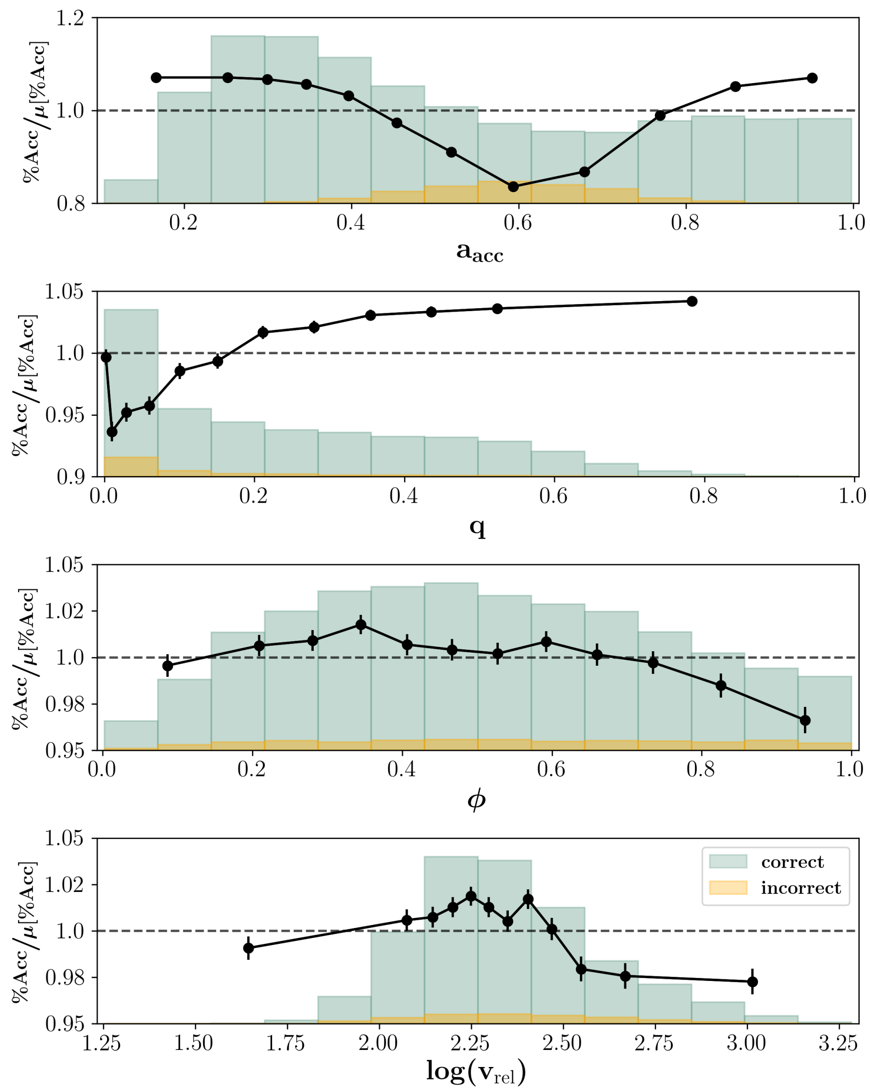


Figure 2.5: Accuracy of subhalo survival predictions for our test set, as a function of the four most important features, shown in four different panels. Black lines and points show the average percentage of accurate predictions within bins of each feature, normalized by the average percentage of accurate predictions of the entire test set. Bins along the x-axis are created such that the same number of subhalos belong to each bin. We include dashed lines where the y-axis value is 1. Above this line, predictions in that bin are on average better than the test set average, and below this line, predictions in the bin are on average worse than the test set average. The histograms show the distributions of the accurately (green) and inaccurately (orange) predicted populations. Since the survival quantity is binary, predictions can only either be correct (prediction matching truth) or incorrect (prediction not matching truth). We note that the histograms are normalized to the figure size and their height does not correspond to the y-axis labels.

dency for subhalos with a_{acc} around 0.5-0.7 to be the most difficult to predict, corresponding to a range of redshifts of around z = 0.68-0.55. The distribution of correctly predicted subhalos peaks near $a_{\mathrm{acc}} = 0.3$, or z = 2.3. As satellite occupation peaks at around z = 2.5 (Wetzel et al., 2009), this peak is likely due to most of the interactions occurring there. In fact, as the vast majority of our subhalos are correctly predicted, these distributions of the correctly predicted samples look nearly identical to the distributions of the complete sample. The peak of the incorrectly predicted subhalos also agrees with the region of greatest uncertainty that we see in the upper left plot of Figure 2.3, where a clear division between always surviving and always dissolving occurs at $a_{\mathrm{acc}} = 0.6$. We note that, although most of the poorly predicted halos exist in this small section of feature space, 80.3% of halos with a_{acc} = 0.5-0.65 are still accurately predicted, significantly better than random guessing. We further investigate this range of scale factor values by training a model with data exclusively in the a_{acc} = 0.5-0.7 range, to see if we can predict this population better when it is isolated. However, this model did not have better accuracy on the subhalos in this range than our model trained on all data. Moreover, the four most important features as selected by our feature selection algorithm are rearranged but not changed by this test. In the remaining panels of Figure 2.5, the distributions of the correctly and incorrectly predicted halos are roughly the same, and cover roughly the same range of values. In the bottom panel, the correctly predicted population peaks where the average accuracies are the highest, indicating a higher fraction of incorrect subhalos at low and high $\log(v_{\mathrm{rel}})$.

To determine if there are significant differences between the correctly and incorrectly predicted subhalos with respect to our additional features beyond these most important four, we perform a KS test between the distributions of these two populations. We do this test using the distributions of correctly and incorrectly subhalos, with respect to each of the additional features beyond our set of the most important four. To ensure that any differences we find in these distributions are not due to correlations with our four most important features, we take a slice of our data in a narrow four-dimensional bin, that fixes a range of values for all of these four features. We then ensure that within this selected bin, the distributions of the correct and incorrect populations are the same according to the KS test. Then, we perform an individual KS test between the two distributions from this slice of data, with respect to each of our additional features. In doing so, we find that the two distributions are found to be the same, with a p-value of above 3σ, for all of our additional features. This suggests that there is no significant difference in any of the additional features between correctly and incorrectly predicted subhalos.

2.4.3 Mass Loss

For all surviving subhalos at z=0, we predict the final mass using a gradient-boosting regressor, with hyperparameters as given in Table 2.1. To determine the accuracy of the model, we define an error metric, δ(M), which we call the prediction error of each individual prediction, using the the difference between true and predicted fractional remaining mass. A subhalo is considered to be accurately predicted if:

$$\delta(M) = \frac{|M_{\text{pred,f}} - M_{\text{true,f}}|}{M_{\text{true,i}}} - \frac{2m_{\text{p}}\sqrt{N_{\text{p,true,i}}}}{M_{\text{true,i}}} \leq tol \tag{2.7}$$

Where *tol* is some tolerance value which determines what difference in fractional mass loss is acceptable as accurate. *M* is the mass of the subhalo. N_{p} is the number of particles belonging to the subhalo, and m_{p} = 3.215 × 10^7h^{-1} M$_\odot$ is the mass of a dark matter particle in the simulation. Subscripts *pred* refer to a value predicted by the model, while *true* refer to the true value from the simulation. Subscripts *f* denote quantities taken at z=0, and *i* denote quantities taken when the subhalo first enters the host. We can then vary the tolerance, determining what percentage of subhalos have their prediction errors within certain tolerance thresholds. We point out that, because this prediction error is a measure of how close a prediction is to the truth, a higher prediction error value corresponds to a worse prediction, and a lower prediction error value means a better prediction.

The first term in this equation measures the difference between our true and predicted

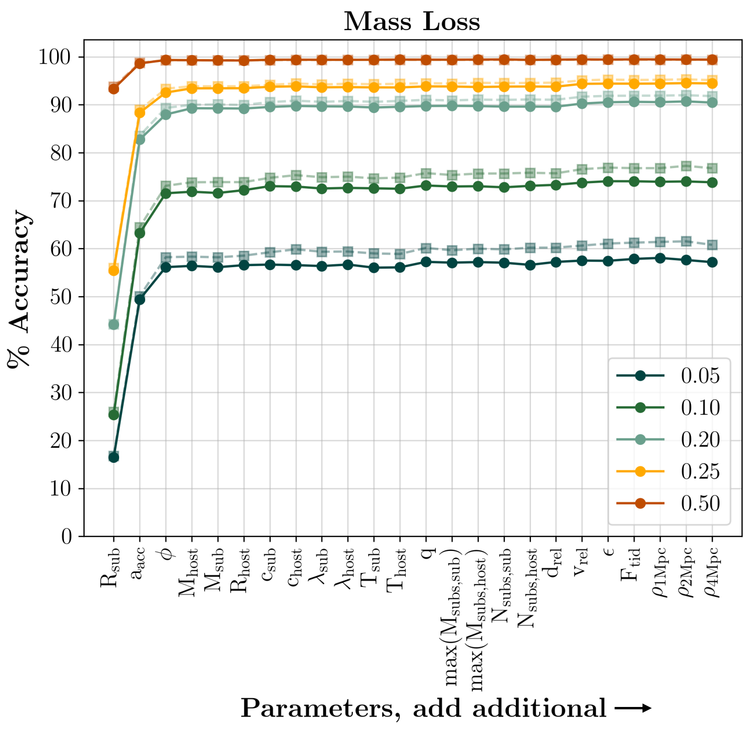


Figure 2.6: Accuracy of model predictions, for both the training and test sets, in the case of predicting subhalo mass loss. On the y-axis, we show the accuracy, defined as the percentage of the subhalo sample with a prediction error below some specified tolerance, as defined by Eq. 2.7. On the x-axis, we show the features used to train the model. For each point, the model was trained using all features to the left of and including that point on the x-axis. The solid lines and circles show the accuracy on the test set, while the dashed lines and square points show the accuracy on the set the model was trained on. Different colored lines show the different tolerance values used to define accuracy. For a complete description of this accuracy metric, see the associated text. The choice and order of features for the first four features in the x-axis is determined by our feature selection algorithm described in Section 2.4.1, while the order of the remaining features is arbitrary.

masses. Although the quantity that our machine learning model predicts is the mass of the subhalo, we determine prediction error by normalizing this value to the initial mass of the subhalo and comparing true and predicted fractions of initial mass. We do this in order to have a metric that equally penalizes errors in prediction for all subhalos, rather than allowing more leniency depending on the subhalo mass. The second term accounts for Poisson noise in the number of particles assigned to the subhalo. By subtracting this noise term from the error, we are stating that a prediction is perfect if it is within the Poisson noise limit. This term is significantly smaller than the first term and only makes a difference in the case of small subhalos where a small number of particles make up a large portion of the mass.

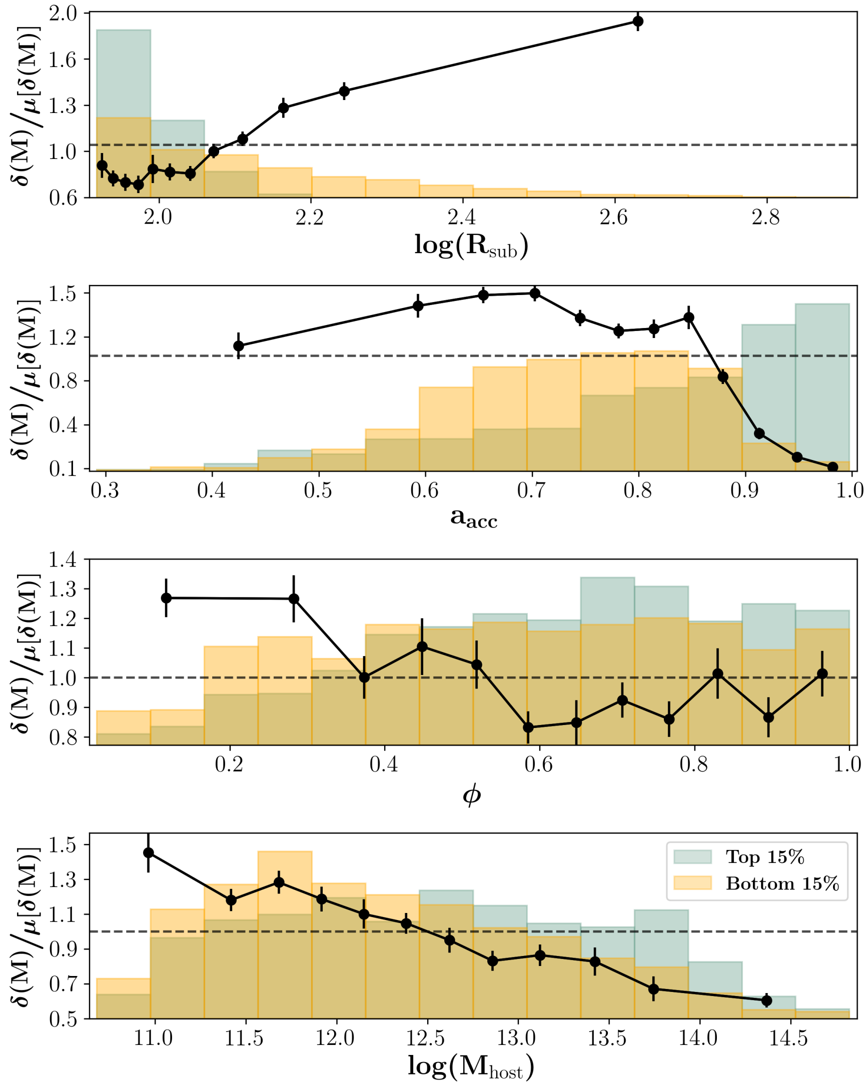


Figure 2.7: Prediction error for subhalo mass loss in our test set, as a function of the four most important features, shown in four different panels. Black lines and points show the average prediction error within bins of each feature, normalized by the average error of the entire test set. Bins along the x-axis are created such that the same number of subhalos belong to each bin. We include dashed lines where the y-axis value is 1. Above this line, errors are higher than average, so predictions in that bin are on average worse than the test set average. Below the dashed line, errors are lower than average and predictions in the bin are on average better than the test set average. The histograms show the distributions of the 15% best (lowest prediction errors; green) and 15% worst (highest prediction errors; orange) predicted populations. We note that the histograms are normalized to the figure size and their height does not correspond to the y-axis labels. The error metric that we use for predicting subhalo mass loss is given by Eq. 2.7 and described in detail in the associated text.

Figure 2.6 shows the accuracy of the model, when trained using the technique described above. Since the accuracy of this model depends on the selected tolerance value, we present our results for a range of tolerances. Again, the training set generally does better than the test set, for all tolerances, due to slight overfitting. It can be seen that, using only the four top-ranking features, 56.5% of subhalos have their final masses accurately predicted to with a margin of error of less than ±5% of their true initial mass. 89.3% of subhalos can be predicted accurately, given predictions within ±20% of their initial mass. Almost all (99.3%) subhalos can have their masses predicted to within ±50% of their initial mass, although it's worth noting that this tolerance encompasses a very wide range of mass loss.

To predict subhalo mass loss, the three most important features are: $\mathrm{R_{sub}}$, a_{acc}, and ϕ. Again, given the ordering from the feature selection method discussed previously, these features drive the steepest information gain, even given a model allowed to select any of the full 26 feature set. At the 20% tolerance level, adding these first three features results in an increase of 44.2%, 38.5%, and 5.3% accuracy percentage gain, respectively. We note that, because the radius and mass of subhalos are directly analytically related to one another, the radius can be replaced with the mass in this model with no difference in the information gain or final accuracy.

Figure 2.7 shows the average prediction error, as a function of the four most important features for making these predictions. We create bins with respect to each of these 4 parameters, spaced such that that the same number of subhalos belong to each bin, and plot the average prediction error in that bin, normalized by the average prediction error of the entire sample. We also show distributions of the subhalos with the 15% best and worst predicted mass loss. There are clear trends in the error of our predictions with respect to each of these four features. The prediction error increases with increasing $\mathrm{R_{sub}}$, so smaller subhalos are predicted better than larger subhalos. Prediction errors also decrease for subhalos that enter their hosts at more recent times. In particular, there is a sharp improvement to predictions for subhalos entering their hosts at $a_{\mathrm{acc}} \geq 0.85$. Subhalos entering their hosts around a_{acc}

= 0.6-0.7 are the hardest to make predictions for. There is a slight trend in prediction error with regards to ϕ. Subhalos that enter their hosts with $\phi \leq 0.3$ are slightly harder to make predictions for than those entering on more grazing (higher ϕ) orbits. Finally, there is a roughly linear trend between prediction error and log(M_{host}), with prediction error decreasing as log(M_{host}) increases, meaning that it is easier to make predictions for subhalos entering larger hosts. This is likely related to mass ratio as well, as smaller hosts will tend to have interactions with larger q than smaller hosts. Indeed, we find a similar trend in log(q), with better predictions for subhalos with smaller mass ratios. As with the survival prediction, we additionally try to train a model with data exclusively where a_{acc} = 0.6-0.8 to determine if it is possible to get better performance in this particularly challenging region. Again, we find that a model trained on only subhalos with a_{acc} = 0.6-0.8 does not perform better on data in this region than the model that was trained using subhalos with all a_{acc} values. In this case, three out of the four most important features as selected by our feature selection algorithm are rearranged but remain unchanged in this test.

Although a significant fraction of our subhalo population cannot have their final masses predicted with high precision, we point out that this model still performs much better than a naive guess. As a baseline model, we assign a predicted final mass fraction to all subhalos as the average final mass fraction of the sample. Using Equation 2.7 to calculate the accuracy of this baseline model, we find that only 15% of subhalos have their final masses accurately predicted to with a margin of error of less than ±5% of their true initial mass, 65% with a margin of error less than ±20% of their true initial mass, and 93% with a margin of error less than ±50% of their true initial mass. It is clear that, although the majority of subhalos can be predicted to within ±50% of their true initial mass by both models, our machine learning model can make more precise predictions than the baseline model.

Interestingly, the distributions of the best and worst predicted subhalo populations do not always separate strongly. Most of the well-predicted subhalos are of smaller size, while the poorly predicted halos span a larger range of sizes, although their highest concentration is at a similar size to that of the well-predicted subhalos. We note that this is likely due to the fact that there are more small subhalos than large, so the majority of our sample falls within this range. However, the distribution of poorly predicted subhalos does tell us that, despite the trend in prediction error with subhalo size, many of our smaller subhalos are still difficult to make predictions for. As the radius of a subhalo is analog to its mass, this also means that less massive subhalos are better predicted than their more massive counterparts. The well-predicted subhalos also tend to reside in larger host masses than their poorly predicted counterparts, suggesting that the well-predicted population is more comprised of unequal mass ratios. The best-predicted subhalos are also those that enter their host at later times, with the contours centering around a_{acc} = 0.9-0.95, likely because those do not have much time to lose mass before the end of the simulation, and thus have final masses similar to their initial masses. The most concentrated regions of poorly predicted subhalos also trace the regions of highest prediction error well. The best predicted subhalos appear to have slightly higher impact angles than their poorly predicted counterparts, although the total span is roughly the same for both.

As before, we want to determine if there are significant differences between the best and worst predicted subhalos with respect to our additional features. As we did with the correctly and incorrectly predicted populations for our survival predictions, we perform a KS test between these distributions. This is done with respect to each of the additional features beyond our set of the most important four, after finding a narrow bin within these four features that removes all differences between the best and worst predicted halos with respect to those four features. The KS test shows that the two distributions are found to be the same, with a p-value of above 3σ, for all of our additional features, meaning that no additional feature exhibits a trend with the goodness of our predictions.

2.4.4 Final Position

We next predict the final position of a surviving subhalo at z=0, relative to the center of the host, using a gradient-boosting regressor. To determine the accuracy of the model, we define a positional error metric, $\delta(d_{\mathrm{rel,f}})$, which we call the prediction error for each of our predictions, using the the difference between true and predicted fractional distance from host center. A subhalo is considered to be accurately predicted if:

$$\delta(d_{\mathrm{rel,f}}) = |d_{\mathrm{rel,f,true}} - d_{\mathrm{rel,f,pred}}| - \frac{2R_{\mathrm{soft}}}{R_{\mathrm{host,f}}} \leq tol \tag{2.8}$$

Where *tol* is some tolerance value that determines what difference in fractional distance from host center is acceptable as accurate. Here, $d_{\mathrm{rel,f}}$ is the distance between subhalo and host halo centers, normalized by the host radius. Subscripts *pred* refer to a value predicted by the model, and *true* refer to the true value from the simulation. $R\mathrm{host,f}$ is the radius of the host at z=0, and $\mathrm{R_{soft}}$ is the softening length of the simulation. The first term in this equation is simply the absolute difference between the true and predicted fractional distance from host center. Since what our model predicts is the actual fractional distance of the subhalo from the host halo center, we do not need to additionally normalize this quantity as we did when predicting mass loss, as this value can be straightforwardly taken as the fraction of the host radius by which the prediction is off. The second term in the equation is an additional tolerance, to account for uncertainty in the subhalo's position within its host due to the force resolution of the simulation. As with the mass loss predictions, we vary the tolerance to determine what percentage of subhalos have their prediction errors within that tolerance, which is how we define the model accuracy.

Figure 2.8 shows the accuracy of the model. For final subhalo position, it appears that more features are needed to reach maximum accuracy, with the first six required before accuracy converges. Moreover, the fraction of well-predicted halos is smaller than in the case of predicting mass loss. 39% of subhalos have their final positions accurately predicted

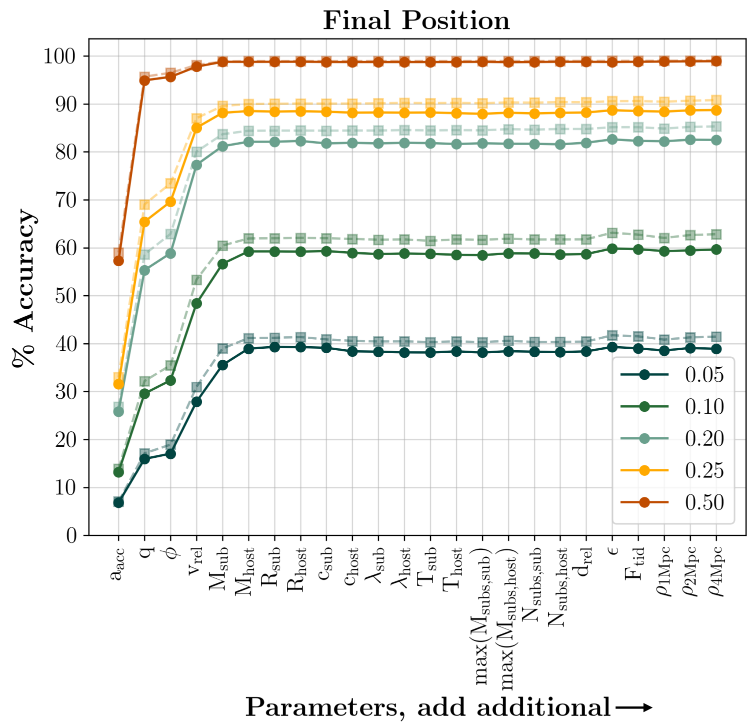


Figure 2.8: Same as Fig. 2.6, but for the case of predicting the final subhalo position. Different colored lines show the different tolerance values used to define accuracy, as defined in Eq. 2.8. For a complete description of the positional error metric we use to calculate this accuracy, see the associated text. The choice and order of features for the first four features in the x-axis is determined by our feature selection algorithm described in Section 2.4.1, while the next two features were chosen by the GBR algorithm. The order of the remaining features is arbitrary.

to ±5% of their final host radius, 82.1% of subhalos can be predicted accurately to within ±20% of their host's radius, and 98.8% to within ±50%.

The most important six features to predict final position are: the initial scale factor, the mass ratio between the sub and host halo, the subhalo's orbital impact angle, the relative velocity with which the subhalo enters, the mass of the subhalo, and the mass of the host halo. Given the ordering from the feature selection method discussed previously, it appears that these first four features were chosen to be quite important, although the subhalo impact angle provides less accuracy gain than some of the other, later-chosen features. The additional two features that we did not find with our feature selection methods, the mass of the subhalo and the mass of the host halo, were found by the machine learning model to be additionally important. Adding these first six features results in an increase of 25.8%, 29.6%, 3.6%, 18.3%, 3.9%, and 1% accuracy percentage gain, respectively, at the 0.2 tolerance level. In this case, our feature selection method does not add information in the optimal order, so some later added features provide more information gain than earlier selected features. For each of our four top features, we show trends in prediction error in Figure 2.9. There is a strong trend with regards to $a_{\rm acc}$, where the later a subhalo enters its host, the better its final position can be predicted. As with mass loss, this is likely because subhalos entering their hosts closest to z=0 have less time to undergo significant changes from the influence of their host, or fall very deeply into the host center. There is a significant decrease in prediction error for subhalos entering later than $a_{\rm acc} = 0.7$, with subhalos entering prior to that time being generally predicted poorly. Subhalos with both lower mass ratios, $\log(q) <$ -3, and higher mass ratios, $\log(q) >$ -1, are predicted better than those at more mid-range mass ratios, with prediction error notably decreasing for the higher mass ratios. There appears to be little trend in prediction error with regards to ϕ. Higher impact angles ($\phi > 0.8$) are predicted slightly worse than subhalos on more plunging orbits, but below this impact angle, there is no significant trend. Similarly, subhalos with $\log(v_{\rm rel}) < 2.2$ are predicted better than subhalos with larger initial velocities, but above this initial velocity there appears to be little trend. This is perhaps due to subhalos incoming with the smallest initial velocities not changing position significantly from their time of entry, making them easier to make predictions for.

We again compare the accuracy of these predictions to a baseline model, where we assign the average fractional distance from the host center of all subhalos to every subhalo in the sample. Using Equation 2.8 to calculate the accuracy of this baseline model, we find that 13% of subhalos are correctly predicted when using a tolerance of 0.05, 47% of subhalos are correctly predicted given a tolerance of 0.2, and 99% of subhalos are correctly predicted using a tolerance of 0.5. Given that the average final fractional distance for subhalos is around 0.55, achieving 99% correct predictions at the 0.5 tolerance level is unsurprising, as this high tolerance allows almost any subhalo within the host's virial radius to be considered correctly predicted. Comparing these baseline accuracies at the 0.05 and 0.2 tolerance levels to our machine learning model, which achieved 39% and 82.1% accuracy, respectively, we see that our machine learning model offers significant improvement over this simpler model.

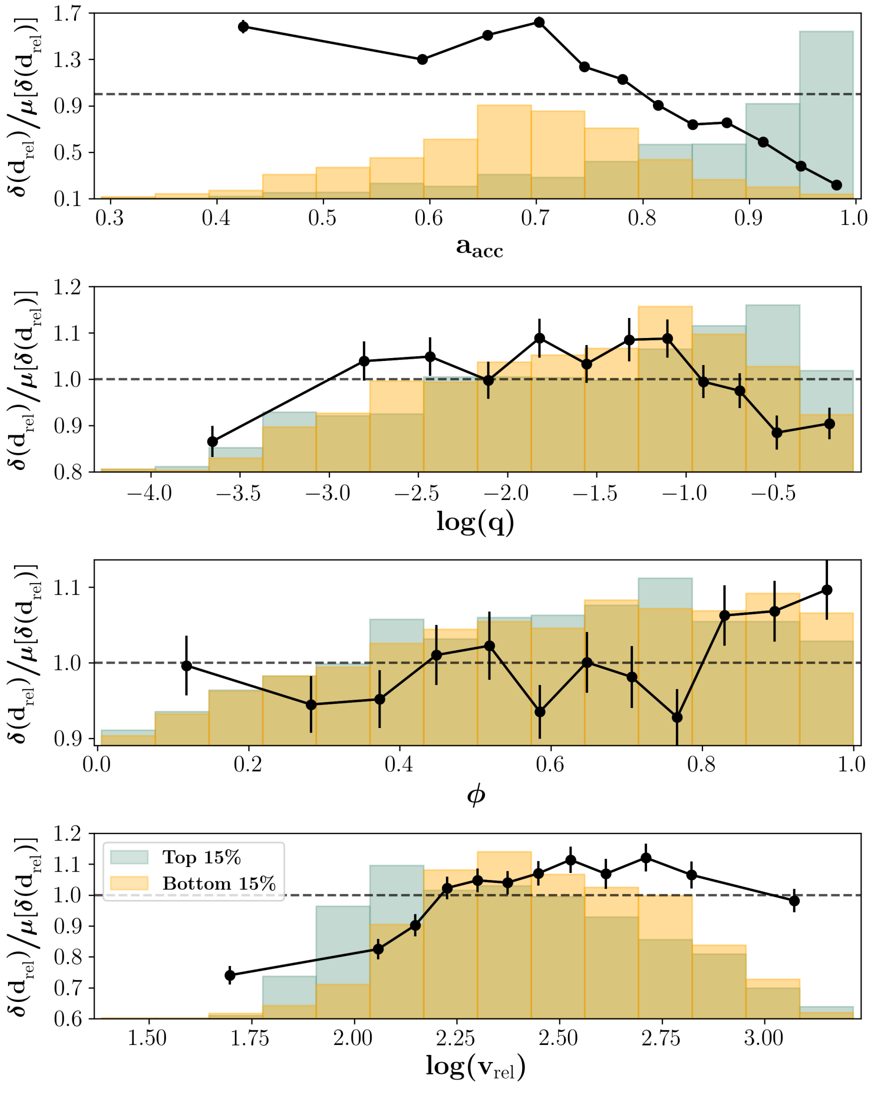


Figure 2.9: Same as Fig. 2.7, but for the case of predicting the final subhalo position. The prediction error, shown on the y-axis, is given by Eq. 2.8 and described in detail in the associated text. A higher error value corresponds to worse predictions in that bin, and a lower error value corresponds to better predictions in that bin.

Figure 2.9 also shows distributions of where the best and worst 15% of predicted subhalos lie. Most of the best-predicted subhalos are those that enter their host closer to z=0, likely because those have less time to move deep into the host and have their orbits altered. The distribution of poorly predicted subhalos peaks at an earlier time, around $a_{\rm acc}$ = 0.7, as those subhalos likely spend more time in their host halos with the potential for larger, less predictable perturbations. Well-predicted subhalos also seem to slightly favor more equal mass ratios than poorly-predicted ones, but the difference between the distributions is fairly minor, except for at log(q) $>$ -1.0, or mass ratios of greater than 1:10, where the higher relative number of well-predicted subhalos to poorly predicted subhalos brings the average prediction error down. From the third panel of Figure 2.9, it appears that best and worst distributions with the impact angle are quite similar. There are slightly fewer poorly predicted subhalos than well predicted subhalos at more grazing orbits with $\phi > 0.8$, which likely explains the increase in average prediction error at those values. Well-predicted subhalos favor slightly lower initial velocities than their poorly-predicted counterparts, with the peak of the well-predicted population occurring at log($v_{\rm rel}$) = 2.1, and the peak of the poorly-predicted population being closer to log($v_{\rm rel}$) = 2.3.

As before, we check the distributions of the best and worst predicted subhalos with respect to each of our additional features, beyond our set of the most important four, using a KS test. After controlling for $a_{\rm acc}$, ϕ, q, and v_{rel} by selecting a narrow bin in this feature space where the two distributions of best and worst predicted subhalos are the same, we perform a KS test between the two distributions with respect to all additional features. In doing so, we find that the two distributions are the same, with a p-value of above 3σ, for all of our additional features, except for the concentration of the subhalo and the eccentricity.

2.4.5 Merge Time

For those subhalos that dissolve before z=0, we predict the time between the subhalos entry and subsequent merging, using a gradient boosting regressor. To determine the accuracy of the model, we define an merger time error metric, δ(t), which again we refer to as the prediction error for an individual subhalo, using the the difference between true and predicted number of crossing times. A subhalo is considered to be accurately predicted if:

$$\delta(t) = \frac{|t_{\text{true}} - t_{\text{pred}}|}{t_{\text{cross,true}}} \leq tol \tag{2.9}$$

Where *tol* is some tolerance value which determines to within how many crossing times a prediction is considered to be accurate. Here, t is the predicted duration of the merger, and t_{cross} is the crossing time of the host halo, at the time that the subhalo dissolves, both in years. Subscripts *pred* refer to a value predicted by the model, and *true* refer to the true value from the simulation. Then, our tolerance is in units of final crossing times of the host halo. This δ(t) is calculated as the difference between the true and predicted elapsed time of infall for the subhalo, normalized by its final crossing time. Normalizing by this crossing time allows us to use this error metric for all subhalos, regardless of when the interaction occurs. As before, by varying the tolerance, we calculate the percentage of halos with acceptable prediction errors to get accuracy.

Figure 2.10 shows the accuracy of the model, when trained using all and increasingly smaller subsets of the features. Since accuracy depends on the selected tolerance value, we show the accuracy given several different choices of tolerance. As always, the training set generally does better than the test set, for all tolerances, due to slight overfitting. To predict subhalo merging time, only three features appear to be necessary to reach maximum accuracy. 41.6% of subhalos can be predicted to within half of a crossing time, 83.5% of subhalos can be predicted to within 1.5 crossing times, and 97.4% can have their merging time predicted to within 3 crossing times. We note that 3 crossing times is typically a few billion years, and is around the average time it takes a subhalo to merge, so this threshold is very lenient.

As with our other predictions, we compare the results of our model to a baseline model that assigns a constant merging time to each subhalo equal the average number of crossing

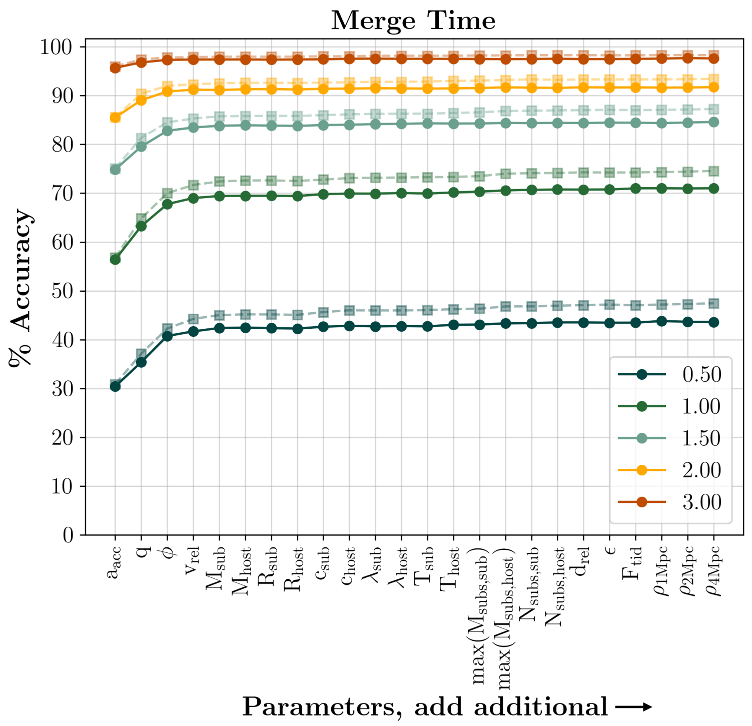


Figure 2.10: Same as Fig. 2.6, but for the case of predicting the subhalo merge time. Different colored lines show the different tolerance values used to define accuracy, as defined in Eq. 2.9. For a complete description of the error metric we use to calculate this accuracy, see the associated text.

times of all subhalos to merge. Then, we use Equation 2.9 to check the accuracy of this simple model and compare it to the accuracy of our machine learning model. We find that, using this baseline model, 40.3% of subhalos are correctly predicted to within .5 crossing times, 83% of subhalos are correctly predicted to within 1.5 crossing times, and 98.8% of subhalos are correctly predicted to within 3 crossing times. The performance of this simple model is extremely similar to that of our machine learning model, at all tolerance levels, suggesting that this model was unable to learn more complex merging behavior.

The three features needed before prediction accuracy levels off with the addition of more features are: the initial scale factor, the mass ratio between the sub and host halo, and the subhalo's orbital impact angle. Adding these first three features results in an increase of 74.9%, 4.7%, and 3.1% percentage gain in accuracy, respectively, at the 1.5 tolerance level. In Figure 2.11, we show trends in prediction error with respect to each of the top four features. In the top panel, we see a trend with $a_{\rm acc}$ for subhalos with entry times at $a_{\rm acc} > 0.45$, where later entry times are on average predicted better. For subhalos entering at times earlier than $a_{\rm acc} = 0.45$, there does not appear to be a trend. Subhalos with larger q are also better predicted than those at more unequal mass ratios, with the prediction error rapidly decreasing as mass ratios become more equal, until around $q = 0.3$, where the trend roughly levels off. This mass ratio of $q = 0.3$ is often presented as the threshold between major and minor mergers (Wetzel et al., 2009), so major mergers are predicted much more easily than minor mergers. This trend is likely due to the fact that major mergers happen more quickly than minor mergers do, and the more equal the mass ratio in a merger is, the more quickly the merger occurs. As such, we would expect higher q mergers to be easier to predict because the target value is smaller. Subhalos on more grazing initial orbits, with ϕ¿ 0.7 have on average higher prediction errors than those on more plunging orbits. Plunging orbits likely take less time to merge than grazing orbits, and thus likely also have less time for their orbits to be changed significantly, making them easier to predict. Initial relative velocity has a consistent trend with prediction error, where the higher $\log(v_{\rm rel})$ is, the worse a subhalo is predicted.

Figure 2.11 also shows the distributions the subhalos with the 15% best and worst prediction errors. The poorly predicted subhalos have a slight tendency to enter their hosts at earlier times than their better predicted counterparts, however the majority of both distributions are subhalos entering their hosts at earlier times. Most of the best predicted subhalos have more equal mass ratios, whereas the worst predicted subhalos are highly concentrated in the smallest q subhalos. The distribution of the best predicted subhalos peaks at more plunging ϕ orbits than the worst-predicted subhalos, with a higher concentration of poorly predicted subhalos at higher ϕ, following the trends that we saw with prediction error. Finally, the distributions in $\log(v_{\rm rel})$ look similar but offset, with the best-predicted subhalo distribution having a peak at around $\log(v_{\rm rel}) = 2.2$, and the worst-predicted subhalo distribution having a peak at around $\log(v_{\rm rel}) = 2.3$. This offset is small, but consistent with the trends in prediction error that we see.

We check the distributions of the best and worst predicted subhalos with respect to each of our additional features, beyond our set of the most important four. After controlling for $a_{\rm acc}$, ϕ, q, and v_{rel} by selecting a narrow feature space where the two distributions of best and worst predicted subhalos are the same, we perform a KS test between the two distributions with respect to all additional features. In doing so, we find that the two distributions are the same, with a p-value of above 3σ, for all features except for the spin of the host halo and the concentration of the host halo.

2.4.6 Subhalo Interactions

Several papers have noted that interactions between subhalos as they orbit within their hosts can be frequent and lead to significant amounts of mass loss, with as much as 40% of mass loss in a subhalo attributed to subhalo interactions (Tormen et al., 1998; Knebe et al., 2006; Klimentowski et al., 2010; Angulo et al., 2009). In our sample, we find that interactions between subhalos are quite common. Around 57% of our subhalos spend at least one snapshot as a sub-subhalo; that is, they enter the radius of another subhalo within the host at some point after they have entered the host itself. After becoming a sub-subhalo, around 23.5% of our total sample remain shrouded as sub-subhalos until they dissolve or until $z = 0$.

To test if these close interactions are important in determining mass loss, or any of our other predicted final quantities, we track and incorporate three additional subhalo features in our model: (1) a flag indicating whether the subhalo becomes a sub-subhalo; (2) a flag indicating whether the subhalo remains a sub-subhalo until either merging or $z=0$; and (3) the time that the subhalo spends being a sub-subhalo. Since the subhalo can enter and then exit another subhalo multiple times, this number may reflect time spent inside more than one subhalo. Although these features are tracked during the whole history of a subhalo's infall, and thus do not align with our initial goal of predicting outcomes of subhalos using

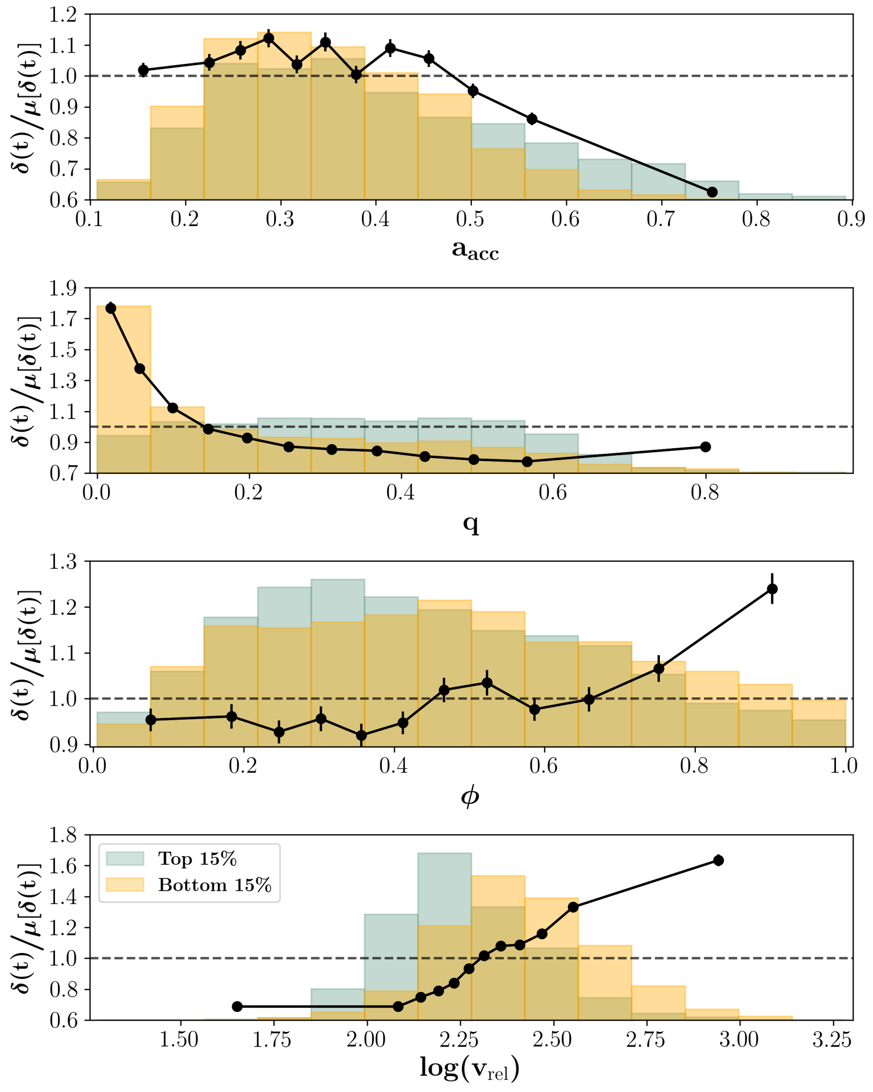


Figure 2.11: Same as Fig. 2.7, but for the case of predicting the subhalo merge time. The prediction error, shown on the y-axis, is given by Eq. 2.9 and described in detail in the associated text. A higher error value corresponds to worse predictions in that bin, and a lower error value corresponds to better predictions in that bin.

only initial conditions, we add these additional features to our model solely to determine if they matter significantly. We find that none of these features increase accuracy when added, meaning that the number and duration of interactions does not inform the evolution of our subhalo final quantities. We also tested whether these interaction features could add enough stochasticity to the merging process to be responsible for the difficulty in making these predictions by checking if the distributions of the best and worst predicted subhalos are significantly different. For each model, we control for the features that were found to be important by selecting a narrow bin within each of them, then we perform a KS test between the distributions of best and worst predicted subhalos with respect to our interaction features. In doing so, we find that the two distributions are the same, with a p-value of above 3σ, for all of our models. This suggests that these close interactions between subhalos are not important for their evolution as they fall into their hosts, neither by affecting the outcome nor by adding noise to the process.

2.5 Summary and Discussion

In this paper, we employed machine learning algorithms to predict the survival, mass loss, final position, and merge time of a subhalo from features taken at the time of its initial infall into its host halo. Our goal was to better understand to what degree these final outcomes are due to stochasticity in subhalo evolution versus real, physically-motivated processes that could be consistently, analytically predicted. we found:

- Subhalo survival vs. disruption (mass loss $> 90\%$) can be predicted remarkably well, with 94.4% of our sample being correctly predicted as surviving or disrupting. To reach this accuracy, four initial features are needed: the scale factor at the time of the start of the interaction, the mass ratio between the subhalo and its host, the impact angle of the subhalo's orbit, and the initial relative velocity between the subhalo and its host. However, to reach this accuracy, the initial scale factor is by far the most influential of these features, and an accuracy of 89.9% can be reached with this feature alone. Subhalos with both late and early entry times are easiest to predict, while those entering their host halos at $a_{\text{acc}} = 0.6$ are more difficult. However, this is also dependent on the subhalo-to-host mass ratio, where subhalos with lower mass ratios instead exhibit this transition closer to $a_{\text{acc}} = 0.3$. This is likely because lower mass ratio subhalos are in general more likely to survive, so a subhalo must enter its host at an earlier time to be subject to changes from its host for long enough to dissolve.

- Subhalo mass loss is a much more stochastic process. Although for 56.5% of our sample we were able to predict a final mass with an error within ±5% of the initial mass, we must loosen our criteria to ±20% of the initial mass in order to consider $\sim$ 90% of our sample correctly predicted. This maximum prediction accuracy is achieved using only three initial features: the radius of the subhalo, the scale factor at the time of the start of the interaction, and the impact angle of the subhalo orbit. In general, our model makes better predictions for smaller subhalos with late entry times than for those that are larger or have earlier infall times.

- Subhalo final positions are also difficult to predict. 39% of our sample can be correctly predicted to within ±5% of their host's initial radius, but an accuracy of 88.5% is only achieved when we loosen our error tolerance to within ±25% of the host radius. To make these predictions, six initial features are needed: the scale factor at the time of the start of the interaction, the mass ratio between the sub and host halo, the impact angle, the initial relative velocity, the subhalo mass, and the host halo mass. As with mass loss, our model makes better predictions for subhalos entering their hosts at later times. However, there does not seem to be a significant trend in prediction accuracy with regards to the other features that were found to be important for making these predictions.

- Subhalo merging timescales are also difficult to predict. 41.9% of our sample can be correctly predicted to within half of their host halo's final crossing time, but an accuracy of 91.1% is only achieved when we loosen our error tolerance to within 2 crossing times. Our model needs four features to make its predictions: the scale factor at the time of the start of the interaction, the mass ratio between the sub and host halo, the impact angle, and the initial relative velocity. Notably, this model shows no improvement in accuracy over a naive model which assigns to all subhalos the average number of crossing times our subhalos take to merge.

- There are some interesting commonalities among both the sets of features needed to make these predictions and the feature spaces in which predictions are poorest. Only five features, in total, are needed to achieve the maximum prediction accuracy for all of our predicted outcomes:. The scale factor, impact angle, relative velocity, and the masses of the host and subhalo (sometimes combined as mass ratio or appearing as virial radius instead) seem to be the only relevant features for determining subhalo evolution. Additionally, the feature spaces that are most difficult to make predictions within also have much overlap. In general, subhalos that enter at a mid-range of initial scales (typically $a_{\rm acc}$ = 0.6-0.7) are challenging to make predictions for, across all of our final outcomes. There also appear to be trends in impact angle, with higher impact angles (more grazing orbits) being easier to predict the behavior of, except for in the case of predicting disruption, where the opposite appears to be true.

- Additional features beyond the set needed to make predictions for each final quantity are not useful, either for making predictions or for characterizing the types of subhalos that are better or worse predicted. Although the best and worst predicted subhalos are typically distributed differently with respect to the features that are used to make predictions, when these features are controlled for, differences in these distributions with respect to all other features are removed. So, additional features outside of the set used to make predictions do not correlate with the stochasticity of our predictions.

It is clear from our results that, for predicting the mass loss, final location, and merging timescales of individual subhalos, an accurate, consistent mapping for a significant fraction of the population cannot be found given our set of initial features. There are several possible reasons for this inability to accurately model subhalo evolution. The first possibility is that some feature or features were missing from the initial set, which would have been fundamental to making accurate predictions. Although we have made sure to include an extensive list of physically-motivated features that were found in the literature to be important for modeling these outcomes, our list was not completely comprehensive. For instance, the halo finder ROCKSTAR outputs 75 features to describe each subhalo, many of which encode information about the ID's of the halos, but also include: halfmass radius, largest shape ellipsoid axes, angular momenta, velocity dispersion, and some others, most of which we decided not to include in our analysis. However, we have no compelling reason to believe that these excluded features would contribute such a meaningful portion of the needed information to bridge this gap in predictability. So, although the possibility remains that additional features could be needed to improve predictions, it seems unlikely that the missing information could be completely encompassed there.

A second possibility is that errors within the simulation, halo catalog, or merger tree make subhalo evolution unpredictable and sometimes incorrect. Much speculation remains as to the accuracy of N-body simulations and their ability to accurately model the physics of subhalo evolution, particularly on these small scales (van Kampen, 2000; Taylor and Babul, 2005; van den Bosch et al., 2018; van den Bosch and Ogiya, 2018). Although several studies have suggested that simulation resolution only effects subhalos with small numbers of particles (e.g., Gao et al., 2004; Nurmi et al., 2006; Diemand et al., 2007), van den Bosch et al. (2018) found that typical state-of-the-art cosmological simulations cannot resolve subhalos well enough to follow their mass loss until complete disruption. In the Bolshoi simulation, for example, van den Bosch (2017) found that only around 20% of subhalo disruption was truly physical, with instantaneous subhalo masses being highly erratic along the orbit. Similarly, van den Bosch et al. (2018) found that most subhalo disruption in modern simulations is artificial or numerical in nature, with only subhalos of exquisite resolution and greater than 10^6 particles per halo showing consistently converged results, especially for those orbiting close to the center of the host. A number of other works have also called into question the reliability of the halo catalogs and merger trees that are generated from these simulations. Comparison projects have found differing results for the fates of subhalos and the subhalo mass functions resulting from different halo finders (Knebe et al., 2011; Onions et al., 2012; Avila et al., 2014; van den Bosch and Jiang, 2016; Behroozi et al., 2015) and merger tree codes (Tweed et al., 2009; Srisawat et al., 2013; Jiang and van den Bosch, 2014). Because these codes fundamentally define subhalos in different ways and trace their properties between snapshots using different methods, these comparison projects found that, when applied to the same simulation, resulting halo catalogs and merger trees could differ quite significantly. This problem may be additionally exacerbated by the frequently tumultuous merger histories of halos (Sinha and Holley-Bockelmann, 2012).

Despite the fact that the veracity of simulation results has been brought into question, we note that it seems unlikely that simulation errors could be entirely responsible for the results we have found. van den Bosch and Ogiya (2018) found that, despite physical disruption[2] being extremely rare within simulations, subhalos only became highly sensitive to numerical disruption after losing over about 90% of their mass. Additionally, Avila et al. (2014) and Srisawat et al. (2013) found that spurious fluctuations in the masses of subhalos within simulations using ROCKSTAR may be frequent, and that subhalos that pass close to the centers of their hosts may have truncated merger trees. However, they also found that Consistent-Trees is usually able to successfully follow the evolution of subhalos, leading to overall reliable mass loss histories. Since our study only relies on final outcomes mapping from the initial conditions, any minor errors along the history should not be important. And, although different halo finders and merger tree codes can differ from one another, their behavior is generally consistent, and thus again should not be responsible for the inconsistent merging behavior that we have found.

Another possibility is that final outcomes can be accurately predicted from initial conditions of the interaction, but the machine learning methods used here were not able to

[2]We mean ”physical” here as it is used in van den Bosch and Ogiya (2018), where it refers to disruption caused by the tidal heating and stripping that unbounds the subhalo particles, as opposed to numerical disruption, which is due to a subhalo falling below the resolution limit of the simulation. In the case of numerical disruption, the subhalo would still exist if the resolution of the simulation were higher.

capture the process. This could be due to a few reasons. It is possible that the particular machine learning method used was not well-suited to the problem. However, we sampled different algorithms, from very simple methods to more complicated methods such as neural networks, and have found that this technique yields the highest fraction of accurate predictions. Another possibility is that we did not have enough data, or the data did not span the parameter space well enough. In this instance, noise in the data could overwhelm the relationship between input and output. We also tested this by repeating our process with smaller random subsets of our data, using instead one half and one quarter of our original sample, to ensure that our results did not change with fewer data points. In doing this, we found no change in maximum accuracy, even with a significantly reduced amount of data, suggesting that the noise in the data is not due only to sample size.

The final possibility is that there is an inherent chaotic nature of these interactions within N-body simulations. We find a large scatter in outcomes to be present in our data - even in narrow bins of the input parameters, there can be large differences in the outcome with regard to all of our predicted quantities, meaning that a consistent relationship between these inputs and outputs can not be found. As we do not expect the previous possible explanations to completely explain this behavior, we believe that a chaotic nature of these interactions is the most likely explanation. In this case, the initial conditions of an interaction are not enough to know the outcome, because similar initial conditions can lead to very different outcomes. This makes it impossible for a model that relies on similar initial conditions producing similar outcomes to get consistent results.

An inherent stochasticity in these merging processes would have implications for simulations and the models that are built from them. For instance, our findings in this work would suggest that analytic models that attempt to model the individual evolution of subhalos are doomed to be unable to describe all subhalo-host halo interactions, as the basic premise that the same inputs would yield the same outputs is not necessarily true. However, the regions of highest uncertainty in our predictions may give insights on how to improve these models. For instance, in predicting merge time, our model clearly had more trouble making predictions for minor mergers over major mergers. This could point to a need to model the merging times of these two populations separately, as our machine learning model, which is able to perform well in the major merger regime, clearly does not apply as well to the minor merger regime. In this instance, there may be an inconsistent dependence of merging timescales on the mass ratio, making it difficult to accurately parameterize its effect.

Despite the inability of these models to make accurate predictions at the individual subhalo level, we may still be able to use a model like this to construct subhalo populations. For instance, to determine the surviving population of subhalos at z=0 given a population of subhalos that enter the host, our model would be able to definitively determine the survival or disruption of some subhalos that enter within certain ranges of our most important features, and determine with some probability the survival or disruption of other subhalos that enter within the more uncertain ranges of our features. A model like this could properly take into account the regions of feature space where outcomes are more variable, to assign outcomes with some scatter, but model more precisely in the regions of feature space where outcomes are better defined. In this way, one could use our model to create realistic distributions of z=0 subhalo populations using moderate resolution simulations that do not actually resolve subhalo evolution.

The features that our model selects also give some interesting insights. For instance, predicting the survival, final position, and merge time of a subhalo, all required the same four features: $a_{\rm acc}$, q, ϕ, and $v_{\rm rel}$, all in the same relative order of importance, to make predictions. This likely means that the same fundamental physical processes are at work in determining all of these quantities. To predict mass loss of a subhalo, our model selected from the same broad subset of features, but the features that provided the most information to the model were not the same. This may mean that some additional processes effect subhalo mass loss, that skew some features to be more necessary in making predictions than were needed for the other final quantities. Another interesting result from the features that our model selected is that the impact angle and relative entry velocity of a subhalo as individual features appear to be more influential in determining subhalo evolution than a feature like eccentricity, which captures information about both. This may mean that allowing a model to individually weigh those two components of subhalo orbits leads to a more generalizable model of subhalo evolution than using eccentricity or circularity alone.

Finally, as our analysis here has shown us trends in the outcomes of subhalos with respect to certain features, we can use this information to draw conclusions about satellite populations. For instance, the overwhelming dependence of survival on the entry time of the subhalo, along with the clear division that we see in Figure 2.3 of survival fraction occurring at z = 0.67-0.43, suggests that these galaxies should be severely stripped of their dark matter, containing 10% or less of their original mass. Alternatively, due to the trend we also see in mass ratio, subhalos that enter at earlier times may retain more of their mass if they entered their host with a mass ratio of less than 1:100. Similarly, we could expect a roughly smooth trend in the fraction of halo mass remaining with the time of entry for subhalos.

In this study, we used a dark matter only simulation to study the evolution of subhalos. However, hydrodynamic simulations play a crucial role in our understanding of subhalo evolution by capturing a more complete context of baryonic physics. The next step in this type of work would naturally be to explore the implications of baryons. Because there are more potential factors dictating the evolution of subhalos in hydrodynamic simulations, such as feedback and enhanced tidal effects (Diemand and Moore, 2011; Brooks et al., 2013; Despali and Vegetti, 2017), we may expect that subhalo evolution in a hydrodynamic simulation is even more difficult to predict. A recent study by Nadler et al. (2018) perhaps confirms this, as they used dark matter properties to predict the survival of subhalos in a hydrodynamic simulation, with slightly less success than we were able to predict survival in a dark matter only simulation, meaning that the addition of baryonic physics makes this prediction more difficult. However, whether or not adding baryonic features to these types of models would improve predictions remains unexplored. Several works have studied the specific effects of baryons on the subhalos in which the galaxies reside (Dolag et al., 2009; Romano-Díaz et al., 2010; Brooks and Zolotov, 2014; Sawala et al., 2017; Garrison-Kimmel et al., 2017; Munshi et al., 2017; Richings et al., 2020), but as can be seen from this work, the chaotic nature of these interactions may make it difficult to quantify these trends into a machine learning model.

Finally, it would be interesting to explore the dependence of the stochasticity of these interactions on the place in the orbit at which the start of the interaction is defined. In this work, we have defined the start of the interaction as when the subhalo crosses the virial radius of the host halo. However, there are many other significant times in a subhalo's orbit which could be chosen instead. For instance, we could define the ”start” of the interaction as the time that a subhalo reaches its first pericenter, or as the time that it crosses the splashback radius of its host rather than the virial radius. These changes could significantly change our results, and an analysis of the performance of these models at various times during the infall could give valuable insight as to whether or not the stochasticity of the interaction increases or decreases after any point in the infall.

CHAPTER 3

Deep Learning the Formation of Structure Down to Smallish Scales

Cosmological simulations are one of the cornerstone tools in modern astrophysics. While these simulations prove incredibly useful over a wide range of applications, their computational expense remains a barrier to creating large ensembles of example universes, particularly of higher resolutions. Modern machine learning techniques offer a potential avenue to create simulated universes with significantly less computational expense. We utilize a U-Net deep convolutional neural network with a dark-matter only initial conditions density field evolved to z=0 using 2LPT as the network input. Then, we predict the z=0 density field from our full dark-matter only N-body simulation as the target output. We train this model with the mean-sqaured error loss function, then train an additional conditional generative adversarial network on the same data. Our input and output simulation boxes have 1 Mpc^3 voxel resolution. Our U-Net model generates a final density field that is accurate to 30% at k=3.0Mpc/h, and accurate to under a percent at the largest scales. The model predicts higher density regions more accurately than lower density regions and overall has a blurred effect. The generative adversarial network is accurate to 35% at k=2.0Mpc/h and to 15% at k=3.0Mpc/h. We discuss the limitations of the approach we used and some potential avenues to improve upon these results in the future.

3.1 Introduction

The cosmic web that we see today is the result of billions of years of evolution in our universe. From the time of the Big Bang until the present, density fluctuations in the universe's matter grow under the influence of gravity, eventually forming structures that host the galaxies and clusters we observe today. Understanding the formation of this structure in depth has been an ongoing topic of study in astrophysics, giving rise to multiple subfields, all of which aim to understand how our current universe forms from the relatively homogeneous primordial matter distribution.

These billions of years of evolution are often simulated using dark matter only cosmological simulations, which have become a fundamental tool for studying structure formation. From probing different cosmologies to understanding how galaxies are distributed, dark matter only simulations have been key in advancing our knowledge of the universe through their comparisons to observations (Springel et al., 2006; Angulo and White, 2010; Borgani and Kravtsov, 2011; Tinker et al., 2012). However, full cosmological N-body simulations, especially of high resolution, are incredibly computationally demanding to run, requiring potentially millions of CPU hours to complete (Boylan-Kolchin et al., 2009; Nelson et al., 2019; Maksimova et al., 2021). This computational expense can act as a barrier to running large suites of high resolution simulations, which are often necessary for properly understanding systematic errors in galaxy surveys and for generating reliable covariance matrices for our simulated universes (Berlind and Weinberg, 2002; Harnois-Déraps and Van Waerbeke, 2015; Klypin and Prada, 2018; Szewciw et al., 2022).

Machine learning, and deep learning in particular, have emerged in recent years as tools that can bypass more complicated computations by learning complex relationships between input and output data. Convolutional neural networks (CNN's) are a type of neural network that can learn the spatial relations between pixels in an image or voxels in a 3D volume (Krizhevsky et al., 2017; Tran et al., 2015; Gu et al., 2018). Since much data in astrophysics is either inherently image-like in nature or can naturally be treated as such, CNN's are an obvious choice for many astrophysical problems. In cosmology, CNN's have been used for a wide variety of tasks, including galaxy identification and categorization (Guo and Martini, 2019; Wu et al., 2019; Bretonnière et al., 2021; Maslej-Krešňáková et al., 2021; Domínguez Sánchez et al., 2022; Farrens et al., 2022; Tang et al., 2022; Zhang et al., 2022), photometric redshift estimation (Hoyle, 2016; Pasquet-Itam and Pasquet, 2018; Pasquet et al., 2019; Shuntov et al., 2020; Schuldt et al., 2021; Dey et al., 2021; Henghes et al., 2022; Lin et al., 2022), strong gravitational lensing identification (Jacobs et al., 2017; Lanusse et al., 2018; Petrillo et al., 2019; Cañameras et al., 2020; He et al., 2020; Huang et al., 2020; Savary et al., 2021; Li et al., 2021; Wilde et al., 2022), and identification and mass estimations of galaxy clusters (Ntampaka et al., 2019; Gupta and Reichardt, 2020; Kosiba et al., 2020; Yan et al., 2020; Su et al., 2020; de Andres et al., 2022; Lin et al., 2022; Hong et al., 2021; Ramanah et al., 2021).

Recently, CNN's have been explored for a number of large scale structure formation problems, by interpreting a density or displacement field of dark-matter only simulation boxes as a 3D volume and transforming it into a more useful fake simulation. Yip et al. (2019), Tröster et al. (2019), and Bernardini et al. (2022) all used CNN-like architectures to add baryons to dark-matter only simulations. Berger and Stein (2019) and Ramanah et al. (2019) directly predict halo masses and locations from low resolution dark matter density fields, and Bernardini et al. (2020) and Lucie-Smith et al. (2020) do so from an initial density field. Chardin et al. (2019) predict maps of reionization times from initial density and gas fields. Wu et al. (2021) reconstruct velocity fields from dark matter density fields.

There have also been several recent studies that aim to tackle the same problem that we do in this work - generating realistic final simulation outputs without running simulations - albeit with some different approaches. Kasmanoff et al. (2020), Ramanah et al. (2020), Li et al. (2020), Ni et al. (2021), and Schaurecker et al. (2021) make use of general adversarial networks (GANs) to super-resolve dark matter structures. These works use GANs to add resolution to lower resolution simulations, making simulation boxes that look like realistic higher-resolution counterparts of the lower-resolution boxes they are given. Rodríguez et al. (2018), Perraudin et al. (2019), Feder et al. (2020), and Ullmo and Aghanim (2021) also make use of GANs, instead generating completely new density fields which visually and statistically look like the output of a simulation run with a different set of random initial conditions. This allows for the fast generation of multiple simulation boxes that all have the same cosmology. Perraudin et al. (2021) created a more generalized GAN model that is able to generate simulated universes for a given set of cosmological parameters. Curtis et al. (2022) used a GAN to generate 2D images of the density field, with a focus on investigating void statistics in those generated images. Mustafa et al. (2019) and Tamosiunas et al. (2021) use GANs to produce weak lensing convergence maps, with Tamosiunas et al. (2021) also investigating generalizing their models to different cosmologies.

CNN's can also be used to approximate the results of a full cosmological N-body simulation, going from an initial density field to the final density field without running a simulation. He et al. (2019) trained a U-Net to learn a fast approximation of the final displacement field from an initial displacement field, generating simulations of 32^3 particles in a 128 (Mpc/h)^3 box and achieving few percent-level accuracy on the power spectrum at the smallest scales. de Oliveira et al. (2020) expanded on this approach and pushed it to finer resolution, instead training their U-Net model to predict final displacement fields of simulation boxes that are of resolution 512^3 particles in a 1000 (Mpc/h)^3 box. They were able to achieve the same percent-level accuracy at the smallest scales, despite the resolution of these simulations being more than 8 times higher.

In this work, we attempt to push to even higher resolutions, making use of a 1000 (Mpc/h)^3 simulation box with 2240^3 particles. Using a high resolution simulation allows us to predict a finer final density grid of resolution 1Mpc, pushing our predictions to include smaller structures. In this regime, the effects of gravity are highly nonlinear, so we expect that accurate predictions down to those scales may be significantly harder to achieve. However, predictions on scales that small would be necessary to eventually produce halo catalogs.

In Section 3.2, we give a complete description of the data. Like He et al. (2019) and de Oliveira et al. (2020), we use a U-Net architecture, which we give images of the initial density field to predict images of the final density field. The details of this architecture are discussed completely in Section 3.3. We go over the performance of this network, both visually and statistically, in Section 3.4. Finally, Section 3.5 includes a discussion of practical findings from this study as well as possible future avenues for this type of work.

3.2 Description of Data

This work makes use of a dark matter only (DMO) cosmological N-body simulation from the Large Suite of Dark Matter Simulations project (LasDamas; McBride et al. (2009)). This simulation was run on the Texas Advanced Computing Center's Stampede supercomputer using the public code GADGET-2 (Springel (2005)). The initial power spectrum was generated using `CMBFAST` (Seljak and Zaldarriaga, 1996; Zaldarriaga et al., 1998; Zaldarriaga and Seljak, 2000). Initial conditions are then generated using `2LPTIC` (Scoccimarro, 1998; Crocce et al., 2006), and evolved to a starting redshift of z=99. All simulations were run with the following cosmological parameters (Planck Collaboration et al., 2014): $\Omega_m = 0.302$, $\Omega_\Lambda = 0.698$, $\Omega_b = 0.048$, h = 0.681, $\sigma_8 = 0.828$, and $n_s = 0.96$. The simulation contains 2240^3 particles in a 1 Gpc/h box, and the mass of a single dark matter particle is 7.46 x $10^9 M_\odot$.

The particles of an N-body simulation are initially laid down on a grid with separation less than our desired 1Mpc grid for our density field. This makes using a gridded version of the initial density field impossible - gridding this already gridded particle distribution leads to an aliasing artifact that dominates the image. Instead, we use as the input density field the initial conditions run to z=0 using 2LPTic. As 2LPT approximates the evolution of structures, this also helps the U-Net not need to learn as large of a problem, as this density field has more of the large scale trends already in place.

However, running 2LPT to z=0 creates a different problem - some areas of the density field may be evolved too far, as 2LPT is unable to capture proper orbital trajectories of particles as they move into more highly dense regions. This shell crossing effect could have the opposite of the intended effect of helping the U-Net learn, so we attempt to mitigate this issue by dampening the initial power spectrum before generating initial conditions. This preserves the evolution that 2LPT predicts on large scales, but small-scale structures are less strongly evolved. We do this by dampening the initial power spectrum exponentially at the critical density according to:

$$P(k)' = P(k)e^{-k/2k_{nl}} \tag{3.1}$$

Where P(k)' is the power spectrum we use to generate these new initial conditions, given*k*and P(k) the output power spectrum of `CMBFAST`. The nonlinear scale, k_{nl}, is defined as the scale at which k_{cross} =*k*for:

$$k_{cross} = (\frac{2\pi^2}{P(k)})^{1/3} \tag{3.2}$$

Figure 3.1 shows the resulting input density field evolved to z=0 from dampening the input power spectrum. The left panel shows the result of evolving with 2LPT the initial conditions generated without a dampened power spectrum to z=0. The middle panel shows the result of evolving with 2LPT initial conditions generated from our dampened initial power spectrum to z=0. The final panel shows the full simulation results when run to z=0. All panels show a slice of one of our simulation boxes from the training set. To increase the clarity of structures in these images, the density values are plotted logarithmically. The bottom panel of Figure 3.1 shows the power spectrum for these three density fields. It can be seen both visually and in the power spectrum that dampening the initial power spectrum generates a result with 2LPT that is more similar to the full simulation box, especially at smaller scales. The undamped initial conditions run to z=0 have visually smoothed out clustering at the smaller scales due to the aforementioned shell crossing effect.

Convolutions must be performed on a grid of voxels, so we grid this density field using a 1.0Mpc grid, and a triangular filter of width 0.5Mpc. This triangular filter allows particles to contribute some of their count to a neighboring cell if the particle is close enough to the edge of its primary grid cell. This means that a particle that lies in the interior 0.5Mpc of a grid cell (±0.25Mpc from the center of the grid cell) will have all of its count given to

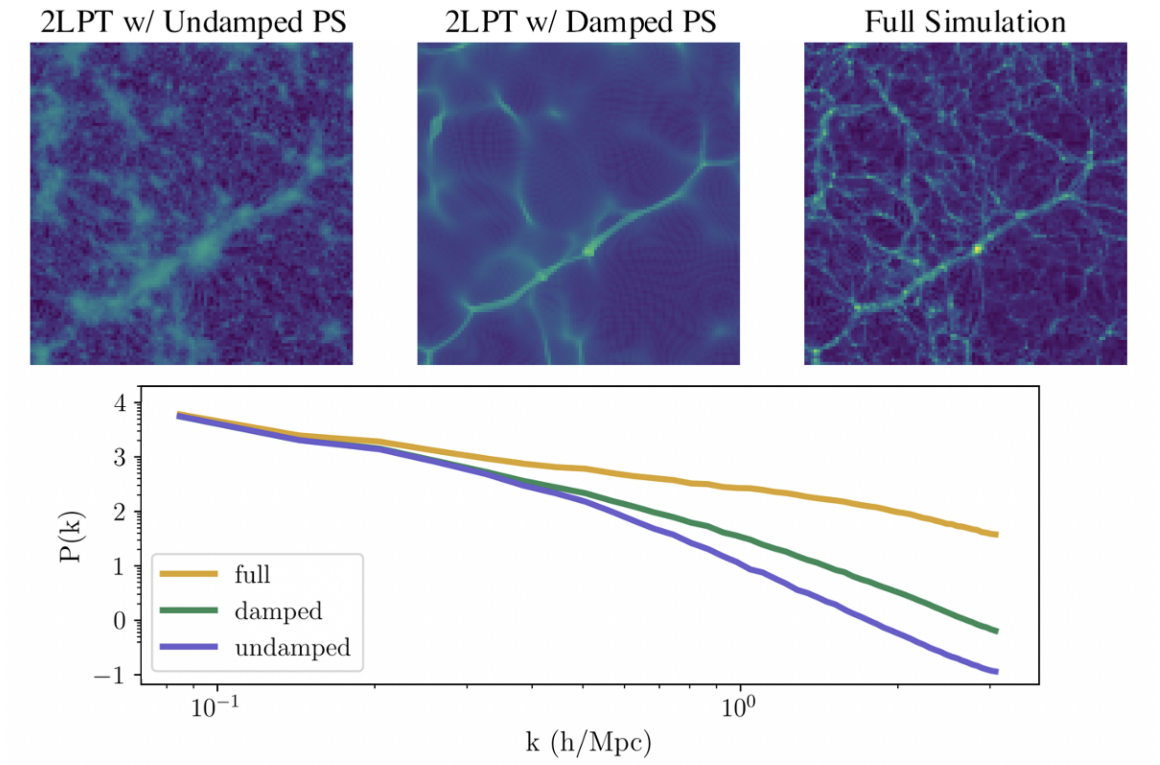


Figure 3.1: An example slice of the density field at z=0 for 2LPT without damping the initial power spectrum (left panel), for 2LPT with a damped initial power spectrum (middle panel), and for the full simulation (right panel). The bottom panel shows the power spectra of these three density fields.

that grid cell. However, a particle that is outside of these bounds will contribute a portion of its count to the neighboring grid cell that it is closer to. A particle that is exactly on the boundary of two grid cells will give half of its mass to each grid cell, a particle that is halfway between 0.25 Mpc from the cell center and the cell edge will give 75% of its count to the primary cell and 25% of its count to the neighbor. This contribution drops off linearly between 0.25Mpc from the grid center and the grid cell edge. This spread happens in all 3 dimensions, so a particle can maximally contribute density to 8 grid cells, with the contribution to each of these cells depending on the particle's x,y, and z offsets from center. While this gridding filter may spread out some features, it also prevents harsh boundaries between neighboring grid cells. We perform the same gridding on the final density field for consistent comparisons.

Large machine learning models have many learnable parameters, and require many calculations during a forward pass of the model. For this reason, there is not enough memory on the GPUs we train on to forward pass the entire 1000 (Mpc/h)^3 box. To fit our memory constraints, we cut this full simulation into smaller boxes of size 128 Mpc^3. This size box is small enough to fit in its entirety onto one GPU, while still being large enough to have structures at all relevant scales. This gives us 3000 smaller boxes for our training data. We also augment this dataset, by up/down flipping, left/right flipping, or 90-degree rotating each of these boxes along one, none, or two of the x,y, and z dimensions. This increases our effective dataset by a factor of sixteen, resulting in 48000 simulation boxes. We reserve a random subset of 64 boxes, before augmentation, for each the testing and validation sets. This results in 2872, 64, and 64 unaugmented boxes for training, testing, and validation, respectively.

To get an accurate prediction for an entire matching 128 Mpc^3 output box, the U-Net would need information outside of this input box, as the density surrounding voxels on the edge is fundamental to evolving those edge voxels properly. To avoid this issue, we simply ignore the predictions of the outer 16 voxels on all sides of our output image, as the sizes of our filters do not have information to predict them accurately. This is done by applying a final layer of the model which truncates these edges, reducing their size to (96,96,96) rather than (128,128,128).

3.3 Machine Learning Methods

In this work, we use a U-Net (Ronneberger et al., 2015) (commonly called V-Net when used on volumes (Milletari et al., 2016)) convolutional neural network. These networks have a downsizing portion which uses a series of convolutional layers to transform the input, learning its features and compressing it into a lower-dimensional representation of itself. This is then followed by an upsizing portion, where the network increases the size of the image again while adding convolutions over the upsized image and the corresponding-size downsized image from previously in the network via skip connections. This upsizing portion works to transform the image into the desired target image, in our case the final density field, while using information from the downsizing portion of the network to inform larger-scale features. I have written a full tutorial on how every step of a U-Net works, available on github at github.com/apetulante/UNet_Tutorial. This tutorial is also included in its entirety as Appendix 1 in this document.

3.3.1 The Architecture

The network architecture that we use in this work is shown in in Figure 3.2. The network consists of 6 convolution layers which progressively downsize the image from (128,128,128) to (16,16,16) using 3 blocks of (3x3x3) filters followed by (5x5x5) filters. This is mirrored by 6 up-convolution layers, which each involve an up-sampling of the image followed by a convolution. These convolutional layers also use 3 blocks of (3x3x3) filters followed by (5x5x5) filters. Each convolution is followed by a rectified linear units activation, and a batch normalization. We end the network with 64 1x1x1 filters followed by 1 1x1x1 filter. These are necessary to return the image to its original size, as the number of ”channels” of an output image will equal the number of filters used in the previous layer. Finally, our

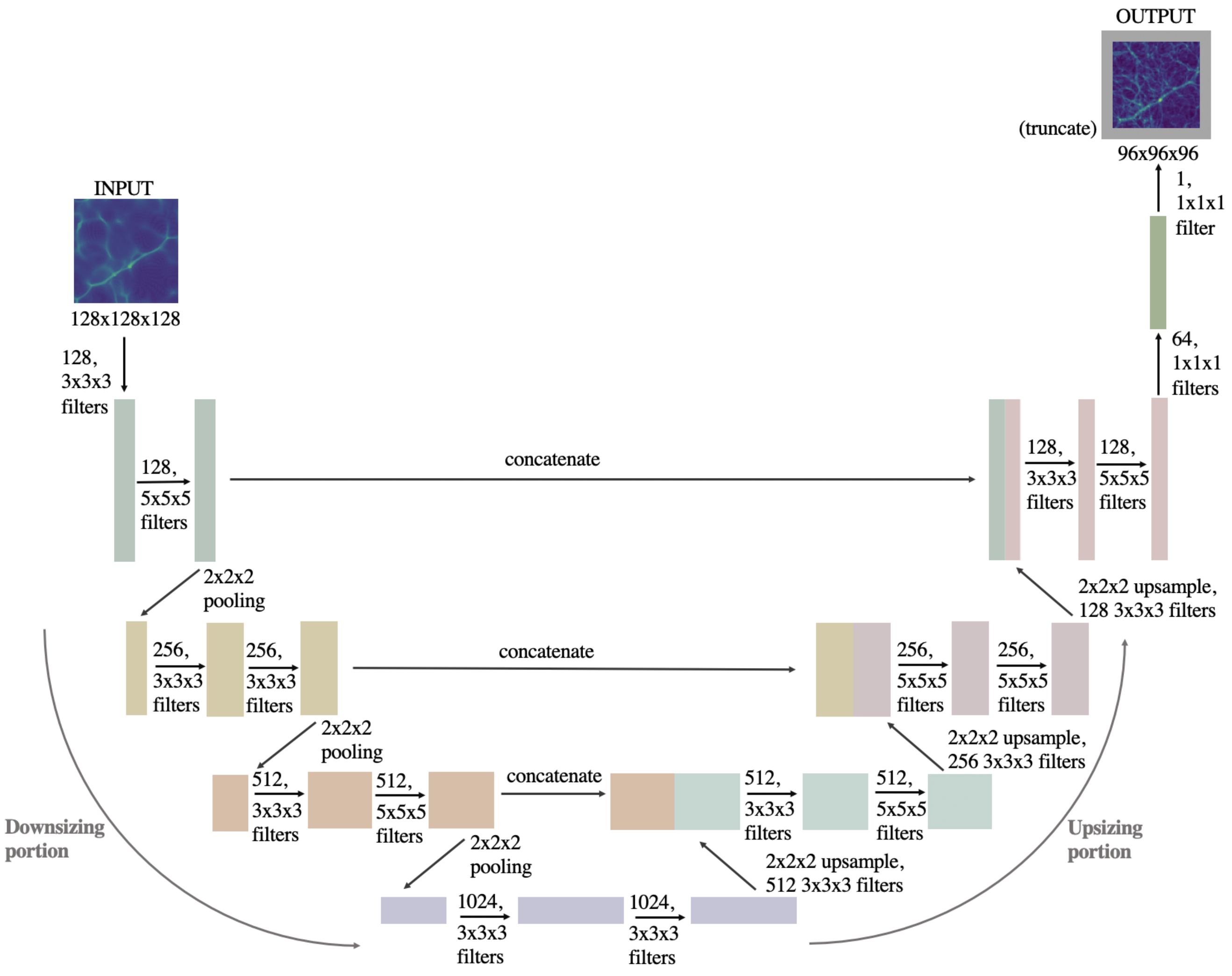


Figure 3.2: The network architecture that we use.

network ends with a truncation layer, which cuts off the outer 16 pixels on all sides of the final image.

3.3.2 Training Procedure

We train the network using the Adam Optimizer (Kingma and Ba, 2014), with a learning rate of 1x10^{-5} and first and second moment exponential decay rates equal to 0.9 and 0.99, respectively. Due to the size of our data, we can not load all of our data into memory at once. Instead, we train the model in batches of 128 pieces of data, randomly drawn from our full dataset, with the model updating after averaging the weight updates from all 128. While this makes the loss highly noisy during training, the loss still generally trends downward and becomes less noisy as the model is better generalized. This also leads to the model requiring many epochs to have had time to see all of the pieces of data, so our model only converges after 3000 epochs. During training, we monitor the validation loss, and attempt to halve the learning rate once the moving average of the validation loss has not decreased in 100 epochs. We stop training once this learning rate change does not improve the loss either. This smoothing of the validation loss allows us to robustly monitor when it generally stops trending down, as any given epoch may move to a worse validation loss given the training is done on random batches.

We use the mean-squared error (MSE) as the loss function:

$$MSE(y_{\text{pred}}, y_{\text{true}}) = |(y_{\text{pred}} - y_{\text{true}})^2| \tag{3.3}$$

The MSE is known to have some issues when it comes to image reconstruction, namely that it has a tendency to produce blurry final images. This is because the MSE requires a direct pixel-by-pixel comparison between the target and predicted image, so it can be more easily reduced by smearing out features in the predicted image. Nonetheless, we find that the MSE as the loss function produces the best results from the pixel-by-pixel comparison metrics that we tried. The majority of this smearing effect appears to happen in the lower-density regions of the image, which do not contain as many of the important structures as the higher-density regions. In Section 3.5, we explore some ways to mitigate this issue.

3.3.3 An Alternate Approach with Conditional GANs

One alternate choice for the loss function is to replace it instead with another network - called a discriminator network - whose job is to tell if the generated images match the real images. The main advantage of this approach is that the network remains unconstrained by the choice of a specific loss function - rather, the entire discriminator network can act as a much more complex heuristic for determining if the generated image matches the target image, or essentially as a learned loss function itself. The generator network (in this case, our U-Net) and the discriminator network are jointly trained as one unit called a generative adversarial network (GAN) (Goodfellow et al., 2014). The conditional GAN is a particular flavor of GAN where the training is also conditioned by giving to the discriminator some additional information about the desired output (Mirza and Osindero, 2014), for instance by having the discriminator determine if pairs of (input, output) images are real or fake, rather than if just the generated image is real or fake. This additional conditioning allows the set of potentially real-passing generated images to be more constrained to match a specific desired output.

To compare these different methods of training a generator network, we adapt a pix2pix-like architecture for 3D images and train this model as well on our simulation boxes (Isola et al., 2017). The pix2pix includes an L1 loss component to the generator loss to encourage generated output images to match exactly the target image, which is the desired outcome of this work. Pix2pix also uses a PatchGAN as the discriminator network. A PatchGAN classifies patches of an image as real or fake, creating a map of regions from the image that each correspond to an independent prediction of realness or fakeness for each patch. One potential issue with the PatchGAN architecture for our use case is that each patch taken from the image is classified assuming independence from the other patches. In truth, all regions in a simulation density field are causally connected, so density fluctuations on scales larger than the size of the patches will not be evaluated even though they exist. Although this is a potential cause for concern, we do not modify the PatchGAN discriminator used by pix2pix beyond changing the input shape. The pix2pix PatchGAN uses patches which cover 70x70x70 pixel regions of the image, which for our grid cells corresponds to a 70x70x70 Mpc box. This is still a large region of the density field, with only the largest mode of density fluctuations being inhibited by this patch size. The main advantage of the PatchGAN - that focusing on smaller regions of the image may improve predictions of

”high-frequency”[1] structures in the image, which correspond to smaller scales in the power spectrum - may outweigh the detriment that using patches brings.

We adapt the pix2pix architecture to our data by changing all convolutions to be in 3D in order to be suitable for our volumes. We additionally remove the two layers at the bottom of the ”U” of the generator so that we can keep our input boxes as (128,128,128) rather than resizing them to (256,256,256) as pix2pix normally requires. In the training procedure, we change the relative weightings of the L1 loss component and the generator loss in the total loss that is used for updating the generator network. We give slightly more weight to the L1 loss, as we found this to be necessary to produce more identical images to the target output. We also decrease the learning rate for the generator model to $5\text{x}10^{-5}$ as we found that for our case the generator was learing much faster than the discriminator. All other aspects of the training procedure remain as described completely in Isola et al. (2017).

3.4 Results

3.4.1 MSE-Trained U-Net

We begin by presenting the visual results of the U-Net when trained using MSE as the loss function. Figure 3.3 shows, for three of our test boxes, the input density field, predicted density field, and true density field. The first column shows the input fields for three 1 Mpc slices of a (96 x 96 x 96) target output box. The middle column shows the prediction of the U-Net for the same slice from that same volume. The last column shows the same slice from the target (true) box from the full simulation. The colorbars shown on the right are scaled to the maximum and minimum of each of the respective trio of images, so that a pixel’s color across all three images corresponds to the same value.

The large scales patterns of structures look similar in all three images, with the input

[1]Here, we use ”high-frequency” as it is commonly used in the field of computer vision. High-frequency structures are those that have high variation over small regions in the image - often appearing the most crisp or sharply defined. This is not related to the frequency with which a pixel value appears, which we show in our histograms.

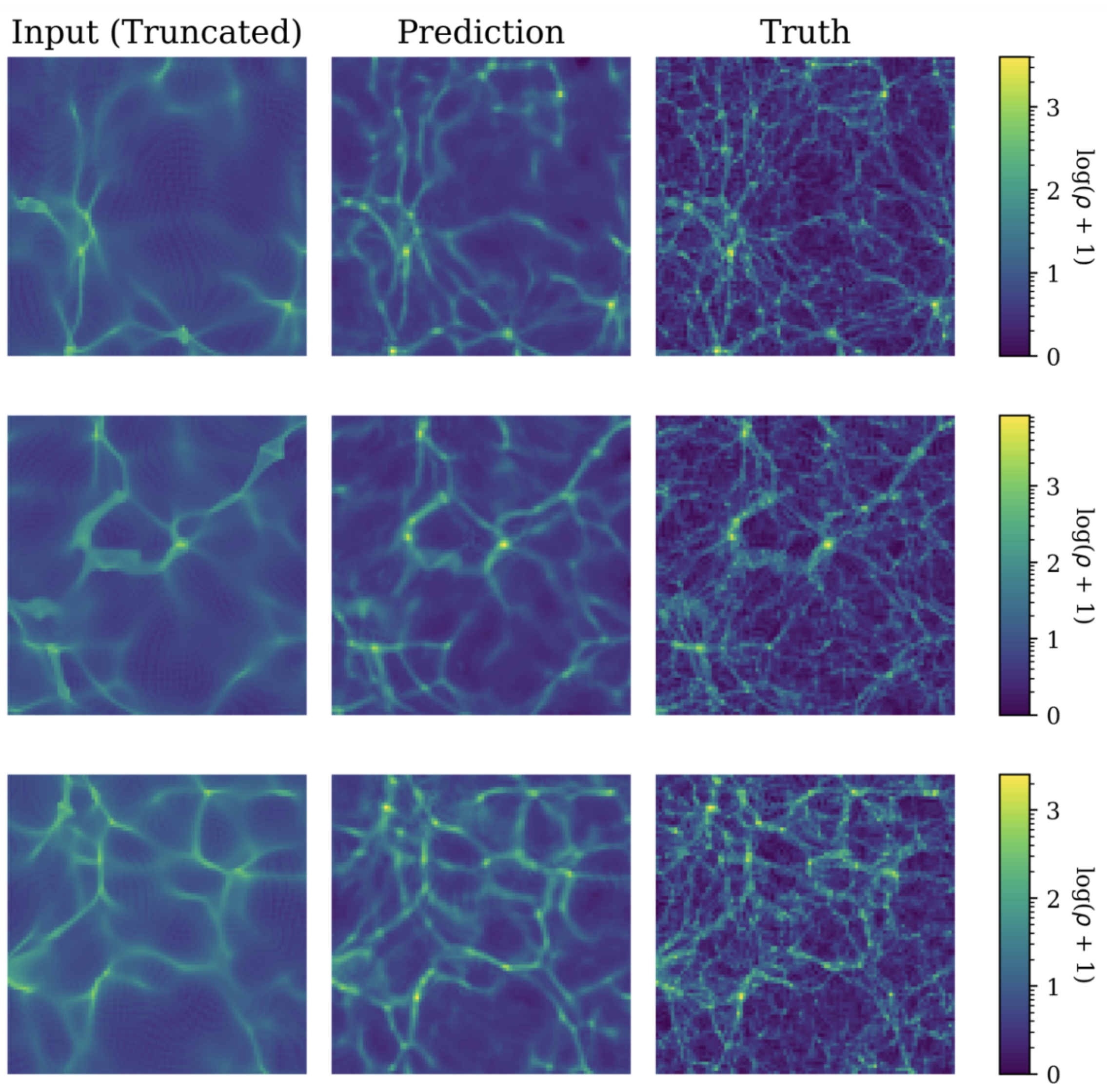


Figure 3.3: An example of the input (first column), predicted (middle column), and true (last column) density fields for three simulation boxes in our test set. The input images are truncated by 16 pixels on each edge to be the same size as the predicted and target images for easier comparison. The colors shown correspond to counts of $\rho + 1$ rather than ρ to account for the undefined nature of log(0).

density field of 2LPT run to z=0 already placing the broad strokes of density fluctuations in the right places. However, the prediction panels show that the U-Net has both sharpened some of those structures as well as added some structures into the regions where the 2LPT input had only very diffuse structure. In particular, the highest density peaks appear to be generally the correct size and pixel value in the predicted image. Compared to the target images however, the predicted boxes look distinctly blurrier. The lowest density regions appear to be predicted the worst, with the true image having a significant number of fine, thin filaments in the lower-density regions which are not in the predicted image at all. The lowest density regions of the image are also generally not dark enough in the prediction, with the overall background color of the predicted images appearing as a lighter blue than the background of the truth images.

3.4.1.1 Pixel-by-Pixel Comparisons

We examine the distribution of densities in our predicted images more closely in Figure 3.4, which shows a histogram of pixel values averaged across 64 of the predicted, corresponding true, and corresponding input boxes in our test set. Above $\sim$4000 counts/pixel, the true and predicted histograms match in shape very well. This can be seen visually in Figure 3.3, where it appears that the highest density, brightest pixels are typically reproduced accurately. Between $\sim$100 and $\sim$4000 counts/pixel, the U-Net predicts less density than is in the true image. In Figure 3.3, these regions typically appear as green pixels. This discrepancy seems to come primarily from a lack of fine filamentary structures in the lower density regions of the images. The predicted images have almost all but the largest filaments significantly blurred out, but less dense areas in the truth images have many small structures defined throughout. Instead, these regions in the predicted images seem to take on the lighter blue color that is associated with pixels of counts $\sim$10. This can also been seen in the histograms, with the model significantly overpredicting pixels to be in the $\rho \sim$3 - $\sim$30 density range and the peak pixel value of the model pixels occurring at $\rho \sim$4 rather

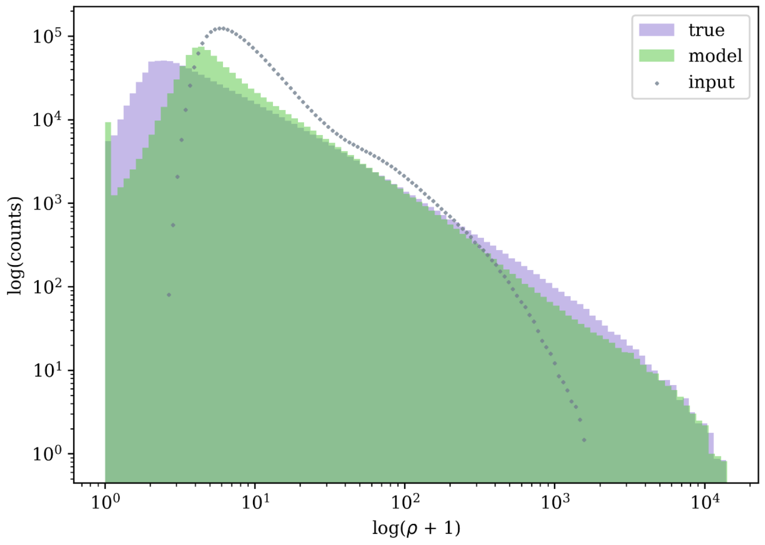


Figure 3.4: Histograms of the pixel densities of the predicted and true density field, averaged for 64 boxes in our test set. The histogram of true values is shown in purple, and the histogram of predicted values is plotted on top in green. We also show, in light gray dots as to not cover the other two histograms, the outline of the histogram of input boxes. We show the counts of $\rho + 1$ rather than ρ to account for the undefined nature of log(0).

than at ρ ~1 like in the truth. This also shows up as a significant underprediction of pixels in the range ρ = 0-4, which again seem to have been given slightly higher values in the predicted images.

At the first bin in the histograms, which corresponds to ρ = 0, there is a distinct spike in the model histogram. This is due to the final activation function of our model being ReLU, which does not allow negative values and instead makes them zero. This is necessary because our final density field should not contain any negative values. However, as the convolved image coming out of the final layer in our model, before the ReLU activation, is allowed to contain negative values, this artifact remains in the final image.

We next take a closer look at some particular regions of our predicted boxes to further investigate where the model tends to fail. Figure 3.5 shows, for three 40x40x1 Mpc patches of our predicted boxes, the corresponding truth patch, and the subtraction of the predicted patch from the truth patch. It is clear from these images that what we have speculated about the locations of the density discrepancies between truth and prediction is true. Overall, the

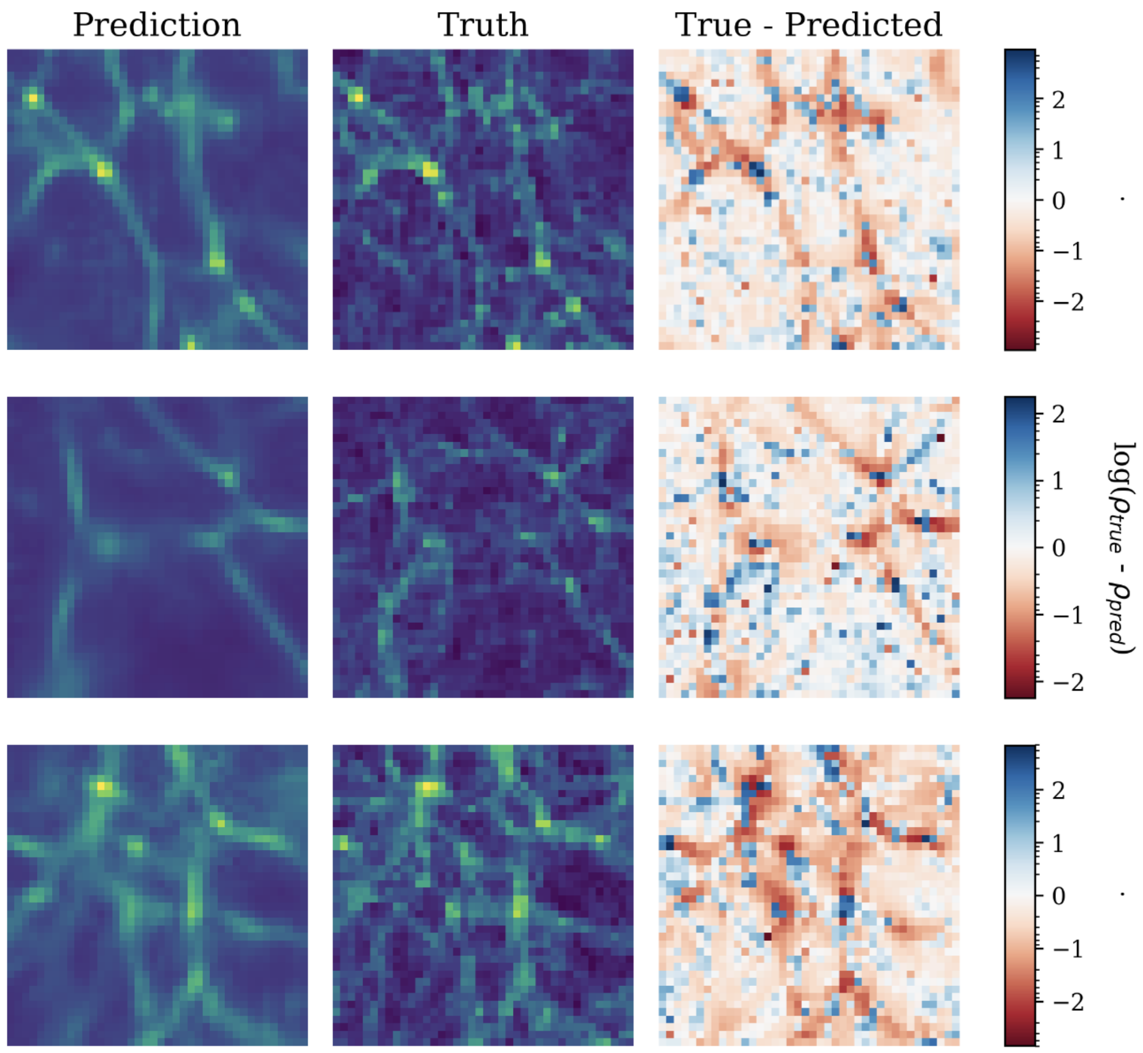


Figure 3.5: Patches taken from our predicted and true test boxes, and the difference map between them. Each patch is a 40x40 region taken from a 1Mpc thick slice of one our test boxes. The first column shows patches taken from truth images, the second column shows patches from the same predicted images, and the third column shows the predicted patch subtracted from the truth patch. The difference map is log-scaled to better show variation. All colorbars are centered at zero, so blue pixels indicate excess density in the truth compared to prediction, and red pixels indicate excess density in the prediction compared to truth.

highest density structures are predicted well, with a trend towards excess density in the predicted image by ∼10 - 100 counts, as can be seen in the major structure in the third row of Figure 3.5. It is clear from all of the panels that the majority of the blue pixels, where the truth is in fact higher in density than the model predicted, form thin filaments that are not resolved by the blurry lower-density regions of the prediction. We can also confirm from the reddish background on all of the subtracted images that the tendency for the prediction is to put too much density into what should be nearly completely empty regions of the density field.

The performance of the model in these ranges points to particular strengths and weaknesses of the MSE as the loss function. Although the general size and location of the brightest pixels is accurate, we can see the MSE's tendency to shift structures in the highest density pixel of the patch in the first row of Figure 3.5. It is also clear from all of these pixel-by-pixel comparisons that using the MSE as the loss function generates overall smoothed out final images. This is most evident in the lowest density regions, where finer structures are smoothed out into a diffuse background over the entire image. The shifted peak in the histogram is likely due to this smoothing effect, where predicting pixel values in a mid-range of ∼3 - 30 helps to minimize the loss in regions where the density was both supposed to be slightly lower and slightly higher,

3.4.1.2 The Power Spectrum

The power spectrum is one of the most common summary statistics that cosmologists use to quantify the density field as a function of physical scale. It is given by:

$$P(k_\mathrm{i}) = \frac{\sum_{k_\mathrm{i} < k \leq k_\mathrm{i+1}} \langle |\delta(k)|^2 \rangle}{(N_\mathrm{i} V)} \tag{3.4}$$

Where δ(k) is the Fourier transform of δ(x) the overdensity field, N_i is the the number of k's that fall into the given k bin $(k_\mathrm{i} < k \leq k_\mathrm{i+1})$ and V is the volume of the region in k-space. The wavenumber k corresponds inversely to physical scale, so a larger k represents structures of smaller physical sizes. We calculate this power spectrum using `nbodykit`'s (Hand et al., 2018) `FFTPower` on our gridded density fields. In Figure 3.6, we show the average power spectrum for 64 of our input, predicted, and true boxes in our test set in the top panel. In the bottom panel, we show the transfer function *T(k)* as used by Feng et al. (2016); He et al. (2019); de Oliveira et al. (2020):

$$T(k) = \sqrt{\frac{P(k)_{\text{predicted}}}{P(k)_{\text{true}}}} \tag{3.5}$$

for more straightforward comparisons to their results. *T(k)* is defined such that values of $T > 1$ correspond to the power in the predicted density field being larger than power in the true density field, and values of $T < 1$ imply less power at that scale in the predicted density field than in the true density field.

At small k-values, the predicted and true power spectra are nearly indistinguishable, matching for $k < 0.5$ Mpc^{-1}h to under one percent. At larger *k*, the true and predicted power spectra begin to deviate, with the model not predicting enough power at smaller scales. By *k* = 1 Mpc^{-1}h, the difference between the power spectra has grown to 5%, and at the smallest physical scales resolved by our gridding resolution, *k* = 3 Mpc^{-1}h, the difference is 30%. While these predictions significantly degrade at larger *k*, it can be seen that the predicted power spectrum is remarkably better than the input power spectrum at all scales, which does not match perfectly at even the smallest *k*, and has degraded to a 30% deviation from the true spectrum at $k \sim 0.2$Mpc^{-1}h.

The performance of our model with regards to the power spectrum is similar to existing works. He et al. (2019) reproduce the results of FastPM (Feng et al., 2016) very well at all scales, only deviating from the FastPM power spectrum by a few percent at their largest *k* = 0.7 Mpc^{-1}h. FastPM itself is accurate when compared to the fully nonlinear power spectrum to 5% at *k* = 1 Mpc^{-1}h for their longer 40 timestep run, and to ∼20% at *k* = 1 Mpc^{-1}h for their quicker 10 timestep run. The accuracy of this 40-timestep version of FastPM at *k*

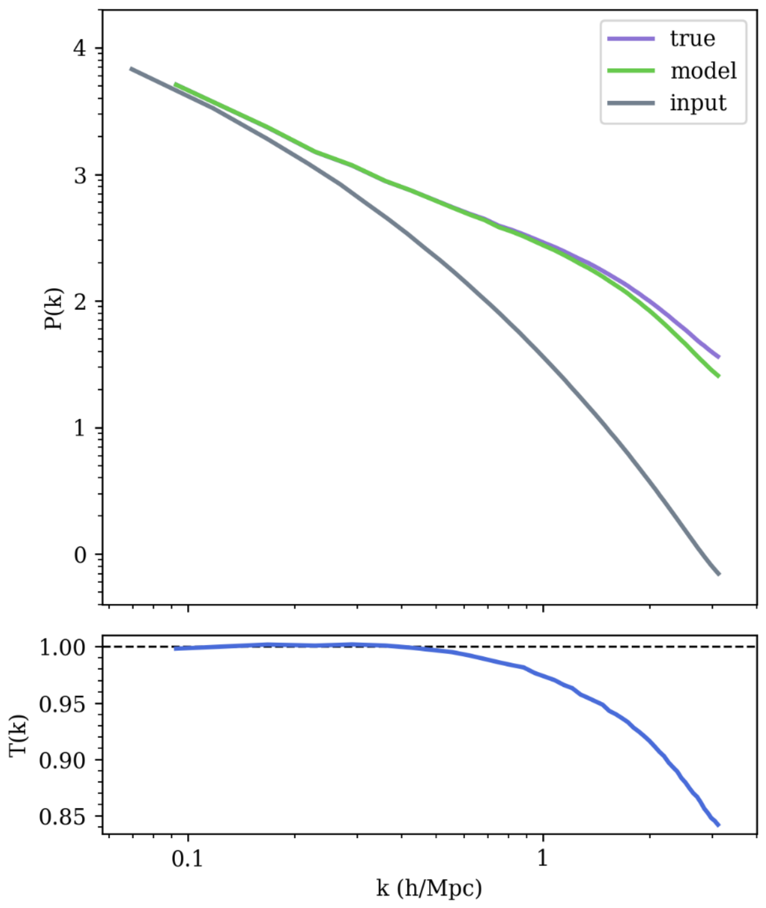


Figure 3.6: The power spectrum, for an average of 64 predicted (green line), true (purple line), and input (gray line) density field boxes in our test set. In the bottom panel, we show the transfer function as defined in Equation 3.5. At the dashed line of *T(k)* = 1, the true and predicted P(k) are identical

.

= 3 Mpc^{-1}h is also similar to our results, where they achieve a 25% difference from the nonlinear power spectrum. de Oliveira et al. (2020) achieve better results, maintaining percent-level accurate results across all scales, down to k as small as $\sim$1 Mpc^{-1}h.

It is important to remember in these comparisons that the simulation resolution is very different in these works compared to ours, which has major consequences for the power spectrum. He et al. (2019) use a simulation with 32^3 N-body particles in a volume of 128 Mpc/h, meaning their simulations contain an average of $\sim$0.016 particles/Mpc whereas our simulations contain on average $\sim$11 particles/Mpc. de Oliveira et al. (2020) use simulations with 512^3 particles in a 1 Gpc/h box, meaning the underlying simulation has $(512/2240)^3$ = 0.012 the resolution of ours. They do however use the displacements of all 512 particles rather than gridding them as we do, leading to an effective $(512/1000)^3$ = 0.13 fraction of our resolution in the target boxes.

3.4.2 GAN-trained U-Net

In Figure 3.7, we show the visual results for the same three test boxes as in Figure 3.3, but for our GAN model. Immediately, the visual results appear somewhat worse as compared to the MSE-trained U-Net. While the predicted images in this set appear to lack some of the fuzziness plaguing the MSE-trained model, features appear more sharp but still too diffuse in their overall shape compared to the truth. It does appear that, as compared to Figure 3.3 the dynamic range of the predicted and true images matches a bit better, with void regions being properly more empty and high-density regions having the correct brightness.

The lack of defined filaments is the most notable difference. For the MSE-trained U-Net, the predicted image appeared as an overall blurry version of the true image, with larger filaments appearing in the correct place and maintaining their rough shape that they had in the input image. The MSE-trained U-Net did not appear to erase where 2LPT had these larger structures, only sharpen or move them slightly. This is not the case for the predicted images of the GAN-trained U-Net, where it appears that many of the larger

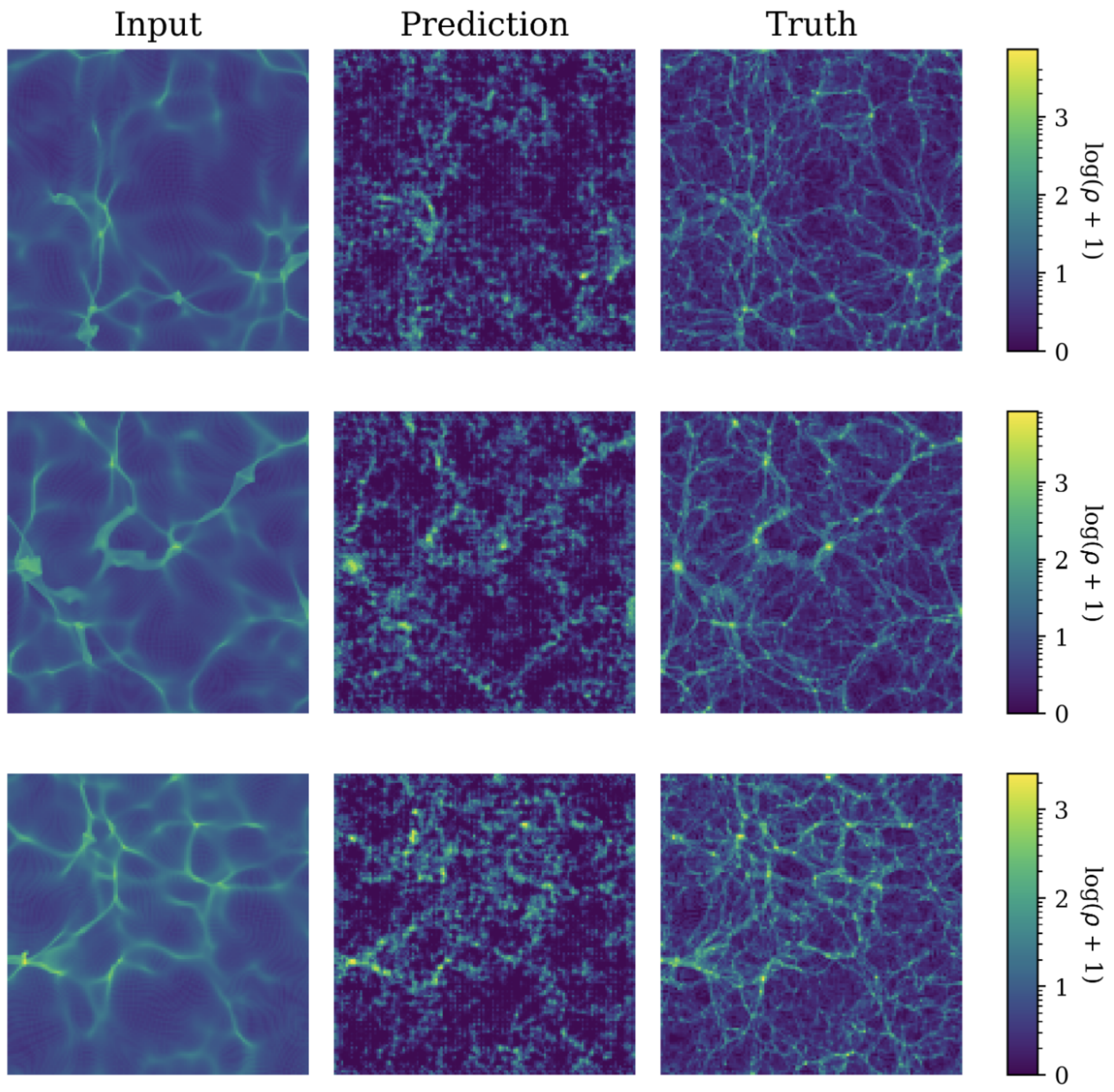


Figure 3.7: The same as Figure 3.3, but for the GAN-trained U-Net instead.

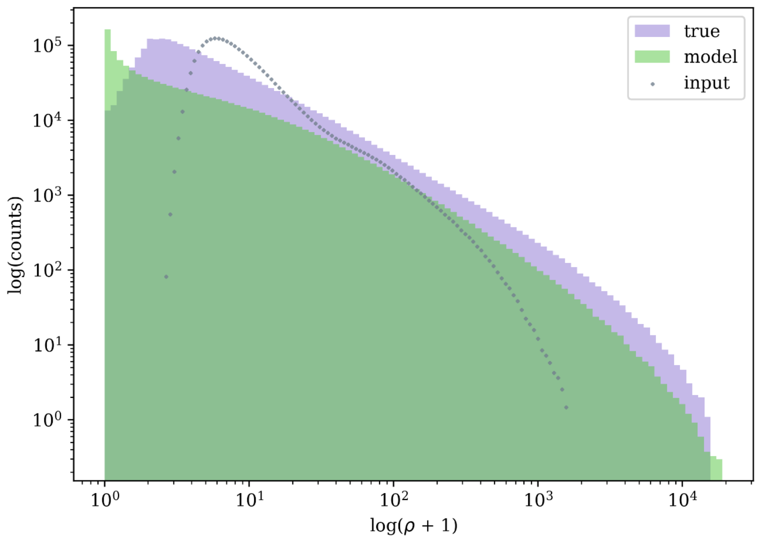


Figure 3.8: Same as Figure 3.4, but for the model that was trained with a conditional GAN structure.

filaments instead become bumpy looking and ill-defined. The highest density regions of these filaments appear in approximately the same place, but the smooth, thin structures that connect them are mostly gone.

In the low-density region, the GAN-trained U-Net does place some structure where the MSE-trained U-Net did not. Rather than an overall smooth background, the predicted images in Figure 3.7 have more mottled structures, which do better represent the thin clumps of filaments that appear in the lower density regions of the true images. The highest density pixels appears reproduced fairly well, although again not as much so as for the MSE-trained U-Net. It appears that the primary improvement for this model over the MSE-trained U-Net is in the lower-density regions, at the overall expense of higher-density pixels.

3.4.2.1 Pixel-by-Pixel Comparisons

We next investigate the pixel density histograms for this model and inspect if these visual differences are also apparent there. Figure 3.8 shows the density distribution as pixel counts for an average of 64 of our true, predicted, and input boxes in our test set. The trends in these histograms are quite different to those in Figure 3.4.

First off, it appears from Figure 3.8 that the GAN-trained U-Net has the opposite problem from the MSE-trained U-Net at the lowest densities. Rather than drastically underpredicting pixels with very low densities (ρ= 0-1), they are overpredicted. This seems to come at the expense of density everywhere else, as the GAN-trained U-Net underpredicts all other densities in the image. These trends can be confirmed when looking at the generated images for this model. The predicted images in Figure 3.7 show that the voids have a generally lower density than they do in the truth, with a much darker overall background color in the image. This can be seen in the histograms as the sharp spike in the first bin. Although it appears that there is more structure in the lower density regions, the unorganized nature of this structure makes it visually hard to compare to the truth.

Another potential reason for these large discrepancies is that the GAN-trained model lacks in total density as compared to the truth. For the MSE-trained model, the total density in the predicted and truth boxes was roughly similar over an average of many. However, the average density of the GAN-trained model's predictions is only $\sim 88\%$ of the average density of the truth. This is in theory an easy post-processing fix when predicting a full simulation box, as the total mass is known and can therefore be preserved. For smaller boxes that are cut from a larger simulation, it is harder to robustly correct the total mass. For this reason, it may be useful to enforce that the total mass of the box matches during training, when the truth is allowed to be known to the model.

As with the MSE-trained model, we inspect some patches of the true and predicted density fields in Figure 3.9. We select the same patches that we used in Figure 3.5 for more direct comparisons between the two models. As we suspected, the regions that the GAN-trained model seems to have trouble with are quite different from what the MSE-trained model struggled to predict. Most obviously, the the subtraction patches in Figure 3.5 were mostly red, indicating that, particularly in the lower-density regions, the tendency of the model was to predict too much density. The opposite is true of the GAN-trained U-Net, where instead the tendency is to underpredict density. This of course matches what we can tell by the histograms, that the GAN-trained model makes too many very low density

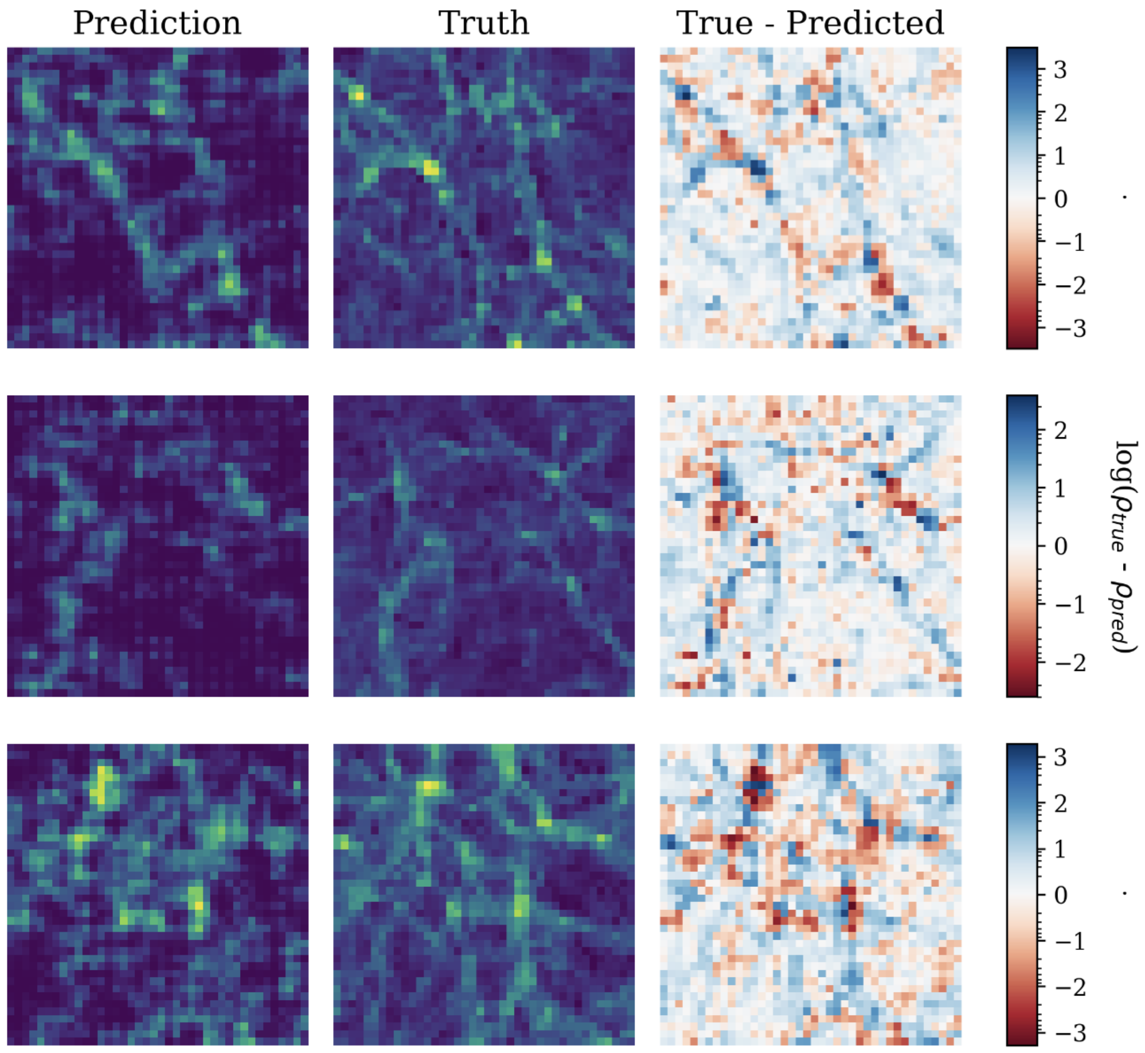


Figure 3.9: Same as Figure 3.5, but for the GAN-trained model.

pixels.

Despite the results in Figure 3.8 suggesting that there are not enough overall higher-valued pixels in the predicted images, it seems from the subtraction patches in Figure 3.9 that the model also struggles to put its highest density pixels in the correct spot. For instance, in the patches in the first row in 3.9, where the bright patch of pixels in the lower right of the image is roughly the correct brightness when compared to the truth, but is shifted relative to the truth. In the bottom row of 3.9, the brightest patches are in roughly the correct spots, but are not as compact as in the true image.

3.4.2.2 The Power Spectrum

The noticeable improvement in the GAN-trained model over the MSE-trained model is in its performance on the smallest scale of the power spectrum. In Figure 3.10, we show the average of 64 input, predicted, and true power spectra on the top panel, and on the bottom panel we show the residual quantity *T(k)* (given by Equation 3.5), the same as in Figure 3.6. On large scales, the true and predicted power spectra match very closely, the same as the MSE-trained model did. However, at smaller scales the power spectrum is reproduced slightly more accurately.

At large k-values, the predicted and true power match well, to under one percent until $k < 0.3$ Mpc^{-1}h, slightly worse than the MSE-trained U-Net that matched until $k < 0.5$ Mpc^{-1}h. Just like with the MSE-trained U-Net, at larger k, the true and predicted power spectra deviate, with not enough power at smaller scales. By $k = 1$ Mpc^{-1}h, the difference in the power spectrum has grown to 10%, and at $k = 1$ Mpc^{-1}h, the difference is up to 35%.

This is overall worse performance than the MSE-trained model in all but the last 2 points on the power spectrum, at the highest k, where the difference instead decreases back down to 15%, at $k = 3$ Mpc^{-1}h. Even though for other smaller scales, 0.3 Mpc^{-1}h $< k <$ 1 Mpc^{-1}h, the GAN-trained model does not do as well, the better performance in this final bin is promising, as the smaller the scales, the harder making accurate predictions is. This,

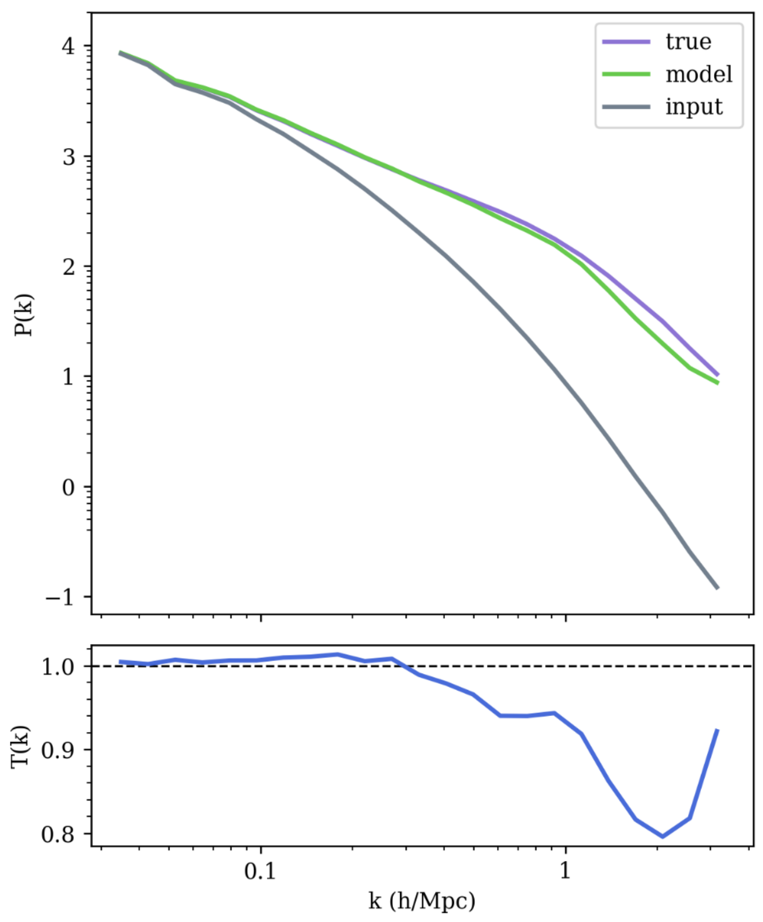


Figure 3.10: Same as Figure 3.6, but for the GAN-trained U-Net instead.

combined with the overall lack of smoothness in the power spectrum, which could perhaps be mitigated with more data, show that there is still much potential for using conditional GAN's to train a simulation generator.

3.5 Discussion

In this work, we utilize U-Net convolutional neural networks to predict the z=0 density field, as generated from an N-body simulation, from the the density field of initial conditions run forward to z=0 using 2LPT. We train both a U-Net in a standard manner with an MSE loss function, and a modified version of a pix2pix GAN with a discriminator network acting as the loss function.

The overall performance of the MSE-trained U-Net was better on almost all accounts. Visually the images were significantly blurred, but the sizes and scales of the higher density structures were well reproduced. The minor improvement of the GAN-trained U-Net was in the voids of the images, which were appropriately more empty with some fine structures throughout. However, the GAN-trained model did have a tendency to predict too many very low-valued pixels, so the improvement here still comes with a cost.

We find that the MSE-trained U-Net performs better on all scales of the power spectrum except for the largest $k = 3.0$ Mpc^{-1}h. Whereas the MSE-trained U-Net drops to a 30% difference in predicted and true power spectrum at this scale, the GAN-trained U-Net increases in accuracy to a 15% difference at this scale. It is for this reason that we believe that there is still much potential for this method of training. Because the discriminator network is able to pick out much more complex differences between the predicted and target image during training, which unlike with the MSE loss are not easily minimized by blurring out features in the output, it seems that it may have potential for improving the smaller scales of the power spectrum where the MSE would be unable.

In this study, we have made use of a 1000 Mpc simulation box with 2240^3 particles, gridded with a 1Mpc grid. At this spatial resolution, the growth of structures is highly nonlinear. Nonetheless, we are able to match the power spectrum with the MSE-trained U-Net down to$k$= 1 Mpc to 5%. We believe that these results show much promise for the continued use of deep learning techniques to learn smaller scale structure formation when provided with higher resolution simulations. These results also align fairly well with those from He et al. (2019) and de Oliveira et al. (2020), suggesting that using a significantly smaller grid size does not affect the model's ability to capture larger scales at all.

One very important comparison that we did not explore here is to replace the generator network in the pix2pix pipleline with the architecture that we use for our MSE-trained U-Net, which we have already found to fit the problem quite well. In this work, we have minimally adapted pix2pix to fit our image size, however, the theory behind conditional GAN's, with a PatchGAN as the discriminator as pix2pix uses, can be applied to train any U-Net architecture. This would be the natural next step for the results that we have found here.

3.6 Lessons Learned and Looking Ahead

In getting to the model presented in this work, there was much that we tried, sometimes with some success, and other times with significantly worse results. Here, we discuss some of the most important takeaways from those tests. Where applicable, we give recommendations for what we believe to be the most promising avenues for continued work on this and similar problems.

3.6.1 Loss Functions

The choice of loss function is one of the most important determinants of success when it comes to training our generator model. Pixel-by-pixel loss functions clearly have issues producing blurry images, missing the majority of structures in low-density regions. While a slight blurring effect doesn't matter significantly for larger scales of the power spectrum, it is evident that our MSE-trained U-Net drops significantly in performance at smaller scales of the power spectrum, which may be largely due to an almost complete lack of any fine structures in the lower-density regions of the image. On the other hand, the GAN-trained U-Net was much more successful at visually producing more small-scale structures in the low-density regions, albeit without the same amount of definition that is seen in the truth. While the GAN-trained model has much better reconstruction of the very smallest scales of the power spectrum, at mid-scales of k it performs worse than the MSE-trained model. Nonetheless, we conclude that going forward, a discriminator loss is likely the best way to train a generator network for producing faux density fields, as recovering small scales of the power spectrum is incompatible with a loss that produces highly blurry images.

Given that it is the primary evaluator of a successfully generated density field that we use in this work, one obvious extension might be to specifically train the model based on its accuracy on the power spectrum, by including performance on the power spectrum in the loss. However, as the power spectrum lacks much vital information about the arrangement of structures in the universe, we have not found this to improve the training of our model, with the primary challenge of this approach likely being that an infinite number of density fields satisfy the criteria of a matching power spectrum, so solely optimizing for a matching power spectrum would most certainly not lead to a matching density field. Of course, the accuracy of the power spectrum can be added as a component to the loss, which perhaps with the correct relative weighting or with a dynamic weighting scheme could improve results, however we did not find this to be the case with the weightings which we did test.

3.6.2 Dynamic Range of Data

One potential issue in training these models may lie in the enormous dynamic range that the density field contains. As can be seen in Figures 3.4 and 3.8, while the majority of pixels are fairly empty, those that do contain the highest density structures are often 10^4 - 10^5 higher in value than the majority of the image. This can make training, especially with pixel-by-pixel losses, difficult to tune and quite finicky when it comes to for instance the specific choice of learning rate depending on the chosen loss function. With pixel-by-pixel losses in particular, there is a danger that the model quickly memorizes the highest-value pixels as a way of ”easily” significantly reducing the loss, and given the relatively very small number of these high density pixels, a very large amount of data is needed for this to not become inevitable. The use of a GAN to train the generator may also mitigate this issue, as the pixel-by-pixel loss component to the generator and the discriminator loss from the PatchGAN may work together to capture the transformation in both the high density and low density regimes.

Another option is to scale the input data, for instance logarithmically, before giving it to the model, which would shrink the dynamic range. In our testing of different models for this work, we again found that the logarithmic scaling didn’t appear to help. The solution may instead lie in a different scaling scheme for the input data, which even more drastically reduces the relative range of the pixels.

One other promising way to deal with this would be to instead model a displacement field rather than a density field. The displacement field specifies the movement of a particle in the x,y, and z dimensions over some time. He et al. (2019) and de Oliveira et al. (2020) use the displacement field as the input in their work, with very good results. The downside of using the displacement field is of course that it must be specified in 3 dimensions: x,y and z, tripling the amount of data. However, it also contains more information than the density field, so if a model is able to compensate the additional size, it could likely lead to better results.

3.6.3 Next Steps for Science

In this work, we primarily have focused on reproducing the density field. However, in order to accurately make halo catalogs, velocities would be required. In principle, predicting the gridded velocity field is as straightforward as predicting the gridded density field. Additionally, as the densities and velocities are complementary information, predicting them together would likely improve the accuracy of both. However, each component of the velocity in x, y, and z would require its own field equal in size to the density field, so predicting both density and three velocity components would quadruple the size of the input and output data that we have used here.

From a science perspective, it will also be necessary to continue pushing the resolution of generated simulations in order for them to be useful for more types of work. To generate halo catalogs that contain information about galaxy-sized halos, for example, a gridding smaller than 1 Mpc, which instead is on the scale of clusters, would be needed. The work we have presented here shows promise that small-scale structures can be learned, at least somewhat well. However, as structure formation is more nonlinear on smaller scales, it may prove very challenging to keep pushing the grid size smaller while maintaining the same level of accuracy.

With continued pushing to higher resolutions, and by adding additional information such as the velocity fields, memory constraints may quickly become the primary challenge for this type of work. Density fields must be sufficiently large as to contain structures across all physical scales, so smaller grid sizes will only exponentially grow the number of input pixels. However, to resolve structures smaller than cluster-size, an even smaller grid than the one used in this work will be required.

CHAPTER 4

Conclusions

In modern astrophysics, dark-matter only simulations are an essential tool for studying structure formation. However, the computational expense of running these simulations remains a barrier for generating large, high resolution simulations. While astronomers have creative ways of reducing the computational cost or analytically modeling some processes rather than simulating them, the need for full simulations remains to accurately model many behaviors. In this dissertation, I have explored how machine learning might provide solutions to some of these computational problems, as it is an extremely powerful tool for modeling behaviors without needing any a priori knowledge about how the input and output relate.

4.1 Summary

In Chapter 2, we used random forests and gradient boosting regressors to predict the survival, mass loss, final position, and merge time of a subhalo from a list of physically-motivated features taken at the time of its initial infall into its host halo. In doing so, we were able to probe the degree of stochasticity in subhalo evolution, testing whether or not analytical models which approximate the fate of subhalos based on their properties at infall time are beginning with consistently behaving subhalo evolution. We found that subhalo survival can be predicted well, but that mass loss, final position, and merge time were more stochastic processes, with very similar initial conditions not necessarily leading to similar outcomes. Only five input quantities (redshift, impact angle, relative velocity, and the masses of the host and subhalo) were needed to determine almost all of the subhalo evolution learned by our models. We also found that some subhalos were harder to make predictions for than others. In particular, those entering at mid-range redshifts (z = 0.67-0.43) were harder to predict for all models, and those coming in with more perpendicular trajectories to the host were easier to make predictions for. We investigated a large range of possible causes of this stochasticity, and conclude that it is most likely that the evolution of subhalos within dark matter only simulations is inherently chaotic in nature.

In Chapter 3, we utilize a U-Net deep convolutional neural network to predict the z=0 density field from a dark-matter only simulation, using as input the initial conditions density field evolved to z=0 with 2LPT. With this model, we were able to reproduce the power spectrum to only 30% at k = 3.0Mpc^{-1}h, but maintained an accuracy of under a percent at the largest scales (k $<$ 0.5Mpc^{-1}h, and dropped to only %5 error for k = 1.0Mpc^{-1}h. Then, we investigated whether a conditional GAN has potential to make the same predictions, while relieving issues caused by using pixel-by-pixel loss functions. Our GAN model performed slightly worse on the power spectrum, instead with an error of %10 error for k = 1.0Mpc^{-1}h, and only matching the power spectrum to under one percent for k $<$ 0.3Mpc^{-1}h. However, at k = 3.0Mpc^{-1}h, the GAN-trained model only had an error of 15% where the MSE-trained model had an error of 30%, showing that there is still much potential for the use of conditional GAN's to train U-Nets for this type of work.

Both models had challenges accurately predicting the lowest density regions of the density field. In particular, pixels that housed single-digits of particles were not predicted well by either the MSE U-Net or the GAN model, with the MSE U-Net underpredicting counts of low density pixels and the GAN model overpredicting vey low pixel values. While these pixels would not house any significant structures in the universe and therefore are in some ways less important to predict correctly, the inability of the models to predict them accurately points to a glaring hurdle in generalizing the modeling of structure formation for all density regimes.

4.2 Future Work

There is much potential for the continued use of machine learning for problems of structure formation, and therefore many avenues of future work to explore for both the studies covered in Chapter 2 and Chapter 3 of this dissertation.

4.2.1 For Modeling Subhalo Evolution

For modeling subhalo evolution, much of the potential future work for what we have discussed in this dissertation would involve characterizing and accounting for the merging stochasticity. For instance, subhalo interactions might play a significant role in the life of a subhalo within its host. If that is the case, accurate modeling of subhalo evolution will require more information about dynamics within the host, which may mean that obtaining information about the interaction at only its starting point is not enough. A deeper investigation into the subhalos that are most poorly predicted can give clues as to what is affecting their evolution inside their hosts. An exploration into different significant times in subhalo's orbit could also elucidate when the subhalos evolution is most chaotic. For instance, if we change the definition of the ”start” of the interaction to instead be the time of first pericenter or the time of crossing the splashback radius, the performance of these models could point to whether the stochasticity of the interaction changes as a function of time within the host.

It would be very interesting to additionally recreate this study in the context of a hydrodynamical simulation. Galaxies are known to have strong effects on their subhalo hosts through feedback and enhanced tidal effects, so subhalo evolution in a hydrodynamic simulation is likely dependent on significantly more factors. However, it would be important to understand whether or not adding information about baryons to this type of study actually changes the accuracy of predictions, as understanding whether or not the additional galactic effects might also be stochastic in nature has implications for future analytic modeling of galaxy evolution.

4.2.2 For Predicting the Cosmic Web

The primary goal of future work for predicting the cosmic web will be to improve the accuracy of predictions for smaller scales of the power spectrum and to make predictions more useful for a wider range of science applications. Improving the predictions themselves will rely on improvements to model architecture and training procedure. In this dissertation, we have demonstrated that a conditional GAN is a promising method for training a generator model if the primary goal is to accurately predict small scales of the power spectrum, and would recommend this type of architecture for future models. That being said, many changes could be made to the pix2pix framework that could better tailor it to density fields, such as changing the patch size of the PatchGAN or changing the pixel-by-pixel loss that is added to the generator loss, which we have not had time to explore in this dissertation but that may improve training. Additionally, the U-Net portion of the pix2pix architecture could be replaced with the U-Net architecture that we have developed in this work, which already has shown fairly good performance on this problem.

For generated density fields to be practicable for science that relies on smaller scale clustering, it will become necessary to push these predictions down to even smaller *k*. This would mean making predictions of density on a finer grid, as 1 Mpc/h, while being in the highly nonlinear regime, is still at the scales of clusters rather than individual galaxies. To make accurate halo catalogs from a generated density field, a grid closer to 0.25 Mpc/h would be needed to begin resolving Milky Way sized galaxies. A finer grid would also be necessary to reduce pixelization effects, which as was recently explored in Schaurecker et al. (2021), have strong effects on small scales of the power spectrum. In a similar vein, we have not predicted the final velocity field in this work, but this is also a crucial piece of information that is obtained from running a full simulations. Velocities from simulations are important for many comparisons to the observed velocity field (e.g. Nusser et al. (1991)), or to make accurate halo catalogs (e.g. Behroozi et al. (2013a)).

In this dissertation, I have primarily focused on examining the power spectrum as a statistic for measuring the accuracy of a generated density field. However, while this is a statistic that is crucial to reproduce correctly, it is far from sufficient for ensuring that a generated cosmic web matches the observed universe for all relevant science. As Dong et al. (2021) recently discussed, many other validation metrics, such as the bispectrum, topological metrics, and the cross-power test are all tests that should be passed to have confidence that an AI-generated cosmic web truly matches a simulated one. Once future works have generated the power spectrum sufficiently well, it will be important to turn to these and additional metrics for ensuring that generated simulations can truly take the place of fully-simulated ones.

When it comes to hydrodynamics and simulating galaxies, it remains yet unexplored whether current machine learning techniques could help bypass the running of full hydrodynamical simulations by generating them in the same manner. As hydrodynamic simulations are enormously more complex, this is likely a much further frontier. However, machine learning shows promise for semi-analytic type frameworks of modeling galaxy formation on top of dark-matter only simulations (e.g. Kamdar et al. (2016)), which, when applied on top of a generated density field could mean a pipeline for a fully-simulated galaxy catalog is reasonably in the future. In order to further probe the viability of this pipeline, it would be of particular interest to do a detailed comparison of a halo catalog generated from a simulated density field versus that of a generated density field, especially if the halo catalog were generated from a pixelized density field which would likely have significant constraints.

4.2.3 On Machine Learning's Continued Usefulness

Of course, as the field of machine learning continues to rapidly expand, new technologies may be key for obtaining the high level of accuracy that current models lack. The analysis of Chapter 2 could be completely redone with a more complex model, which could incorporate information about all other nearby subhalos in an attempt to capture more information about dynamics in the host that may be responsible for some of the stochasticity. Perhaps a model that aimed to predict each step in a subhalos infall, in the full context of how all other subhalos in that same host were evolving, could lead to a machine learning model that highly accurately determines subhalo fates, but still only needs information on infall for every subhalo.

New technologies may also be what is needed to someday make running N-body simulations all but unnecessary. Some recent developments in the field, including the emergence of transformer models, provide ways to better include information about all regions of the simulation box so that each pixel could be better predicted in the full context of the simulation box. Additionally, as hardware and hardware acceleration techniques continue to improve, it may be feasible to grid the density field finer and finer, allowing for very high resolution predictions and larger models. In the absolute ideal scenario, a prediction on a particle-by-particle basis would be modeled, completely reproducing the output from a real simulation with no lost information. Given the enormous number of particles in a high-resolution simulation, this is likely a goal for the very far future, which will require clever architectures or optimization techniques to become possible.

CHAPTER 5

Appendix I: A Jupyter Notebook Tutorial

This Jupyter Notebook was originally authored using Google Colab, and was later formatted for download to a personal computer. It can be found in its full, original version in both formats at https://github.com/apetulante/UNet_Tutorial. Some minor formatting changes have been made for its inclusion here.

This is a Jupyter notebook tutorial, walking through how a UNet convolutional neural network works. It has been written assuming that the reader has some prior knowledge about neural networks and/or convolutional networks, but that they may have forgotten most of it and need a refresher of the basics.

5.1 Pre-Tutorial Setup

Before we start, let's check that the GPU is ready to go if we have one, and import packages that we'll need, and talk about the motivations behind using a U-Net.

5.1.1 Some Basic Setup

If you wish to re-run any cells in this notebook, and you're on Google Colab with GPU access, run this cell. Otherwise, this entire notebook should run fine on a CPU (but will be faster on a GPU).

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

Now, we'll import the rest of the packages we'll need. If re-running the cells that load images, we'll also define a string which is just the path to the folder that contains this notebook as well as all of the other images/data needed to run it (should be in folders as they are on github).

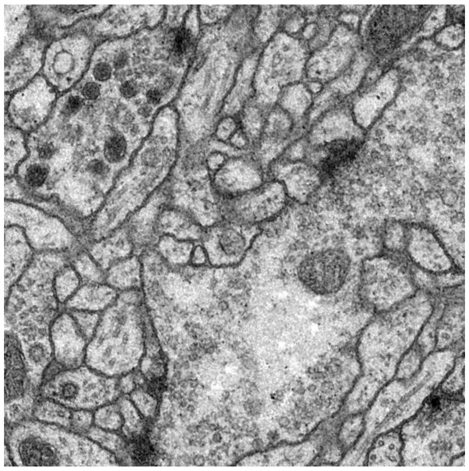
```
!pip install -q keras
import numpy as np
import os
import matplotlib.pyplot as plt
import matplotlib
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras import backend as K
import tensorflow as tf

from PIL import Image

from matplotlib import animation
from IPython.display import display, HTML
from IPython.display import Image as Im
```

5.1.2 What is a UNet?

This subsection makes use of data and results originally presented in https://github.com/zhixuhao/unet

You might be wondering: what exactly is a UNet? Why would I want to use one? How is it different from other convolutional neural nets? So we'll start by giving a bit of motivation for why UNets are so useful. Let's say we have some image:

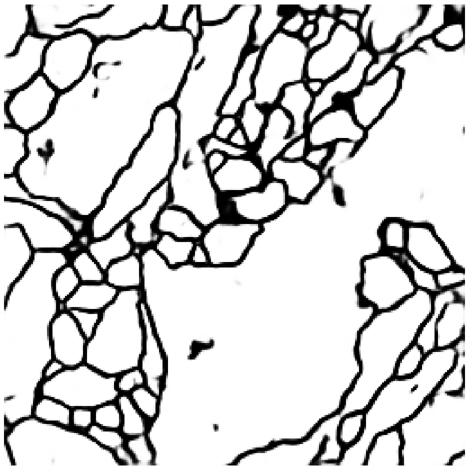
```
display(Im('%s/data/test_input.png' %filepath, height=270, width=270))
```



A ”normal” CNN task might be to say ”that’s a stomach” given that the image could have been from a stomach, brain, or skin. (Full disclosure: I don’t know what this image is of, but it’s a medical image of some sort).

Let’s say instead though, that we want this:

```
display(Im('%s/data/test_output.png' %filepath, height=270, width=270))
```



That’s not a single class output: it’s another image. And ”normal” CNN’s don’t give an image from an image, they collapse an image into one number or one set of numbers. Enter the UNet. Fundamentally, it’s a CNN that’s architecture is such that you get an image back out of the same size as the input image.

One way to think about this is really as pixel-by-pixel classification: in the above example, we're deciding whether each pixel should be assigned black or white. But as you'll see, a UNet is more generalizable than that, and the final image doesn't necessarily need to correspond to pixel classifications.

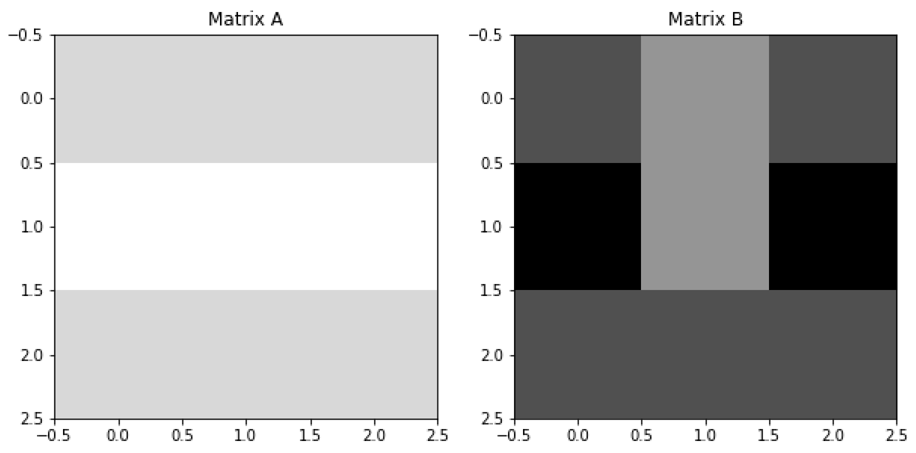
5.2 How Convolutions Work

UNets are a type of convolutional neural network (CNN), so understanding how convolutions work is fundamental to understanding how these networks work. In this section, we briefly go over how to perform convolutions and the building blocks of a convolutional layer in a CNN. While I cover all the basics here, this is meant as more of a refresher and assumes you have previously seen how a convolutional network works (but may have forgotten the details since).

5.2.1 The Convolution Operation

First, we need to cover exactly what a "convolution" means. The building blocks of convolutions are essentially dot products over matricies - we multiply the values in a matrix by the corresponding values in another matrix, then add the values together to get a number. Let's define matricies A and B, then take the dot product between them.

```python
a = np.array([[1,1,1],[0,0,0],[1,1,1]])
b = np.array([[3,2,3],[4,2,4],[3,3,3]])

# to visualize the matricies
fig = plt.figure(figsize=(10,5))
ax1, ax2 = fig.add_subplot(1,2,1), fig.add_subplot(1,2,2)
ax1.imshow(a, vmin=0, vmax=4, cmap="Greys"), ax1.set_title("Matrix A")
ax2.imshow(b, vmin=0, vmax=4, cmap="Greys"), ax2.set_title("Matrix B")

print(np.sum(a*b)) # since a/b are arrays, a*b is element-wise
    multiplication
```



Note that, this is different than doing matrix multiplication, which would result in another matrix, rather than just one number.

Really, we want to think of this operation as giving us some linear combination of a matrix. If we want a linear combination of matrix X with values *x1,x2,x3, ... x9*, we can define some matrix A with values *a,b,c, ... i*, such that:

$$y = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} = \begin{aligned} &(a \times x_1) + (b \times x_2) + (c \times x_3) + (d \times x_4) + (e \times x_5) + \\ &(f \times x_6) + (g \times x_7) + (h \times x_8) + (i \times x_9) \end{aligned}$$

Then, we can interpret the values in matrix A *(a,b,c, ... i)* as **weights**, each of which decides how strong the contribution from matrix X's values *(x1,x2,x3, ... x9)* should be.

5.2.2 Convolving an Image

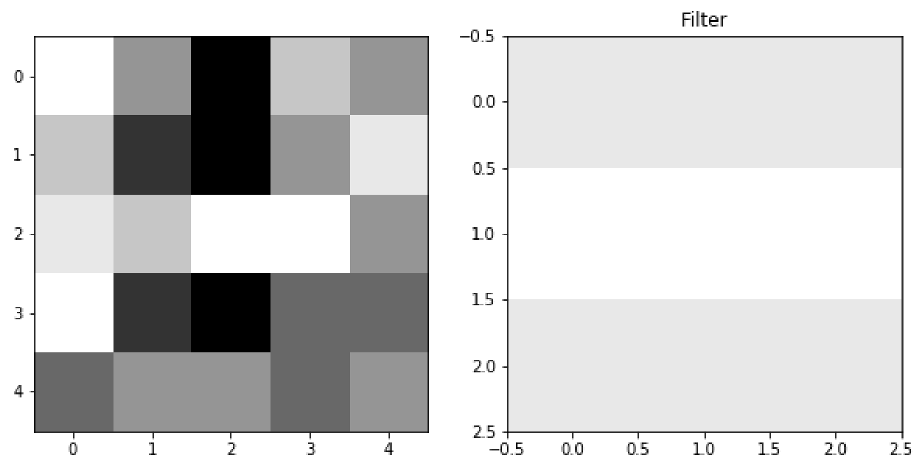
When we convolve an image, we simply perform this operation over and over again, on each pixel of an image. So, matrix A would be our matrix of weights, and matrix X is a matrix of pixel values, where the center value is our current pixel of interest. Convolving an image means performing this operation on every pixel of the image, then replacing its value with the one that is given by our matrix dot product. In this way, the image becomes another version of itself - one where each pixel is some linear combination of the pixels that were around it:

$$x_{5,new} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} = \begin{array}{r} (a\times x_1)+(b\times x_2)+(c\times x_3)+(d\times x_4)+(e\times x_5)+ \\ (f\times x_6)+(g\times x_7)+(h\times x_8)+(i\times x_9) \end{array}$$

In convolutional network applications, we typically call this matrix of weights a **filter**. In other applications that use convolutions, it may also be called a **kernel**.

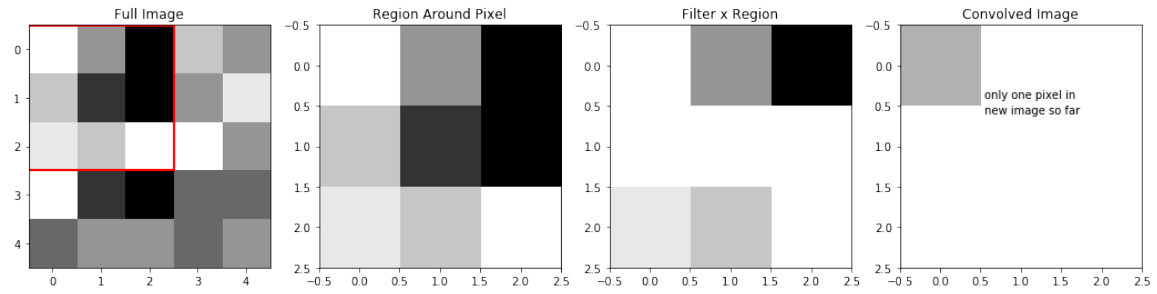
Let's use matrix A from before, this time to convolve a simple, 5x5 image. The image we are going to convolve is shown below:

```
im = np.array([[0,3,6,2,3],
               [2,5,6,3,1],
               [1,2,0,0,3],
               [0,5,6,4,4],
               [4,3,3,4,3]])

# show the image we'll convolve, and the filter we'll convolve it with
fig = plt.figure(figsize=(10,5))
ax1, ax2 = fig.add_subplot(1,2,1), fig.add_subplot(1,2,2)
ax1.imshow(im, cmap="Greys"), plt.title("Starting Image")
ax2.imshow(a, cmap="Greys",vmin=0,vmax=6), plt.title("Filter")

conved_im = np.zeros((3,3)) # we'll replace these as we get the new
    values
```



The first step is to take the 3x3 block in the upper left of our image, and multiply that by our weights:

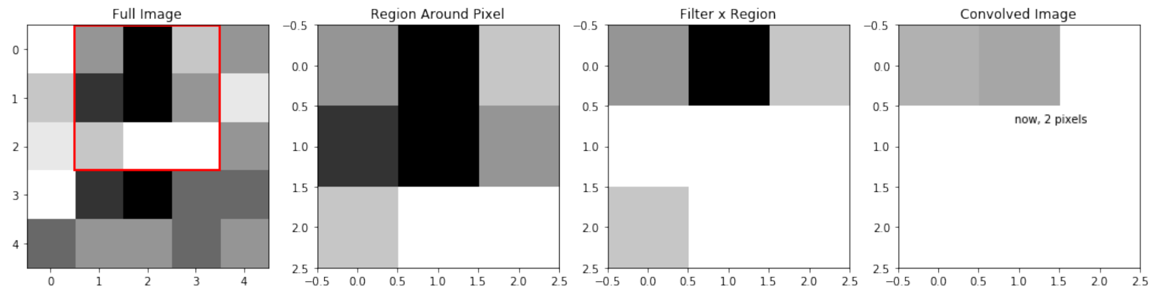
```
fig = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4 = fig.subplots(1,4)
ax1.imshow(im, cmap = "Greys"), ax1.set_title("Full Image")
ax1.add_patch(matplotlib.patches.Rectangle((-.48,-.48),2.98,2.98,fill=
    False,color='red',lw=2)) #show region to convolve

ax2.imshow(im[0:3,0:3],cmap="Greys"), ax2.set_title("Region Around Pixel
    ")
ax3.imshow(im[0:3,0:3]*a,cmap="Greys",vmin=0,vmax=6), ax3.set_title("
    Filter x Region")

conved_im[0][0] = np.sum(im[0:3,0:3]*a)
ax4.imshow(conved_im,cmap="Greys",vmin=0,vmax=29), ax4.set_title("
    Convolved Image")
     #here, I prematurely set vmax to what the maximum of conved_im will
    be, otherwise scaling will change as it plots
plt.annotate("only one pixel in", (.55,.4))
plt.annotate("new image so far", (.55,.6))
```



We can see, that because our filter was a row of ones, a row of zeros, then another row of ones, when we apply this filter to our region, the middle row becomes zeros, while the top and bottom rows of the region are unchanged. So, the sum of Filter X Region which creates our new pixel is really just the sum of the top and bottom rows of the region.

Next, let's move over one pixel, and do the same thing:

```
fig = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4 = fig.subplots(1,4)
ax1.imshow(im, cmap = "Greys"), ax1.set_title("Full Image")
ax1.add_patch(matplotlib.patches.Rectangle((.5,-.48),3,2.98,fill=False,
    color='red',lw=2)) #show region to convolve

ax2.imshow(im[0:3,1:4],cmap="Greys", vmin=0, vmax=6), ax2.set_title("
    Region Around Pixel")
ax3.imshow(im[0:3,1:4]*a,cmap="Greys",vmin=0,vmax=6), ax3.set_title("
    Filter x Region")

conved_im[0][1] = np.sum(im[0:3,1:4]*a)
ax4.imshow(conved_im,cmap="Greys",vmin=0,vmax=29), ax4.set_title("
    Convolved Image")
     #here, I prematurely set vmax to what the maximum of conved_im will
    be, otherwise scaling will change as it plots
plt.annotate("now, 2 pixels", (.95,.7))
```

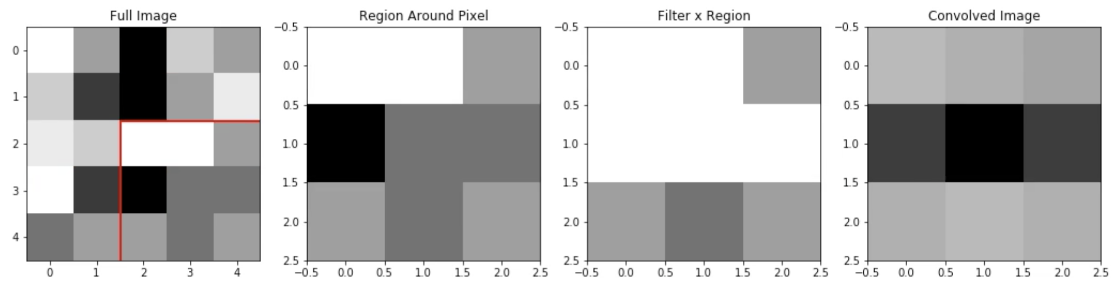


We can keep moving it over, and filling in the pixels of this ”new version” of our image:

```
fig = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4 = fig.subplots(1,4)

display_ims = []
conved_im = np.zeros((3,3)) # reset this
for i in range(conved_im.shape[0]):
  for j in range(conved_im.shape[1]):
    im1 = ax1.imshow(im, cmap = "Greys",animated=True)
    ax1.set_title("Full Image")
    im1 = ax1.add_patch(matplotlib.patches.Rectangle((-.48+j,-.48+i)
   ,3,3,fill=False,color='red',lw=2)) #show region to convolve

    im2 = ax2.imshow(im[i:i+3,j:j+3],cmap="Greys", vmin=0, vmax=6,
   animated=True)
    ax2.set_title("Region Around Pixel")
    im3 = ax3.imshow(im[i:i+3,j:j+3]*a,cmap="Greys",vmin=0,vmax=6,
   animated=True)
    ax3.set_title("Filter x Region")

    conved_im[i][j] = np.sum(im[i:i+3,j:j+3]*a)
    im4 = ax4.imshow(conved_im,cmap="Greys",vmin=0,vmax=29, animated=
   True)
         #here, I prematurely set vmax to what the maximum of conved_im
   will be, otherwise scaling will change as it plots
```

```
    ax4.set_title("Convolved Image")

    display_ims.append([im1, im2, im3, im4])

ani = animation.ArtistAnimation(fig, display_ims, interval=1000, blit=
    True, repeat_delay=1000)
plt.close()

HTML(ani.to_html5_video())
```

This image originally appeared in the Jupyter Notebook as a video. The image presented here is the final frame of that video. The full video can be viewed in the original Jupyter Notebook by going to https://github.com/apetulante/UNet$_T$utorial.



5.2.3 Padding

You’ll notice that this convolution reduced our image size: while we started with a 5x5 image, our convolved version was only 3x3. This is because we’re only able to fit a 3x3 filter onto a 5x5 image, 3x3 times. We aren’t able to make ”new” pixels out of the ones on the border of the image - our filter can’t fit. For this reason, we usually **pad** images before we convolve them - or add values all around the border of the image. Padding ensures two important things:

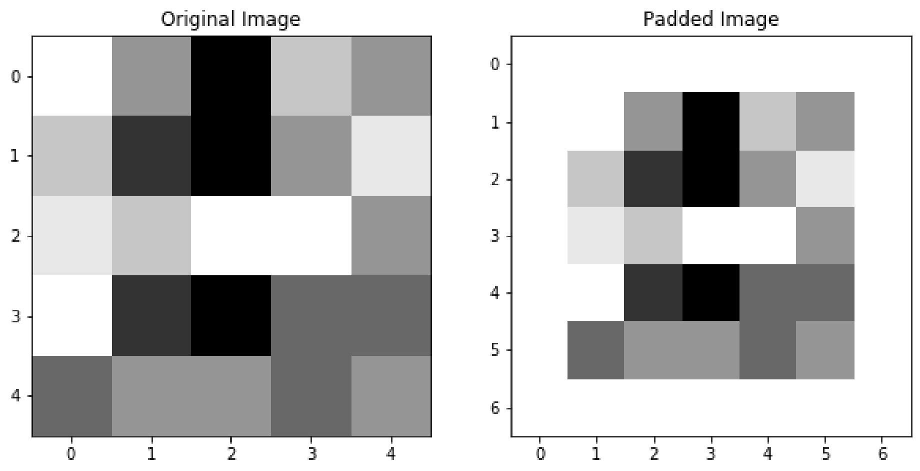
1. That the image isn’t downsized by a convolution.

2. That pixels on the outer edges ”count” as much as pixels in the middle. That is - that they’re convolved over as many times as pixels closer to the center of the image.

There are many different choices for padding, each with their own unique advantages, but the most common/universal (and the only one we’ll discuss here) is **valid**, **zero** padding. **Valid** means that we add whatever padding we need in order to keep the image the same size. **Zero** just means that the values we add along the borders are all zeros.

If we valid zero-pad the image we were just using, we would get:

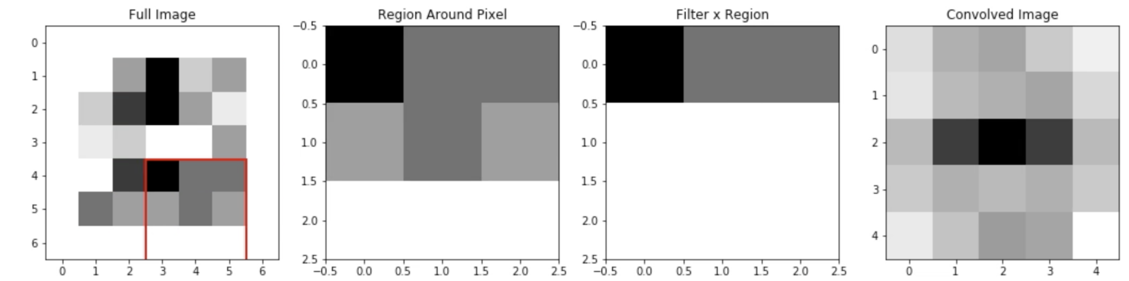
```
padded_im = np.pad(im, pad_width = (1,1), mode="constant",
    constant_values=0)

# show the image we'll convolve, and the filter we'll convolve it with
fig = plt.figure(figsize=(10,5))
ax1, ax2 = fig.add_subplot(1,2,1), fig.add_subplot(1,2,2)
ax1.imshow(im, cmap="Greys"), ax1.set_title("Original Image")
ax2.imshow(padded_im, cmap="Greys",vmin=0,vmax=6), ax2.set_title("Padded
    Image")
```



We can see, if we re-do the convolution we did to this image in the last section, now with the padded image, that pixels on the edge of the image are also convolved over, and the image size is preserved:

```
fig = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4 = fig.subplots(1,4)

display_ims = []
conved_im = np.zeros((5,5)) # now, we'll have 5x5 pixels to fill in
for i in range(conved_im.shape[0]):
  for j in range(conved_im.shape[1]):
    im1 = ax1.imshow(padded_im, cmap = "Greys",animated=True)
    ax1.set_title("Full Image")
    im1 = ax1.add_patch(matplotlib.patches.Rectangle((-.48+j,-.48+i)
    ,3,3,fill=False,color='red',lw=2)) #show region to convolve

    im2 = ax2.imshow(padded_im[i:i+3,j:j+3],cmap="Greys", vmin=0, vmax
    =6, animated=True)
    ax2.set_title("Region Around Pixel")
    im3 = ax3.imshow(padded_im[i:i+3,j:j+3]*a,cmap="Greys",vmin=0,vmax
    =6, animated=True)
    ax3.set_title("Filter x Region")

    conved_im[i][j] = np.sum(padded_im[i:i+3,j:j+3]*a)
    im4 = ax4.imshow(conved_im,cmap="Greys",vmin=0,vmax=29, animated=
    True)
        #here, I prematurely set vmax to what the maximum of conved_im
    will be, otherwise scaling will change as it plots
    ax4.set_title("Convolved Image")

    display_ims.append([im1, im2, im3, im4])

ani = animation.ArtistAnimation(fig, display_ims, interval=1000, blit=
    True, repeat_delay=1000)
plt.close()

HTML(ani.to_html5_video())
```

This image originally appeared in the Jupyter Notebook as a video. The image presented here is the final frame of that video. The full video can be viewed in the original Jupyter Notebook by going to https://github.com/apetulante/UNet$_T$utorial.



Valid padding means padding in order to maintain image size. In our example above, this means we just had to add a border of single-pixel width to our image. In general though, the size of the border you need to add will depend on a few parameters. The parameters that determine the size after you perform a convolution are:

- **n**: the size of the input image (assumed square, so it's nxn)
- **f**: the size of the filter (assumed square, so it's fxf)
- **s**: the **stride**, or how much you move the filter over before you do the next convolution. In our above examples, we've always used s=1, but in general, s can be any number that will still make the filter fit evenly inside the image. We will assume that you use the same stride along all of the image dimensions.
- **p**: the width of the padding to be added, assumed the same amount will be added all around the image.

Then, the output dimension of the image will be:

$$n_{out} \times n_{out} = \frac{n - f + 2p}{s} + 1$$

So, depending on the image size, filter size, and stride selected, you can determine the width of the padding that will need to be added to keep $n_{out} = n$.

5.2.4 Adding Bias and Activations

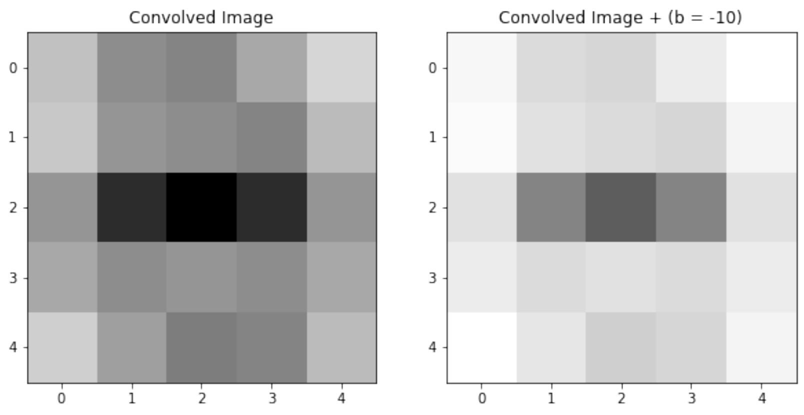
We've already talked about how we can intepret each step of a convolution as replacing a pixel with a linear combination of it and the pixels around it. But a classic linear combination has the format:

$$y = m_1x_1 + m_2x_2 + m_3x_3 + ... + b$$

And so far, we haven't added b. We refer to this extra parameter as the **bias**, it's a single value that get's added to every pixel of the image after the image has been convolved. You need a bias for sort of the same reason that you need b when you fit a line: because it can act to shift the entire image one way, in a way that otherwise can be impossible given just the values $\times$ weights.

Beacuse the bias is just a single number added to every pixel, it's a very simple augmentation of the image:

```
bias = -10
fig = plt.figure(figsize=(10,5))
ax1, ax2 = fig.add_subplot(1,2,1), fig.add_subplot(1,2,2)
ax1.imshow(conved_im, cmap="Greys", vmin=-5, vmax=29), ax1.set_title("
    Convolved Image")
biased_im = conved_im + bias
ax2.imshow(biased_im, cmap="Greys", vmin=-5, vmax=29), ax2.set_title("
    Convolved Image + (b = %d)" %bias)
```

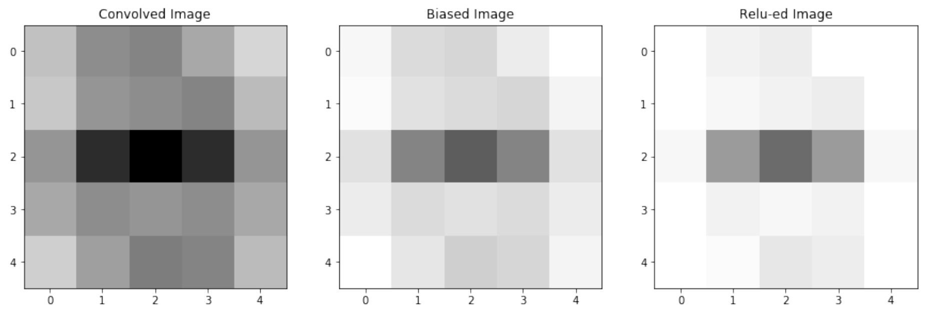


The other step that happens after the convolution step in a convolutional layer is the activation. In the activation step, the image is subject to a function, so each pixel of the image is changed according to that function. In convolutional neural networks, the most common of these functions is ReLU (Rectified Linear Units), which looks like:

$$ReLU(x) = max(0, x)$$

So, when an image is passed through the ReLU activation, each pixel becomes either 0 (if the value was negative) or remains the same (if the value was 0 or positive). An image passed through this activation will look like:

```
fig = plt.figure(figsize=(15,5))
ax1, ax2, ax3 = fig.add_subplot(1,3,1), fig.add_subplot(1,3,2), fig.
    add_subplot(1,3,3)
ax1.imshow(conved_im, cmap="Greys", vmin=-5, vmax=29), ax1.set_title("
    Convolved Image")
ax2.imshow(biased_im, cmap="Greys", vmin=-5, vmax=29), ax2.set_title("
    Biased Image")
relu_im = np.maximum(0,biased_im)
ax3.imshow(relu_im, cmap="Greys", vmin=0, vmax=29), ax3.set_title("Relu-
    ed Image")
```



When we added the bias to our image, some of our pixels became negative. That means that, after we applied the activation function, these pixels actually became zero-valued, meaning our final image now has some areas of whitespace that weren't there before.

It may seem as though this activation function merely removes information: pixels that previously had value are now becoming zeroed-out, now lending us no information about the image. We won't go into a detailed explanation as to why activation functions are so important (as well as an explanation of the advantages and disadvantages of different choices for the activation function), but there are many resources online that do a deep-dive into this topic. For now, I'll just give the main reasons why we include the activation step:

- Dying gradients
- Prevent weights from blowing up

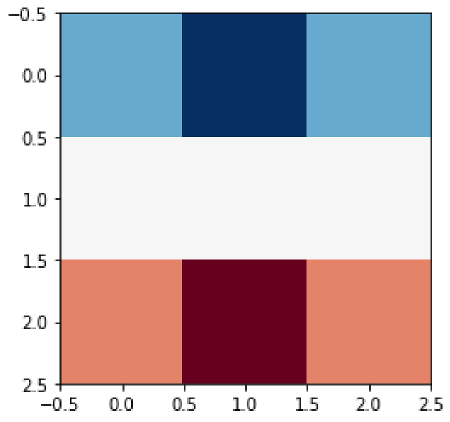
5.3 Interpreting Filters

The point, really, of a UNet, is to learn the weights of the filters, and the biases, that transform an image and allow us to augment that image into another image. So, we may want to attempt to look at the filters and determine the ways that it might be transforming our image and helping to learn patterns.

5.3.1 The Horizontal Edge Detector

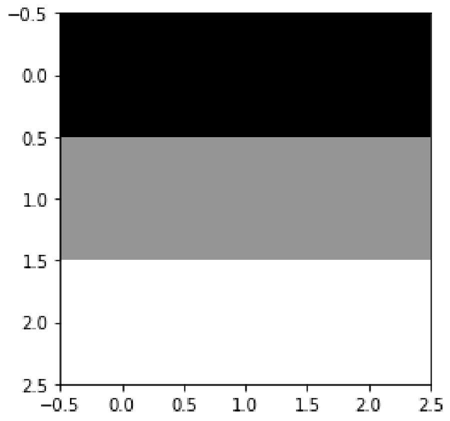
Some filters, such as the *horizontal edge detector* are fairly easily intepretable in the ways that they transform an image. We'll take a look at the horizontal edge detector below.

```python
horiz_edge_filter = np.array([[ 1,  2,  1],
                              [ 0,  0,  0],
                              [-1, -2, -1]])

plt.imshow(horiz_edge_filter, cmap = 'RdBu')
```



This filter is comprised of: a row of positive values, a row of zero values, and a row of negative values. It may not be immediately obvious how this can pick out horizontal edges, but consider the case of an image with a very simple horizontal edge:

```python
horiz_edge_im = np.array([[ 1, 1, 1],
                          [ 0, 0, 0],
                          [ -1, -1, -1]])

plt.imshow(horiz_edge_im, cmap = 'Greys')
```

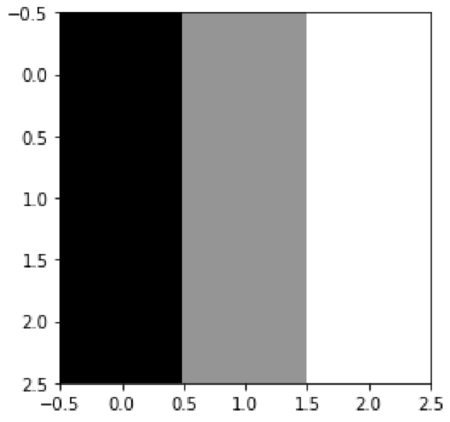


If we convolve this image with this filter (that is, take the sum of the element-wise products of these two 3x3 matricies) we will get a single number:

```
conved_val = np.sum(horiz_edge_filter*horiz_edge_im)
print("The new pixel value would be:", conved_val)
```

The new pixel value would be: 8. If instead, this image had been of a vertical edge. Then, when we check what value the convolution gives us, we instead get:

```
vert_edge_im = np.array([[ 1, 0, -1],
                         [ 1, 0, -1],
                         [ 1, 0, -1]])

plt.imshow(vert_edge_im, cmap = 'Greys')

conved_val = np.sum(horiz_edge_filter*vert_edge_im)
print("The new pixel value would be:", conved_val)
```



The new pixel value would be: 0. We can see that the structure of the filter is that it's symmetric along its horizontal axis. That is, the row of positive values is mirrored by a row of negative values at the bottom of the filter. This means that, any portion of an image which it is applied to, which is also symmetric along its horizontal axis, will give us a value of 0, because the positive and negatives will cancel out.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times \begin{bmatrix} a & b & c \\ a & b & c \\ a & b & c \end{bmatrix} = a + 2b + c + 0 - a - 2b - c = 0$$

Whereas a portion of an image that changes values along its horizontal axis will give us a nonzero value.

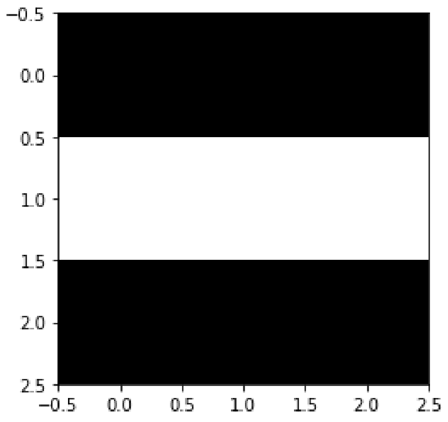
Note that, the values in the center row of the image never matter, because the center row of the filter is all zeros.

5.3.2 The Horizontal Edge Detector in Action

You may notice that an image like the one below, which you would identify as a horizontal line, will not get identified by this filter, because

- It's horizontally symmetric, and
- Every element-wise multiplication includes a zero.

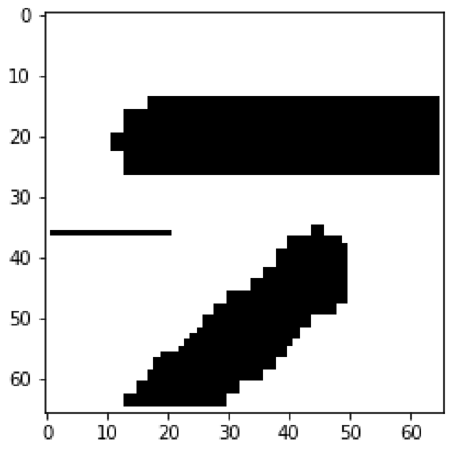
```python
horiz_edge_im = np.array([[ 1, 1, 1],
                          [ 0, 0, 0],
                          [ 1, 1, 1]])

plt.imshow(horiz_edge_im, cmap = 'Greys')
```



But, in practice, we apply these filters over a larger image, not over an image of matching size, so we'll see that single-pixel edges are still detected by this filter, *just not when the edge is on the center pixel*.

Let's start by loading in an image with some edges, which we'll pass our horizontal edge detector over.

```python
from PIL import Image
im = np.array(Image.open('%s/images/horiz_im.png' %filepath))[:,:,2]
im = im/255
im = np.round(im)

im[35][0:20] = 1 #add a single-pixel-width edge, to see if we can detect
    that too
padded_im = np.pad(im, pad_width = (1,1), mode="constant",
    constant_values=0)


plt.imshow(padded_im, cmap="Greys")
```

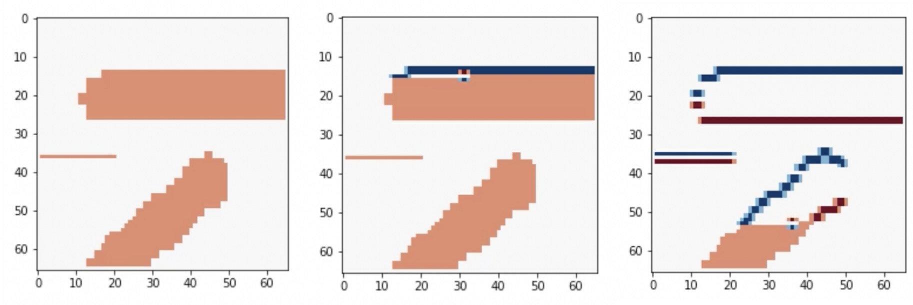


Now, if we convolve with our filter like we did in Part 1:

```python
fig = plt.figure()

# first, get the filter sizes which will help us later
```

```python
filter_sz = horiz_edge_filter.shape[0]
filter_width = int(np.floor(filter_sz/2))

conv_ims = []
conved_im = padded_im.copy() # set it to a copy, that way we can watch the image transform
for i in range(1, im.shape[0]+1): # from 1-65 instead of 0-64, because we added the padding
  for j in range(1,im.shape[0]+1):
    # first, replace the pixel in the image with the convolved one
    conved_region = padded_im[i-filter_width:i+filter_width+1,j-filter_width:j+filter_width+1]*horiz_edge_filter
    conved_im[i,j] = np.sum(conved_region) # replace pixels of the copy with the convolution
    # make an image where the filter is overlayed, too
    filter_im = conved_im.copy()
    filter_im[i-filter_width:i+filter_width+1,j-filter_width:j+filter_width+1] = conved_region
    if (i>12 and i<18 and j>10) or (i>50 and i<55 and j>10) or (i>33 and i<37 and j<20):
    # it takes too long to plot the whole movie, so just do interesting parts
      conv_ims.append([plt.imshow(filter_im, animated=True, cmap = 'RdBu_r',vmin=-2,vmax=2)]) # with filter overlayed
      conv_ims.append([plt.imshow(conved_im, animated=True, cmap = 'RdBu_r',vmin=-2,vmax=2)]) # convolved im result

ani = animation.ArtistAnimation(fig, conv_ims, interval=100, blit=True, repeat_delay=1000)
plt.close()

HTML(ani.to_html5_video())
```

These images originally appeared in the Jupyter Notebook as part of a video. The images presented here are select frames of that video that represent the complete progression. The full video can be viewed in the original Jupyter Notebook by going to https://github.com/apetulante/UNet$_{T}$utorial.



There are a few important things to note about this output image:

1. That the output image contains both blue and red pixels - this filter is able to pick out not only the edge, but the *direction* the edge is going in - that is, higher valued to lower valued pixels or vice versa.

2. That the single-pixel width edge *is* detected by this filter - and is replaced with these two representations of the edge - where the edge created a border of low to high valued pixels, and where it creates a border of high to low valued pixels.

3. The perfectly vertical portions of the thick lines are ignored by the filter - but diagonal regions are still detected.

In A U-Net, we want to learn the filters that can transform our image. To do this, we usually must convolve an image with many different filters, with deeper layers applying filters to versions of the image which have already been convolved. So as you can probably guess, the filters of a real UNet are usually not doing something as simple and interpretable as the horizontal edge detection.

In Section 5.5, we'll talk more about how to view and interpret the filters of a UNet trained on a real-world example, but for now, we'll take a break to establish exactly what the UNet is doing.

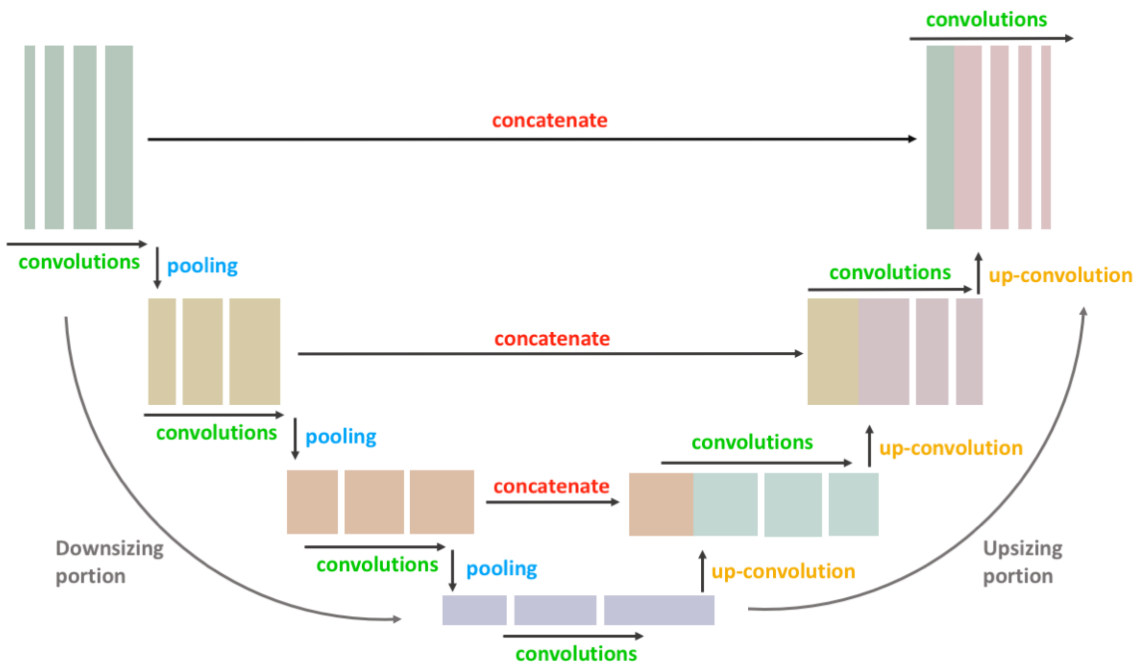
5.4 The Architecture of a UNet

To get a better handle on how exactly these operations work, how they transform our image, and how they change the dimensionality of the image at each step, let's closely investigate an extremely simple example of a UNet.

5.4.1 The UNet Structure

The basic structure of a UNet looks like this:

```
display(Im('%s/images/UNet_Structure.png' %filepath, width=650, height
    =400))
```



U-Nets are named as such because they have this U-like shape, where the input image is first reduced in dimensionality in the downsizing portion, then increased in dimensionality back to its original size in the upsizing portion. As you can see, there are 4 main types of operations, which we'll briefly describe here but give a detailed description of in the upcoming subsections:

1. **convolutions**: We have already discussed the convolution operation and components of a convolutional layer. In these convolutional portions, the image is convolved repeatedly, with differing (and often differently sized) filters.

2. **pooling**: The image is decreased in dimensionality, by representing regions of a few pixels of the image with only one pixel.

3. **up-convolutions**: Sort of the opposite of pooling, one pixel is copied several times to become multiple pixels of the image, in order to increase the image dimensionality.

4. **concatenations**: An image from a previous part of the network is stacked with the image from the current part of the network

5.4.2 Convolutions and Convolution Blocks

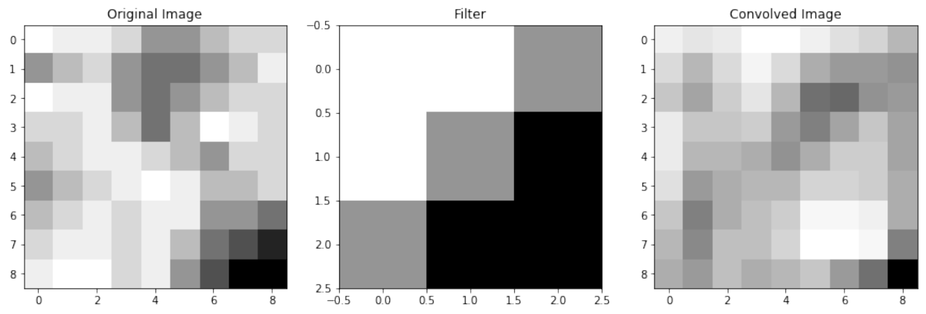
We already talked about convolutions in Section **??**. Here, we'll take a look at exactly what makes a convolutional layer, and how those layers stack to extract the information we want from our image.

You can use a filter to convolve an image. But usually in a convolutional network, we want to use many filters to convolve an image, because each filter is learning something different about the image. Additionally, we usually perform multiple convolutions in a row of the same number of filters/shape of filter, in what's often called a **convolutional block**. Here, we might convolve our image with 5 3x3 filters, then take the result of that convolution, and convolve it with another 5 3x3 filters.

So what exactly is the output of a convolution, and what do we do when we have 5 of them?

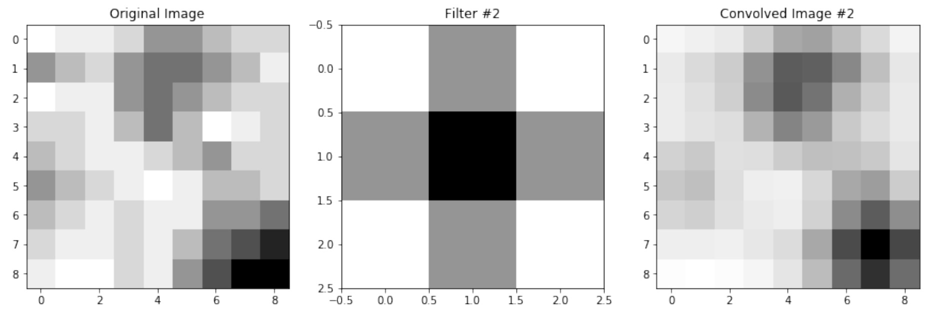
If we start with an image, say 8x8, and we convolve it with a 3x3 filter, then provided that we valid padded it first, we get out a 8x8 image which is some version of the original:

```
im = np.array([[0,1,1,2,4,4,3,2,2],  # the region of pixels
               [4,3,2,4,5,5,4,3,1],
               [0,1,1,4,5,4,3,2,2],
               [2,2,1,3,5,3,0,1,2],
               [3,2,1,1,2,3,4,2,2],
               [4,3,2,1,0,1,3,3,2],
               [3,2,1,2,1,1,4,4,5],
               [2,1,1,2,1,3,5,6,7],
               [1,0,0,2,1,4,6,8,8]])

filt1 = np.array([[-1,-1,0], # a filter I made up
                  [-1,0,1],
                  [0,1,1]])

from scipy.ndimage import convolve    # a handy function that can do
    convolutions for us
conv_im1 = convolve(im, filt1, mode = 'constant')  # set mode=constant
    for valid padding

fig = plt.figure(figsize=(15,5))
ax1, ax2, ax3 = fig.subplots(1,3)
ax1.imshow(im, cmap='Greys'), ax1.set_title('Original Image')
ax2.imshow(filt1, cmap='Greys'), ax2.set_title('Filter')
ax3.imshow(conv_im1, cmap='Greys'), ax3.set_title('Convolved Image')
```



If we have a second filter, then we have another version of the image which was convolved with that filter:

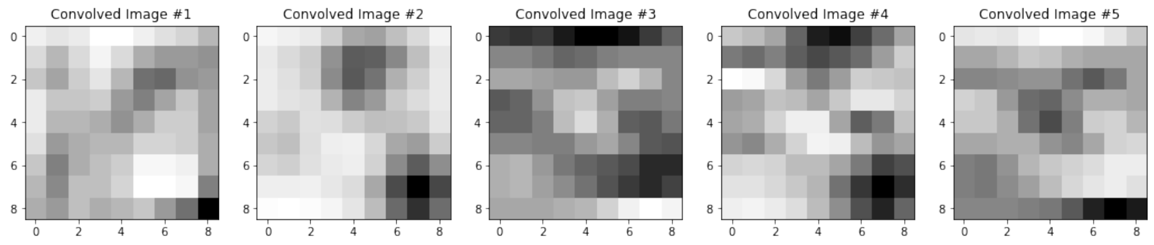
```
filt2 = np.array([[1,2,1],   # another filter I made up
                  [2,3,2],
                  [1,2,1]])

conv_im2 = convolve(im, filt2, mode = 'constant')   # set mode=constant
    for valid padding

fig = plt.figure(figsize=(15,5))
ax1, ax2, ax3 = fig.subplots(1,3)
ax1.imshow(im, cmap='Greys'), ax1.set_title('Original Image')
ax2.imshow(filt2, cmap='Greys'), ax2.set_title('Filter #2')
ax3.imshow(conv_im2, cmap='Greys'), ax3.set_title('Convolved Image #2')
```



If we have 5 such filters, then we have 5 unique ”versions” of the original image:

```
filt3 = np.array([[1,2,1], [0,0,0], [-1,-2,-1]])     # more filter that I
     made up, not necessarily
filt4 = np.array([[0,1,0], [2,3,2], [-1,-2,-1]])     # ones that should
    do anything interesting
filt5 = np.array([[-1,-2,-1], [0,0,0], [1,2,1]])

conv_im3 = convolve(im, filt3, mode = 'constant')
conv_im4 = convolve(im, filt4, mode = 'constant')
conv_im5 = convolve(im, filt5, mode = 'constant')

fig = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4, ax5 = fig.subplots(1,5)
```

```
ax1.imshow(conv_im1, cmap='Greys'), ax1.set_title('Convolved Image #1')
ax2.imshow(conv_im2, cmap='Greys'), ax2.set_title('Convolved Image #2')
ax3.imshow(conv_im3, cmap='Greys'), ax3.set_title('Convolved Image #3')
ax4.imshow(conv_im4, cmap='Greys'), ax4.set_title('Convolved Image #4')
ax5.imshow(conv_im5, cmap='Greys'), ax5.set_title('Convolved Image #5')
```

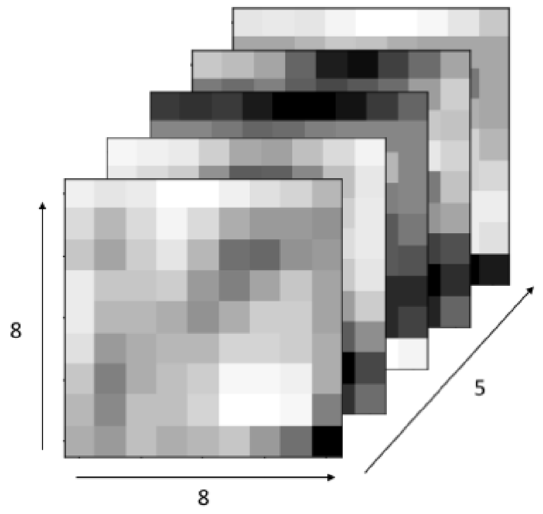


Now, we have 5 representations of our original image, each with some unique features that were emphasized or de-emphasized because of the filter that created them. So, we want a way to keep all of this information that our filters gave us. But, we also want a way to be able to associate these versions of the image with one another. The dark pixels in the bottom right of all of these convolved images above, for example, are all some representation of the dark region in the lower right of our starting image. That is, the bottom right regions of our convolved images still correspond to and give us information about the bottom right region of our original image.

So, what we do is *stack* the images so that each one becomes a **channel** of one complete image. These channels are just like the RGB channels you might be used to in normal color images: each one contains some information about the image, and each matching pixel across different channels is telling you something about the same region of the image. That's exactly what our convolved images are doing - they're each telling us different pieces of information about the same regions of the original image.

When we stack these convolved images into channels, we increase the depth of the image: our **8x8x(1 channel)** original image is now an **8x8x(5 channel)** image.

```
display(Im('%s/images/operation_examples/conv_example_im1.png' %filepath
    , height=370, width=370))
```



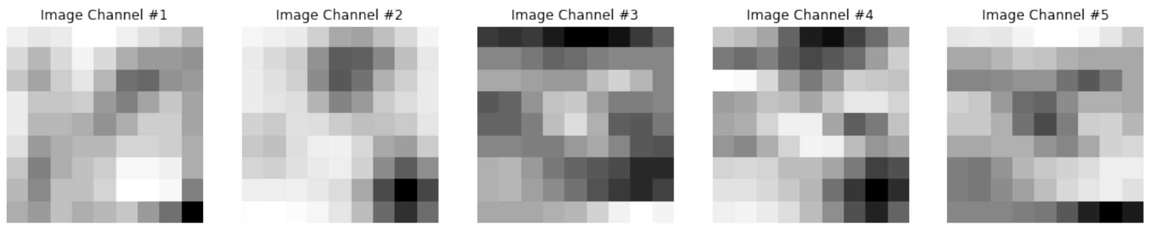
Next, it's typical to convolve our image a second time. Let's say that our convolution block involves a second set of convolutions, where this time we want to use **3, 3x3 filters**.

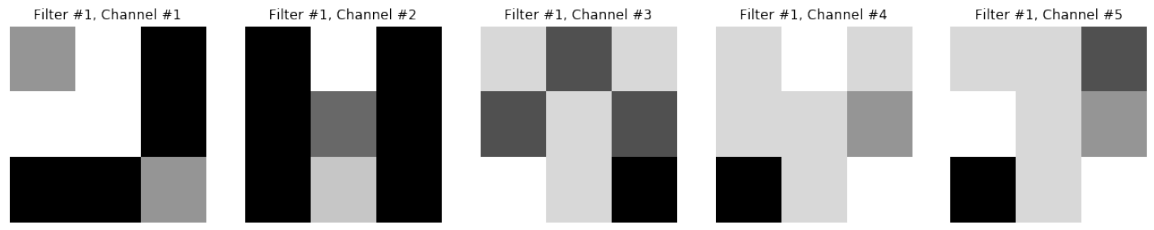
You might be wondering: how are we going to convolve an 8x8**x5** image with a 3x3 filter? The answer is that our filters will now also need to have 5 channels, so really, we'll be using 3, 3x3**x5(channel)** filters. (*Note: I keep making this distinction that these third dimensions are *channels*. That's because it's an important distinction: convolutions can happen in 3D, too, so a 3x3x5 filter (not a 3x3x(5 channel)) filter actually would, in general, be a somewhat different operation, which I'll point out and talk about more in a little bit. This is why we would still refer to the second set of filters in this convolution block as 3x3 filters, instead of specifying that they have 5 channels. The number of channels is implied by the network architecture; if we were to call them 3x3x5 filters it would sound like we are doing 3D convolutions.*)

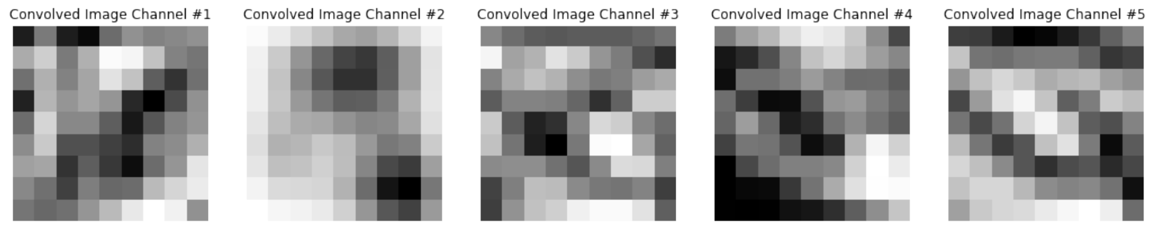
When we convolve a multi-channel image with a multi-channel filter (always with the matching number of channels as the image), what we are effectively doing is convolving each channel of our image with its own filter, and then adding the results togeher. So, in the second part of our convolution block, where we have 3, 3x3x(5 channel) filters, it's really like each filter gives us 5 versions of our image.

Let’s look at what **one** of the filters is doing:

```
fig = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4, ax5 = fig.subplots(1,5)
ax1.imshow(conv_im1, cmap='Greys'), ax1.set_title('Image Channel #1'),
    ax1.axis('off')
ax2.imshow(conv_im2, cmap='Greys'), ax2.set_title('Image Channel #2'),
    ax2.axis('off')
ax3.imshow(conv_im3, cmap='Greys'), ax3.set_title('Image Channel #3'),
    ax3.axis('off')
ax4.imshow(conv_im4, cmap='Greys'), ax4.set_title('Image Channel #4'),
    ax4.axis('off')
ax5.imshow(conv_im5, cmap='Greys'), ax5.set_title('Image Channel #5'),
    ax5.axis('off')

fig2 = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4, ax5 = fig2.subplots(1,5)
# Filter #1, 5 channels, each 3x3
filt1_ch1 = np.array([[0,-1,1],[-1,-1,1],[1,1,0]])
ax1.imshow(filt1_ch1, cmap='Greys'), ax1.axis('off'), ax1.set_title('
    Filter #1, Channel #1')
filt1_ch2 = np.array([[2,-1,2],[2,1,2],[2,0,2]])
ax2.imshow(filt1_ch2, cmap='Greys'), ax2.axis('off'), ax2.set_title('
    Filter #1, Channel #2')
filt1_ch3 = np.array([[-1,1,-1],[1,-1,1],[-2,-1,2]])
ax3.imshow(filt1_ch3, cmap='Greys'), ax3.axis('off'), ax3.set_title('
    Filter #1, Channel #3')
filt1_ch4 = np.array([[-1,-2,-1],[-1,-1,0],[2,-1,-2]])
ax4.imshow(filt1_ch4, cmap='Greys'), ax4.axis('off'), ax4.set_title('
    Filter #1, Channel #4')
filt1_ch5 = np.array([[-1,-1,1],[-2,-1,0],[2,-1,-2]])
ax5.imshow(filt1_ch5, cmap='Greys'), ax5.axis('off'), ax5.set_title('
    Filter #1, Channel #5')

fig3 = plt.figure(figsize=(18,5))
ax1, ax2, ax3, ax4, ax5 = fig3.subplots(1,5)
# Convolve each channel of the 8x8x5 image with the corresponding filter
     channel
filt1_ch1_convIm = convolve(conv_im1, filt1_ch1, mode = 'constant')
ax1.imshow(filt1_ch1_convIm, cmap='Greys'), ax1.axis('off'), ax1.
    set_title('Convolved Image Channel #1')
filt1_ch2_convIm = convolve(conv_im2, filt1_ch2, mode = 'constant')
ax2.imshow(filt1_ch2_convIm, cmap='Greys'), ax2.axis('off'), ax2.
    set_title('Convolved Image Channel #2')
filt1_ch3_convIm = convolve(conv_im3, filt1_ch3, mode = 'constant')
ax3.imshow(filt1_ch3_convIm, cmap='Greys'), ax3.axis('off'), ax3.
    set_title('Convolved Image Channel #3')
filt1_ch4_convIm = convolve(conv_im4, filt1_ch4, mode = 'constant')
ax4.imshow(filt1_ch4_convIm, cmap='Greys'), ax4.axis('off'), ax4.
    set_title('Convolved Image Channel #4')
filt1_ch5_convIm = convolve(conv_im5, filt1_ch5, mode = 'constant')
ax5.imshow(filt1_ch5_convIm, cmap='Greys'), ax5.axis('off'), ax5.
    set_title('Convolved Image Channel #5')
```





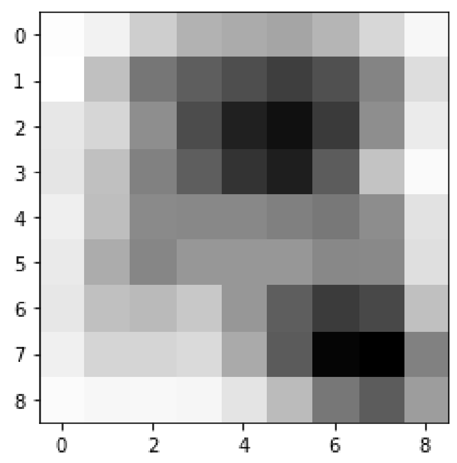

Convolved Image Channel #1
Convolved Image Channel #2
Convolved Image Channel #3
Convolved Image Channel #4
Convolved Image Channel #5

So, we start with a 5-channel image, we convolve each channel with the corresponding filter of a 5-channel filter, and we get out 5 images.

If we have 3 of such filters, that would give us 3 (filters) x 5(channels) = 15 versions of our image. You might expect that we would stack all of these again, and end up with a 8x8x15 output from this convolution, *but that's not the case*.

Although we *stack* the outputs of our filters into channels, we actually *add* the channels of an image after it's convolved. So really, the output of our first convolution, using Filter 1, a 3x3x(5 channel) filter on our 8x8x(5 channel) image is:

```
filt1_convOutput = filt1_ch1_convIm + filt1_ch2_convIm +
    filt1_ch3_convIm + filt1_ch4_convIm + filt1_ch5_convIm

plt.imshow(filt1_convOutput, cmap="Greys")
```



So, actually, if the second set of convolutions in our convolutional block had 3, 3x3 filters, the output of the layer would be an 8x8x(3 channel) image. That is, *the number of channels in the output of a convolutional layer is equal to the number of filters used*. Regardless of the number of channels the input to the convolutional layer had, because we add the channels together after applying our filters, we always end up with *one image per filter*.

Okay, now you might be wondering: *why* do we add these multiple channels together, but we stacked the outputs from different filters instead of adding those together. Why don't we do the same thing in both cases? The logic is roughly this: think of the purpose of each filter to be to learn something different about our image. Adding together the outputs from different filters would muddle their information together, so we want to make sure to keep the information preserved by stacking. But multi-channel filters, while they sort of act like multiple filters over multiple images, are truly *one* filter over *one* image. So, if we want those filters to focus on learning one thing about the image, then we want to add the channels together: because the multiple channels should be *working together* to tell us something about the image.

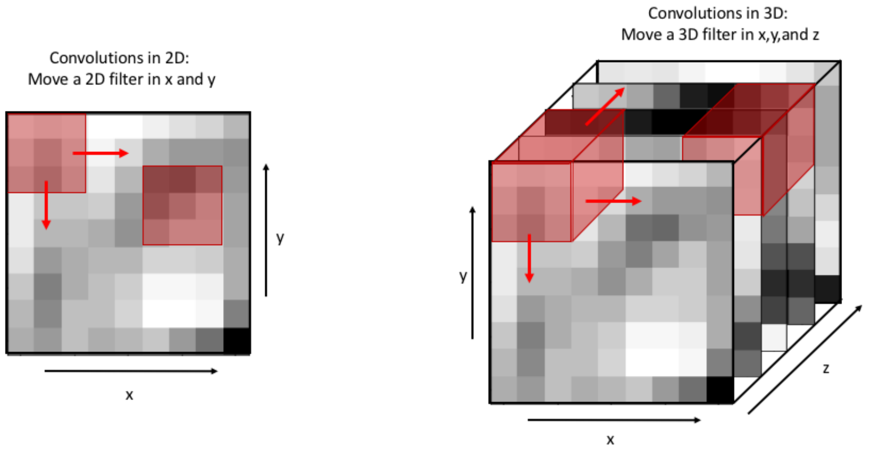
So, in summary:

- Convolutional blocks are typically made up of a few convolutional layers.

- A **covolutional layer** typically involves convolving the image input to the layer with many filters, all of the same size.
- The output of a convolutional layer is an image with multiple **channels** - one per filter.
- If the input to a convolutional layer has multiple channels, the filters used on the image in that layer must all have the same number of channels as the image.
- Each channel of the input image is convolved with a corresponding channel of the filter, to create a corresponding channel of the output.
- The **channels** of an output image convolved with one filter are *added* together to make one image per filter, but the images generated by different filters are *stacked* to create the multiple channels of the output.

Some follow up on that note about 3D convolutions 3D convolutions are convolutions over volumes. A 3x3x3 filter over a volume performs a similar operation as a 3x3 filter over a 2D image, in that the weights of the filter are multipled by a region of the image, then summed together to get one pixel value. The only real difference, is that a 3D filter on a 3D volume also strides over the volume dimension, instead of just across the image in the 2D dimensions.

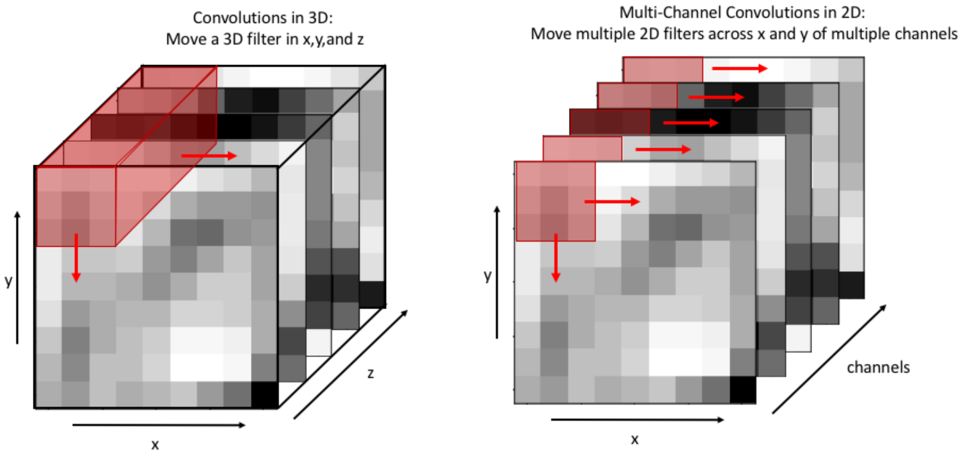
```
display(Im('%s/images/operation_examples/3D_vs_2D_convs.png' %filepath,
    height=400, width=850))
```



Thus, a 3D convolution over a volume will usually also produce a volume: the original image will usually be padded in all dimensions, so that as the filter slides over all dimensions of the image, multiplying the weights and adding them together, the output volume has the same dimensions as an input volume.

In principle, our 3x3x(5 channel) filters are acting the same on an 8x8x(5 channel) image as a 3x3x5 3D filter would act on an 8x8x5 3D volume, in that we are mutiplying the weights by the pixel values and adding the results together to get one pixel value.

```
display(Im('%s/images/operation_examples/3D_vs_2DMultiChannel_convs.png'
    %filepath, height=400, width=850))
```


Convolutions in 3D:
Move a 3D filter in x,y,and z
y
z
x
Multi-Channel Convolutions in 2D:
Move multiple 2D filters across x and y of multiple channels
y
channels
x

But, that doesn't mean that multi-channel convolutions and 3D convolutions are generally the same thing. This only happens because, if the dimension of a filter matches the dimension of an image, the filter can't slide over in that dimension. In the example above, the 3x3x5 3D filter can't slide in the z dimension, so it can only move along x and y just like our multi-channel filter would. But 3D filters will typically be symmetric the way that 2D filters are typically symmetric: a 3D filter would likely be size 3x3x3, instead of 3x3x5, just like a 2D filter is almost always something like 3x3 instead of 3x5. Thus, a 3D convolution will usually be able to slide back along the z dimension of an image, and output a volume.

This is an important distinction because 3D volumes can *also* have multiple channels - in which case there would be multiple 3D filters making up one multi-channel 3D filter, and then thinking about multi-channel convolutions as just 3D convolutions doesn't work anymore.

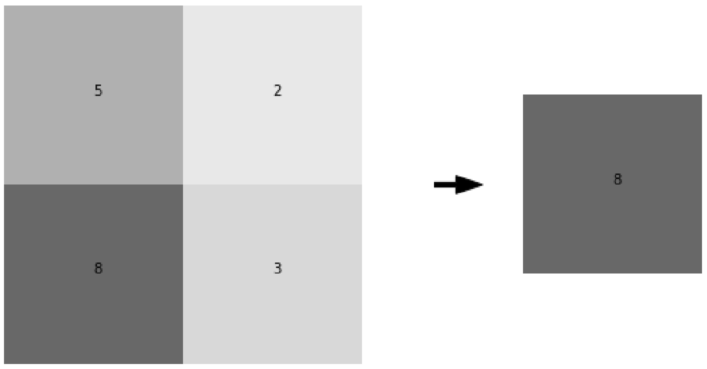
5.4.3 Pooling

The basic idea behind pooling is to reduce the dimensionality of an image, by representing some region of pixels with just one pixel instead.

Max pooling is the most common type of pooling, and the type that we will use in our examples in this tutorial. In max pooling, a region of pixels is represented by the maximum-valued pixel within that region. So, we would represent a region of pixels like this one:

```
im = np.array([[5,2],   # the region of pixels
               [8,3]])
im_max = np.max(im)     # what pooling would give us

fig = plt.figure(figsize=(10,5))
ax1, ax2 = fig.subplots(1,2)
ax1.axis('off'), ax2.axis('off')

# plot the image region with values
```

```
ax1.imshow(im, cmap="Greys", vmin=0, vmax=12)
for j in range(im.shape[0]):
  for i in range(im.shape[1]):
    ax1.annotate(im[i][j], (j,i))

# plot the resulting region
ax2.imshow(np.pad(np.array([im_max]*4).reshape(2,2),pad_width = (1,1),
    mode="constant", constant_values=0), # just makes it look nice
            cmap="Greys", vmin=0, vmax=12)
ax2.annotate(im_max, (1.5,1.5))
ax2.arrow(-1.25,1.5,1,0,width=.05,head_width=.2,color='k') #draw an
    arrow
```



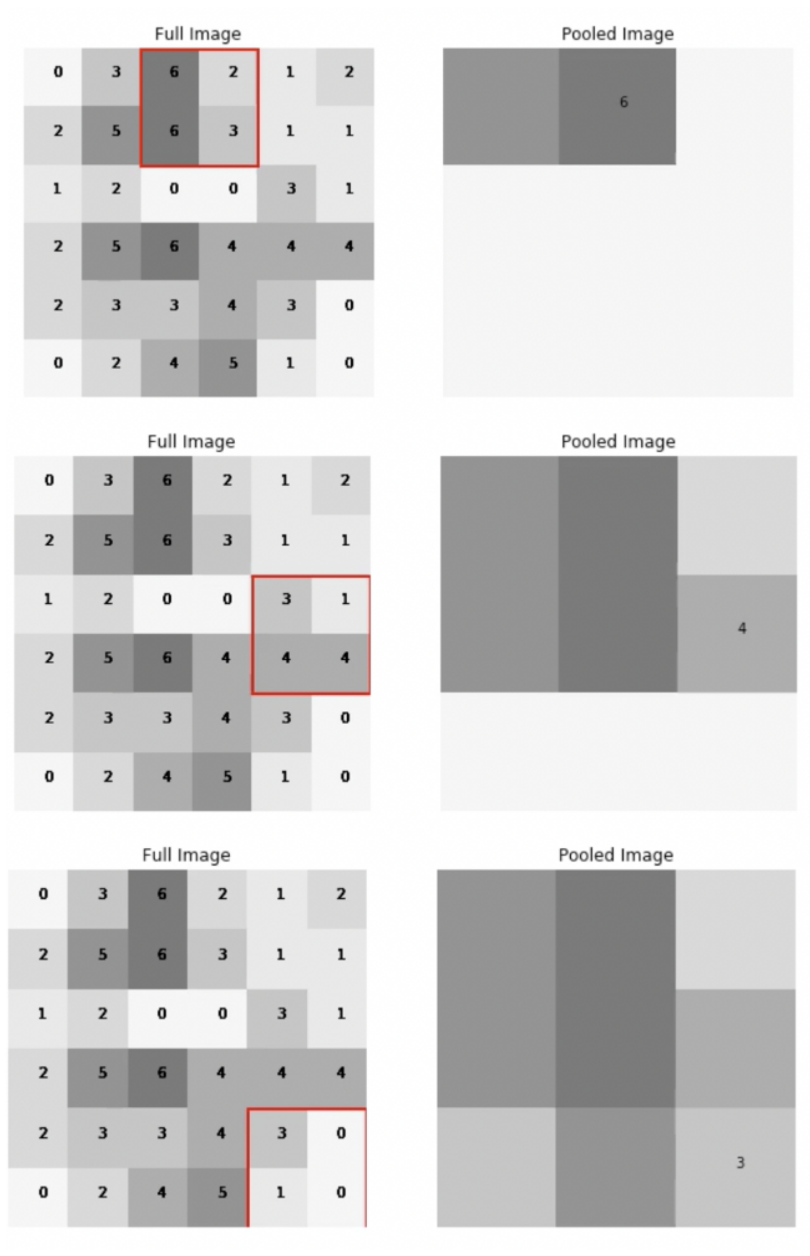
This would be **2x2** max pooling, because the region of pixels that we replace with a single pixel is size 2x2. We can arbitrarily choose the region size that we use for pooling, but this region is almost always square, and 2x2 is a very typical choice.

2x2 max pooling an entire image involves taking every 2x2 region in the image and replacing it like so:

```
im = np.array([[0,3,6,2,1,2],
               [2,5,6,3,1,1],
               [1,2,0,0,3,1],
               [2,5,6,4,4,4],
               [2,3,3,4,3,0],
               [0,2,4,5,1,0]])

fig = plt.figure(figsize=(10,5))
ax1, ax2 = fig.subplots(1,2)
ax1.axis('off'),ax2.axis('off')

display_ims = []
pooled_im = np.zeros((3,3))        # output image will have output shape
    = original shape / 2 for 2x2 pooling
for ind1 in range(pooled_im.shape[0]):
  i = ind1*2          # so that we have an index that moves over by 2
   pixels each time, instead of 1
  for ind2 in range(pooled_im.shape[1]):
    j = ind2*2
    im1 = ax1.imshow(im, cmap="Greys", vmin=-1, vmax=10, animated=True)
    for k in range(im.shape[0]):
      for l in range(im.shape[1]):
        im1 = ax1.annotate(im[k][l], (l,k))    # plot the pixel values
    ax1.set_title("Full Image")
    im1 = ax1.add_patch(matplotlib.patches.Rectangle((-.48+j,-.48+i)
   ,2,2,fill=False,color='red',lw=2)) #show region of pooling

    pooled_im[ind1][ind2] = np.max(im[i:i+2,j:j+2])
    im2 = ax2.imshow(pooled_im,cmap="Greys",vmin=-1,vmax=10, animated=
   True)
    ax2.set_title("Pooled Image")

    display_ims.append([im1, im2, ax2.annotate(int(pooled_im[ind1][ind2
   ]), (ind2,ind1))]) #also show pixel values

ani = animation.ArtistAnimation(fig, display_ims, interval=1000, blit=
   True, repeat_delay=1000)
plt.close()
```

```

HTML(ani.to_html5_video())
```

These images originally appeared in the Jupyter Notebook as part of a video. The images presented here are select frames of that video that represent the complete progression. The full video can be viewed in the original Jupyter Notebook by going to https://github.com/apetulante/UNet$_T$utorial.



We won't discuss any of them in detail here, but there are other types of pooling. **Aver-**

age pooling, for instance, takes the average pixel value of a region as the new pixel value.

You might be wondering: what's the advantage of throwing away information?

1. Computationally, it's advantageous to remove some information, especially if we can still retain the ”most important” information when we do so. In convolutional neural networks in particular, the number of operations we need to perform scales with the size of the image as we convolve it, so reducing the image size can greatly reduce the number of computations we need to do.

2. Pooling may help to ”sharpen” certain features in the image. Because filters are sort of trying to pick out specific features in an image, choosing the pixel that gave the highest ”signal” in a region of an image may help to single out the most important parts of that image.

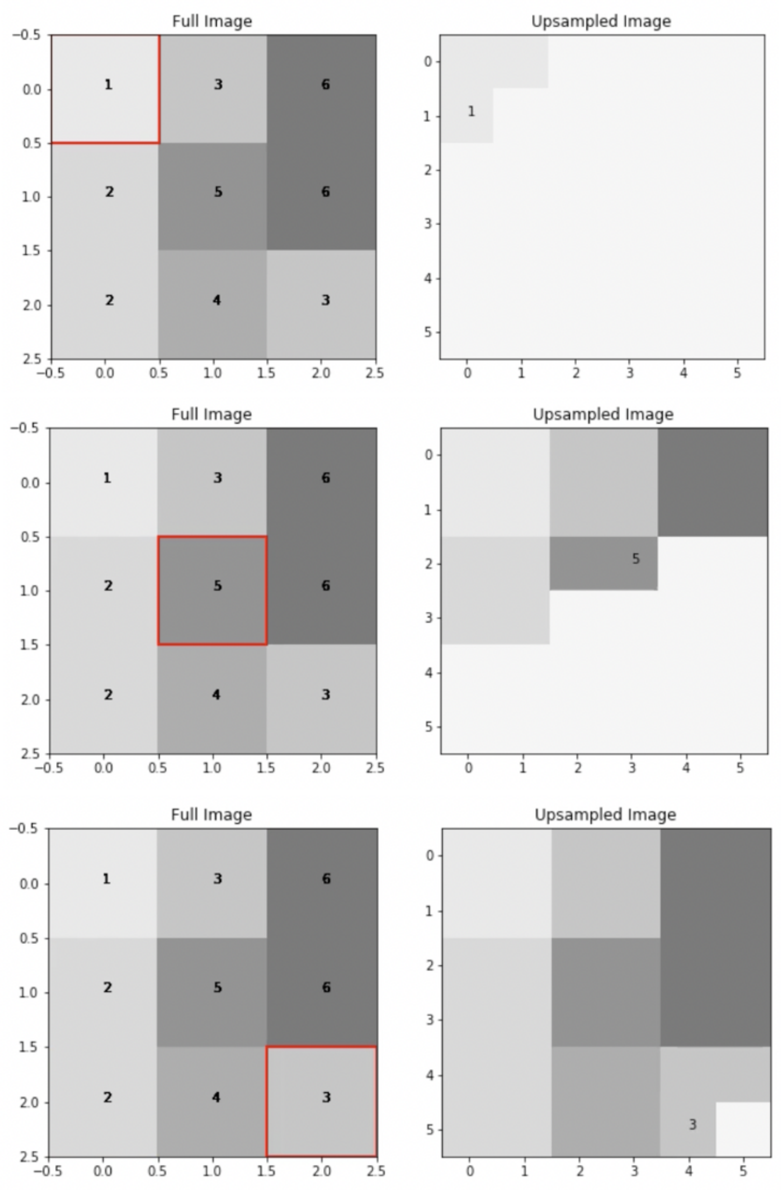
5.4.4 Upsampling

Upsampling is unique to UNets - the step is performed because we need to increase the size of our image after a series of convolutions and pooling has decreased it. In this way, it's like the opposite of pooling - instead of shrinking an image by representing a region of pixels with one pixel, we create a region of pixels by copying one pixel into multiple pixels.

```
im = np.array([[1,3,6],
               [2,5,6],
               [2,4,3]])

fig = plt.figure(figsize=(10,5))
ax1, ax2 = fig.subplots(1,2)

display_ims = []
upsampled_im = np.zeros((6,6))        # output image will have output
   shape = original shape * 2 for 2x2 upsampling
```

```
for i in range(pooled_im.shape[0]):
  ind1 = i*2
  for j in range(pooled_im.shape[1]):
    ind2 = j*2
    im1 = ax1.imshow(im, cmap="Greys", vmin=-1, vmax=10, animated=True)
    for k in range(im.shape[0]):
      for l in range(im.shape[1]):
        im1 = ax1.annotate(im[k][l], (l,k))     # plot the pixel values
    ax1.set_title("Full Image")
    im1 = ax1.add_patch(matplotlib.patches.Rectangle((-.5+j,-.5+i),1,1,
   fill=False,color='red',lw=2)) #show we're upsampling

    for k in range(ind1,ind1+2):
      for l in range(ind2,ind2+2):
        upsampled_im[k][l] = im[i,j]
        im2 = ax2.imshow(upsampled_im,cmap="Greys",vmin=-1,vmax=10,
   animated=True)
        display_ims.append([im1,im2,ax2.text(l,k,int(upsampled_im[k][l])
   )]) # plot the pixel values too
    ax2.set_title("Upsampled Image")

ani = animation.ArtistAnimation(fig, display_ims, interval=600, blit=
   True, repeat_delay=1000)
plt.close()

HTML(ani.to_html5_video())
```

These images originally appeared in the Jupyter Notebook as part of a video. The images presented here are select frames of that video that represent the complete progression. The full video can be viewed in the original Jupyter Notebook by going to https://github.com/apetulante/UNet$_T$utorial.



So, you can see that even though the upsampled image looks identical to the original, it actually has dimensions 6x6 instead of 3x3, and 4 times the number of pixels.

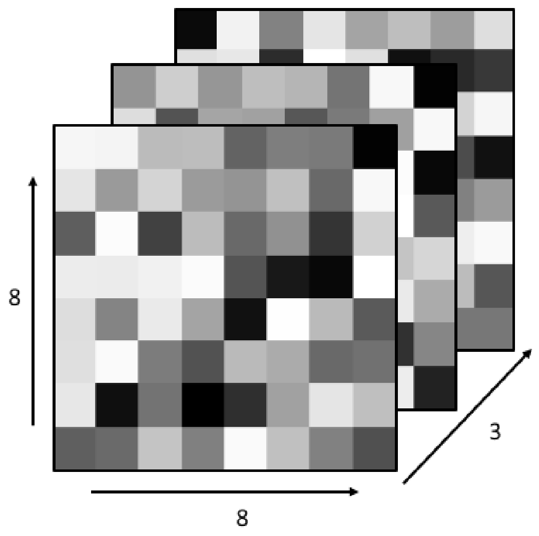
5.4.5 Concatenations

Concatenations are also unique to UNets. As we convolve our image and pool it, we lose the spatial information of our features. If we've reduced the dimensionality of our starting image to 2x2, for example, then each pixel in that 2x2 image represents about a quarter of our initial image, meaning that we've lost all information about finer resolution features within each quarter. If we were to simple upsample and convolve our image back up to its original size, there would be no way to get that information back, because upsampling just copies the same pixels over again - it doesn't increase the resolution of the details. For this reason, we need to do **concatenations**.

In the concatenation step, we take the output of a previous layer in the downsizing portion of the UNet, and stack it with the output from the upsizing portion of the UNet that has the same dimensions. In this way, we get to use the finer resolution information that the downsizing steps still had, but we also get our larger scale information from our upsampling step.

For instance, let's say in the second convolutonal block of a UNet, we convolve our image with 3 filters, so we have an output that looks like this:

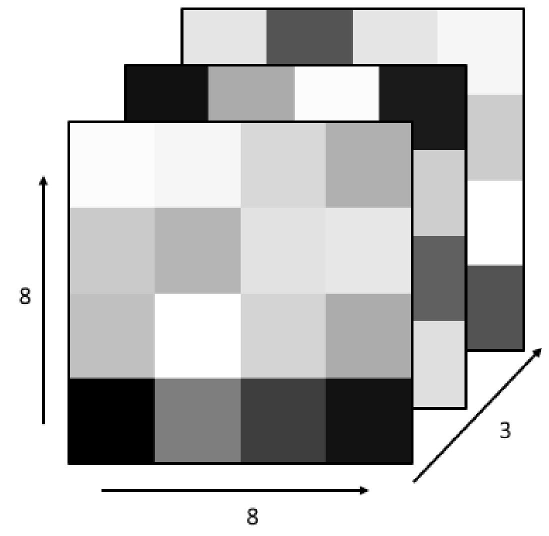
```
display(Im('%s/images/operation_examples/concat_example_im1.png' %
    filepath, height=370, width=370))
```



Then, let's say that in the next steps in the UNet, this image is pooled down to size 4x4x3, and more convolutions are done on the image, keeping it at size 4x4x3 but further transforming it.

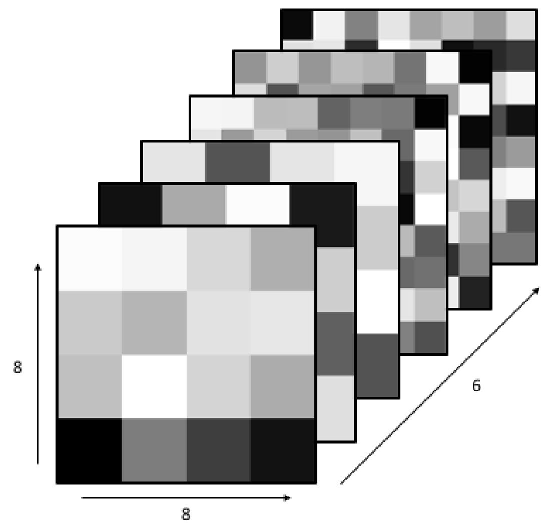
If after these convolutions, we begin the upsizing portion of the UNet, we would begin with an operation which upsamples the 4x4x3 image back into an 8x8x3 image, which looks like:

```
display(Im('%s/images/operation_examples/concat_example_im2.png' %
    filepath, height=370, width=370))
```



In the concatenation step of the UNet, these two 8x8x3 images are stacked, so the image becomes 8x8x6. Then, this stacked image would go on to be convolved further, with the 6 stacked images all acting as different channels of the same image.

```
display(Im('%s/images/operation_examples/concat_example_im3.png' %
    filepath, height=400, width=400))
```



5.5 A Very Simple UNet Example

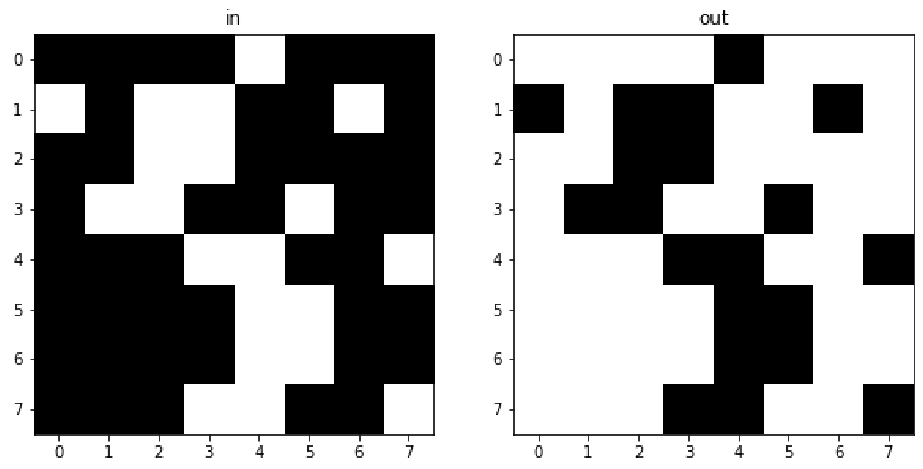
To get a better handle on how exactly these operations work, how they transform our image, and how they change the dimensionality of the image at each step, let's closely investigate an extremely simple example of a UNet.

5.5.1 The Data/Problem

Say we have a simple 8x8 image, made of black and white pixels randomly scattered. And we want to create a UNet to invert the image for us. Our data might look like this:

```
im_in = np.array([[0,0,0,0,1,0,0,0],    # example image in, which we'll
    also use for testing later
              [1,0,1,1,0,0,1,0],        # obviously, not a real random
    scattering, but just an example
              [0,0,1,1,0,0,0,0],
              [0,1,1,0,0,1,0,0],
              [0,0,0,1,1,0,0,1],
              [0,0,0,0,1,1,0,0],
              [0,0,0,0,1,1,0,0],
              [0,0,0,1,1,0,0,1]])

```

```
im_out = np.abs(1-im_in)              # example image output, just the
    inversion of the input image

# show our example images
fig=plt.figure(figsize=(10,5))
ax1,ax2 = fig.subplots(1,2)
ax1.imshow(im_in, cmap="Greys_r")
ax1.set_title("in")
ax2.imshow(im_out, cmap="Greys_r")
ax2.set_title("out")
```



And we can easily generate a dataset of 100 examples:

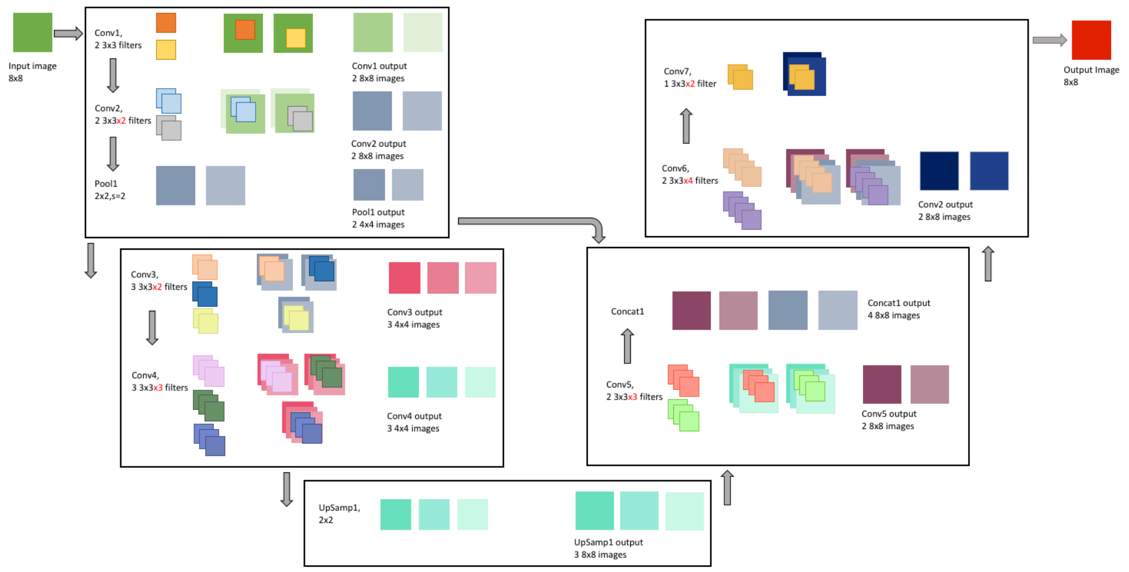
```
X_example = []
y_example = []
for i in range(100):
  X_example.append(np.round(np.random.rand(8,8)).reshape(8,8,1))
  y_example.append(np.abs(1-X_example[-1]).reshape(8,8,1))

X_example = np.array(X_example)
y_example = np.array(y_example)
```

This inversion operation is obviously very simple: It takes one line of code and 2 operations (a subtraction and an absolute value) to perfectly invert our image. But, because this is a transformation of an image, a very simple UNet should also be able to perform this inversion for us, so that's what we'll try to make here.

5.5.2 The Architecture

We'll use a simple UNet, with a few 3x3 filters, to do this inversion. The architecture will look like this:

```python
display(Im('%s/images/simple_example_UFormat.png' %filepath, width=950,
    height=480))
```



This might look a little overwhelming right now, but we're going to go through each of the operations that this network will perform in more detail in the upcoming sections.

We will also add layers to the model in keras as we go through them. To start building a model in keras, we just need to start defining our layers. This begins with the input:

```python
input_size = X_example[0].shape    # get the size of the input images,
    in our case this is 8x8x1
print(input_size)

```

```python
inputs = Input(input_size)          # then, we just define an input layer
    and tell keras to expect images of size 8x8x1
```

5.5.3 Conv Block 1

The first convolution block has 3 steps:

- Conv1: 2 3x3 filter convolutions
- Conv2: 2 3x3 filter convolutions
- Pool1: 2x2 pooling

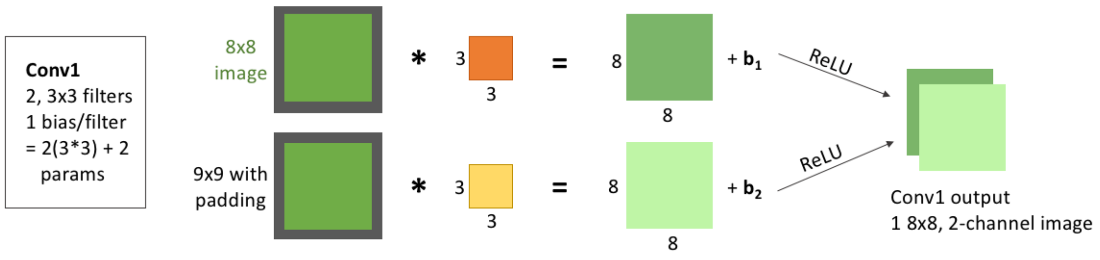
In the first convolution step, the input image is convolved twice: Once with **one 3x3 filter** , and another time with **another 3x3 filter** . We first pad the input image with zeros, so that the convolved image is 9x9, and the result is **2, 8x8 images**, that we then add our bias to and then pass through the ReLu activation function. Each of these images is a ”representation” of the original image. These images are stacked to become two channels of the same image, and the layer output is **1, 8x8x2(channel) image**.

Because we have 2 filters, each with 3x3 weights, and an associated bias for each filter, this means our first layer has a total of:

$$2 \times (3 \times 3) + 2 = 20$$

learnable parameters.

```python
display(Im('%s/images/layers/conv1.png' %filepath, height=270, width
   =1000))
```



We can add this to the model:

```
conv1 = Conv2D(filters = 2,    # here, we tell the layer we want to use 2
     filters
               kernel_size = (3,3),  # the filters are of size 3x3
               activation = 'relu',    # we want to use the ReLU
    activation function
               padding = 'same',      # same padding means the output
    size will equal the input size(before padding)
               kernel_initializer = 'he_normal')(inputs)   # we'll
    initialize the weights with the He normal distribution. We also
                                                          # need to tell
     this layer what the input to it will be, which is the input
                                                          # layer (
    inputs)
```

The next convolution step, takes the output from the first convolution step, and again convolves it twice: Once with **one 3x3x2 filter**, and another time with **another 3x3x2 filter**. Note that these filters now need to have 2 channels, because the output from the first layer had 2 channels. As we discussed in the previous section, when a 2-channel filter convolves a 2-channel image, the outputs are added together to generate the output. That is, the **darker blue** filter convolves the **lighter green** channel of the image, and the **lighter blue** filter convolves the **darker green** image. Then, the two channel outputs are added together to create the **darker purple** image generated from the convolution. The same, of course, happens with the grey filter and the image, generating the **lighter purple**, the second of our two output images.

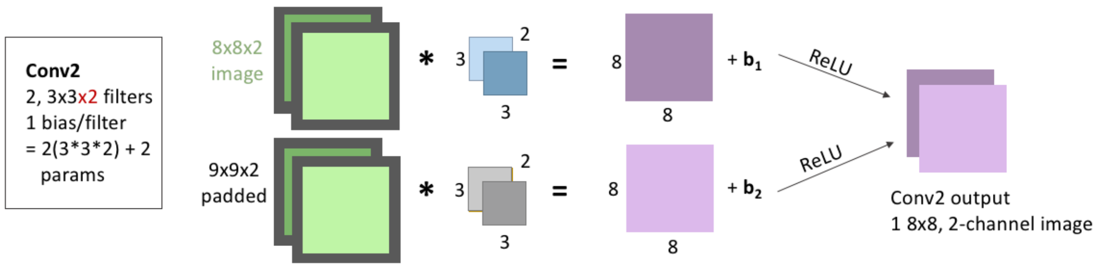
As with the first convolution layer, we pad the input image with zeros, add a bias after the convolution, and pass the images through the ReLu activation function. These output images are again stacked to become two channels of the same image, making the layer output **1, 8x8x2(channel) image**.

Because we have 2 filters, each with 3x3x2 weights, and an associated bias for each filter, this means this second layer has a total of:

$$2 \times (3 \times 3 \times 2) + 2 = 38$$

learnable parameters.

```
display(Im('%s/images/layers/conv2.png' %filepath, height=270, width
    =1000))
```



We add this to our model, the same way we added the first convolution:

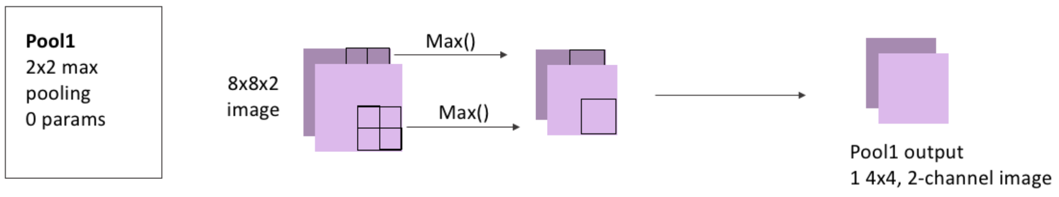
```
conv2 = Conv2D(filters = 2,    # 2 filters again
               kernel_size = (3,3),  # size 3x3 filters. Keras is smart,
    so we don't need to tell it that these filters need to have 2
                                     # channels; it will know that
    because it will know that the input to the layer has 2 channels
```

```
              activation = 'relu', padding = 'same',     # we'll be
    keeping the activation, padding, and initializer the same for all
              kernel_initializer = 'he_normal')(conv1)   # of our layers
    . But note, the input to this layer was now the output from
                                                         # conv1
```

The last step in this convolution block is pooling, where we downsize our image by applying 2x2 pooling to it.

There are no learnable parameters in a pooling step, but it's important to note that we *do not* pool across channels - our **8x8x2** output becomes **4x4x2**, because the two channels are each pooled seperately and remain stacked.

```
display(Im('%s/images/layers/pool1.png' %filepath, height=270, width
    =1000))
```



Adding a pooling layer to our model is also straightforward with Keras:

```
pool1 = MaxPooling2D(pool_size=(2, 2))(conv2)    # we just need to tell
    it the size of the region to pool,
                                                 # and that we're pooling
     the output from conv2
```

5.5.4 Conv Block 2

The second convolution block mimics the first, but we'll increase the number of filters:

- Conv3 has 3 3x3 filter convolutions

- Conv4 has 3 3x3 filter convolutions

We also won't pool here, as the image is already small enough, and the next step will be to re-increase the image size.

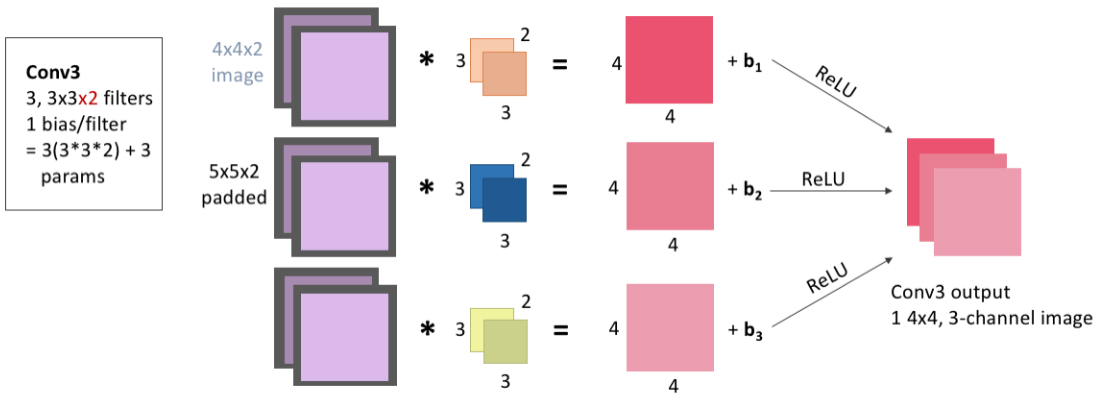
For Conv3, the input image is convolved three times: with 3 filters that each have size 3x3, and 2 channels because our output from the pooling layer had 2 channels. As always, we first pad the input image with zeros, add our bias after the convolution, and pass through the ReLu activation function. These images are stacked to become three channels of the same image, and the layer output is **1, 4x4x3(channel) image**.

Because we have 3 filters, each with 3x3x2 weights, and an associated bias for each filter, this means this layer has a total of:

$$3 \times (3 \times 3 \times 2) + 3 = 57$$

learnable parameters.

```
display(Im('%s/images/layers/conv3.png' %filepath, height=370, width
    =1000))
```



Adding this to our model:

```
conv3 = Conv2D(filters = 3, kernel_size = (3,3),  #again, we don't need
    to tell it that we'll need 2-channel filters
                activation = 'relu', padding = 'same', kernel_initializer
     = 'he_normal')(pool1)
```

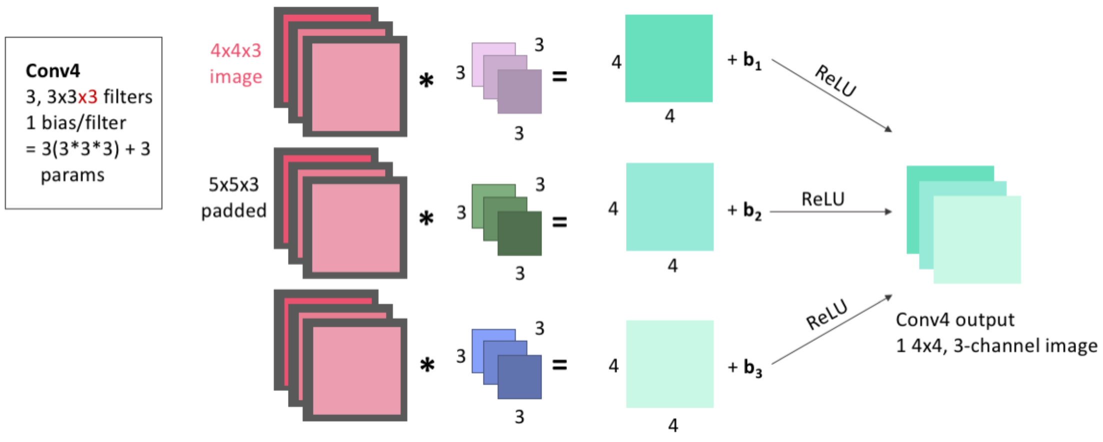
For Conv4, the input image is convolved three times: with 3 filters that each have size 3x3, and 3 channels because our output from the Conv3 layer had 3 channels (because it was convolved with 3 filters). We pad the input image with zeros, add our bias after the convolution, and pass through the ReLu activation function. These images are stacked to become three channels of the same image, and the layer output is **1, 4x4x3(channel) image**.

Because we have 3 filters, each with 3x3x3 weights, and an associated bias for each filter, this means this layer has a total of:

$$3 \times (3 \times 3 \times 3) + 3 = 84$$

learnable parameters.

```
display(Im('%s/images/layers/conv4.png' %filepath, height=370, width
    =1000))
```



Again, this is easy to add to the model:

```
conv4 = Conv2D(filters = 3, kernel_size = 3,  # if you just give
    kernel_size a single number, it assumes a square filter of that
    dimension
               activation = 'relu', padding = 'same', kernel_initializer
     = 'he_normal')(conv3)
```

5.5.5 UpConv Block 1

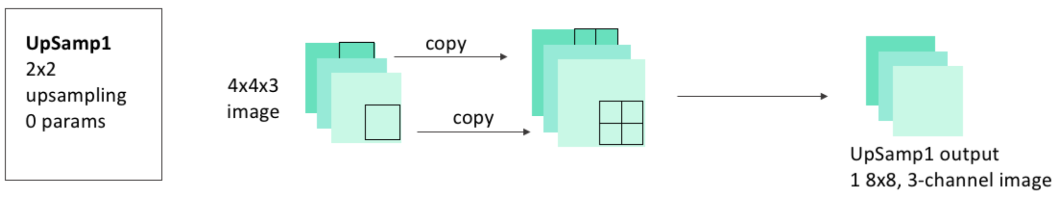
Next, we begin the upsizing portion of the UNet. This first Up-Convolution block will have 3 steps:

- UpSamp1 will do 2x2 upsampling
- Conv5 has 2 3x3 filter convolutions
- Concat1 will stack Conv5 output with Conv2 output

In the upsampling step, we'll upsize our image taking each pixel and copying it into a 2x2 square.

There are no learnable parameters in an upsampling step, but it's important to note that, as with pooling, channels aren't upsampled - our **4x4x3** image becomes **8x8x3**, because the two channels are each upsampled separately and remain stacked.

```
display(Im('%s/images/layers/upsamp1.png' %filepath, height=270, width
    =1000))
```



Adding the upsampling layer to our model is also as easy as adding the pooling layer was:

```
up1 = UpSampling2D(size = (2,2))(conv4)
```

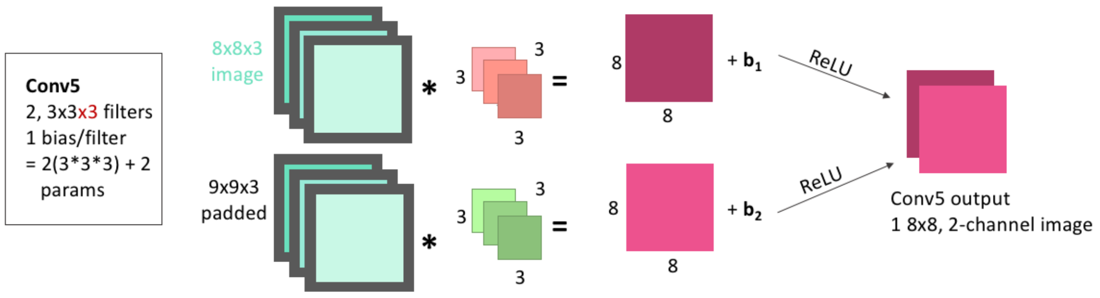
In the Conv5 step, the image we just created by upsampling is convolved two times: with 2 filters that each have size 3x3, and 3 channels. As always, we pad the input image with zeros before the convolution, and add one bias for each filter before passing the image through the ReLu activation function. These images are stacked to become two channels of the same image, and the layer output is **1, 8x8x2(channel) image**.

Because we have 2 filters, each with 3x3x3 weights, and an associated bias for each filter, this means this layer has a total of:

$$2 \times (3 \times 3 \times 3) + 2 = 56$$

learnable parameters.

```
display(Im('%s/images/layers/conv5.png' %filepath, height=270, width
    =1000))
```



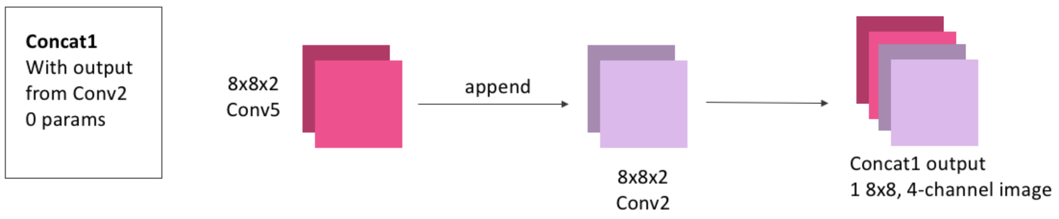
We can add this convolutional layer to the mode making sure that we are applying it to the output from the upsampling layer:

```
conv5 = Conv2D(2, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(up1)
```

Next is the concatenation step. We now have an 8x8x(2 channel) image as the output from Conv5. We also had, from our downsizing steps, an 8x8x(2 channel) image as the output from Conv2. In the concatenation step, we stack these together so that we have an 8x8x(4 channel) image.

Concatenations, because they just involve stacking images, will have no learnable parameters.

```
display(Im('%s/images/layers/concat1.png' %filepath, height=270, width
    =1000))
```



To do this concatenation with keras, we just need to specify what layer outputs (conv5 and conv2) we're looking to concatenate:

```
concat1 = concatenate([conv2,conv5], axis = 3)      # axis = 3 tells the
    model that we need to stack these images as extra channels. Both
                                                    # conv5 and conv3
    will have shape (None, 8, 8, 2), so axis = 3 means to stack along
    the axis
                                                    # which has shape 2,
    the channel axis
```

5.5.6 UpConv Block 2

This is the final block in our model. It will contain

- Conv6: 2, 3x3 convolutions

- Conv7: 1 3x3 convolution

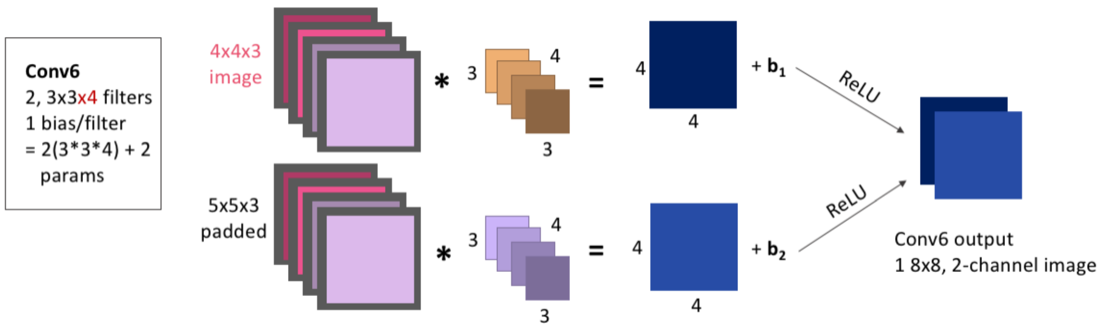
Conv6 will convolve our concatenated image, with 2 filters that each have size 3x3, and 4 channels. We will pad, add bias, and ReLU as usual. The layer output is **1, 8x8x2(channel) image**.

Because we have 2 filters, each with 3x3x4 weights, and an associated bias for each filter, this means this layer has a total of:

$$2 \times (3 \times 3 \times 4) + 2 = 74$$

learnable parameters.

```
display(Im('%s/images/layers/conv6.png' %filepath, height=370, width
    =1000))
```



We add this to the model the same as any other convolutional layer, making sure we apply it to the output from our concatenation:

```
conv6 = Conv2D(2, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(concat1)
```

Finally, Conv7 will convolve our image, with 1 filter of size 3x3, and 2 channels. Because our input only had 1 channel, our final convolution must use only 1 filter, to ensure that the output has only 1 channel. We will pad and add bias as usual.

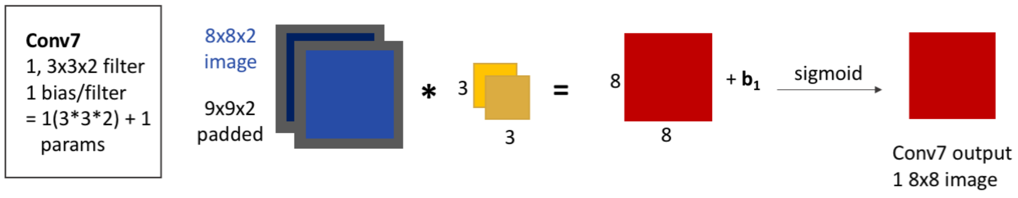
The one difference from all of our other convolutional layers that we'll make is using the **sigmoid** activation function rather than ReLU. Because of the shape of the sigmoid function, values are more easily forced to be either 0 or 1. Because this is our output layer, and we know that our data was comprised of exclusively 0 or 1-valued pixels, the signmoid function will hopefully help to squash our pixel values to the correct one of these two values.

The layer output is **1, 8x8 image**. Because we have 1 filter with 3x3x2 weights, and an associated bias, this layer has a total of:

$$1 \times (3 \times 3 \times 2) + 1 = 19$$

learnable parameters.

```
display(Im('%s/images/layers/conv7.png' %filepath, height=270, width
    =1000))
```



Adding this to our model:

```
conv7 = Conv2D(1, 3, activation = 'sigmoid', padding = 'same',
    kernel_initializer = 'he_normal')(conv6) # note the change in
    activation function
```

5.5.7 The Final Model

ow, let's finish putting the model together, and have a look at the model summary that keras gives us, and try it out.

To finish up our model, we just need to define it by telling keras what layer is the input and what is the output. We'll also need to compile the model before we can use it, where we'll get to choose a few hyperparameters. To keep it simple, we'll choose a common optimizer, the **Adam** optimizer, and only specify the **learning rate**. We also need to choose what loss function to use, and we'll use the **mean-squared error**, which keras already has built in for us.

This tutorial isn't meant to cover the huge body of options for all of these hyperparameters, loss functions, and other functionalities that we can add when compiling our model, but the keras website: https://keras.io/models/model/ does a good job of listing all of the options it has for the compile method.

```
simple_model = Model(input = inputs, output = conv7) # we tell it that
    the first layer is the input layer, and that conv7 is going to be
    the layer that gives us the output. All of the layers in between
    were connected as we defined them, so we don't need to give the
    model any of those here.

simple_model.compile(optimizer = Adam(lr = .0005), loss = 'mse') # Adam
    is an extremely common optimizer, and the lr is the learning rate
```

Keras will also display for us a summary of our model, showing the different layers, their shapes, and the number of learnable parameters per layer, and is a handy way to make sure that the model is consistent and doing everything we expect it to.

```
simple_model.summary()
```

```
Model: "model_1"
________________________________________________________________________________
```

```
Layer (type)                     Output Shape         Param #
   Connected to
==========================================================================

input_1 (InputLayer)             (None, 8, 8, 1)      0
__________________________________________________________________________

conv2d_1 (Conv2D)                (None, 8, 8, 2)      20          input_1
   [0][0]
__________________________________________________________________________

conv2d_2 (Conv2D)                (None, 8, 8, 2)      38
   conv2d_1[0][0]
__________________________________________________________________________

max_pooling2d_1 (MaxPooling2D)   (None, 4, 4, 2)      0
   conv2d_2[0][0]
__________________________________________________________________________

conv2d_3 (Conv2D)                (None, 4, 4, 3)      57
   max_pooling2d_1[0][0]
__________________________________________________________________________

conv2d_4 (Conv2D)                (None, 4, 4, 3)      84
   conv2d_3[0][0]
__________________________________________________________________________

up_sampling2d_1 (UpSampling2D)   (None, 8, 8, 3)      0
   conv2d_4[0][0]
__________________________________________________________________________

conv2d_5 (Conv2D)                (None, 8, 8, 2)      56
   up_sampling2d_1[0][0]
```

```
________________________________________________________________________________________________________

concatenate_1 (Concatenate)      (None, 8, 8, 4)      0
    conv2d_2[0][0]

    conv2d_5[0][0]
________________________________________________________________________________________________________

conv2d_6 (Conv2D)                (None, 8, 8, 2)      74
    concatenate_1[0][0]
________________________________________________________________________________________________________

conv2d_7 (Conv2D)                (None, 8, 8, 1)      19
    conv2d_6[0][0]
========================================================================================================

Total params: 348
Trainable params: 348
Non-trainable params: 0
________________________________________________________________________________________________________
```

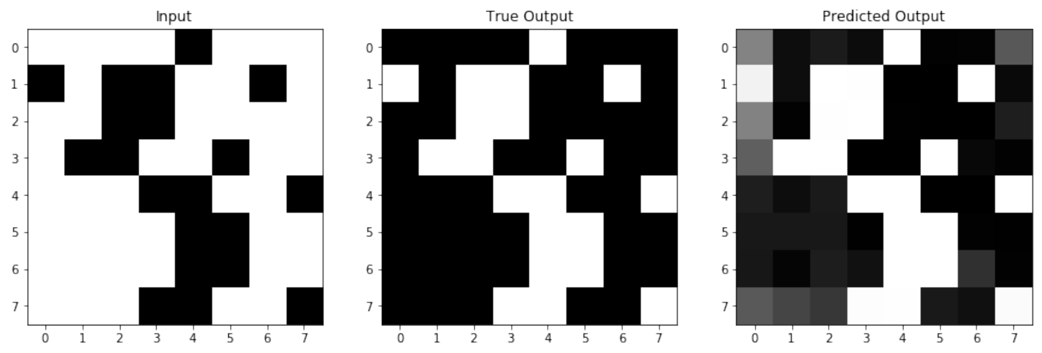
We can see, if we go back and check the number of learnable parameters, and the output shapes for each of these layers, that they match exactly what we expected. The fact that the model compiles properly is good news, too - we'll get an error if we tried to build a model that doesn't connect properly or where the shapes don't make sense.

Training a model in keras is also super simple. Let's try training this model, on our example data, for 500 **epochs** - 500 iterations of the model seeing all of the example images and adjusting the weights accordingly.

```
simple_model_history = simple_model.fit(X_example, y_example,   # the
    fake data we made, X is input, y is output
                                        epochs = 500,           # we'll
    try out 100 epochs
                                         verbose = 1)            #
    verbose = 1 tells keras that we want to see how well the model is
    doing at every epoch
```

So, how did the model do? Let's check by using the *im_in* and *im_out* example images that we made at the start of this section, which the model hasn't seen before, to see if it can do the inversion for us.

```
fig = plt.figure(figsize=(15,5))
ax1,ax2,ax3 = fig.subplots(1,3)

predicted_im_out = simple_model.predict(im_in.reshape(1,8,8,1)).reshape
    (8,8)
# even though our image was 8x8 to begin with, keras needs the shape to
    match the input shape it expected, so we have to reshape twice

ax1.imshow(im_in, cmap="Greys"), ax1.set_title("Input")
ax2.imshow(im_out, cmap="Greys"), ax2.set_title("True Output")
ax3.imshow(predicted_im_out, cmap="Greys"), ax3.set_title("Predicted
    Output")
```



It doesn't do too bad! You could rerun the last 2 cells, training the model for another 500 epochs (it will keep training what it currently has instead of training all over again, unless you re-run model.compile()), and check if it does even better (spoiler: it does). But either way, let's move on and look at exactly what's happening under the hood of this network a little more closely.

5.6 Interpreting Filters

So far, we've been talking in detail about what a UNet does in a more conceptual way. But now that we've built a real example, let's take a closer look at how real data is transformed by a real model. Then, we'll build a model for a real-world problem, and try to see how much we can visualize and understand from a model doing a much more complicated transformation.

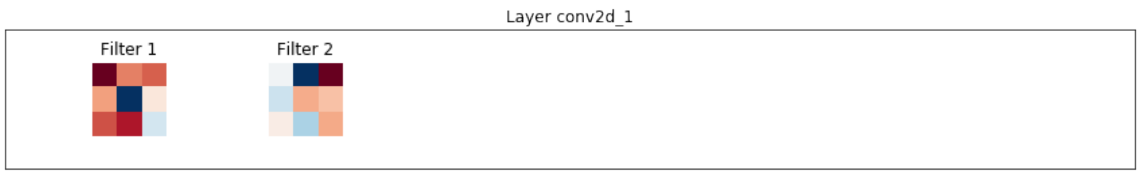
5.6.1 Our Simple Example

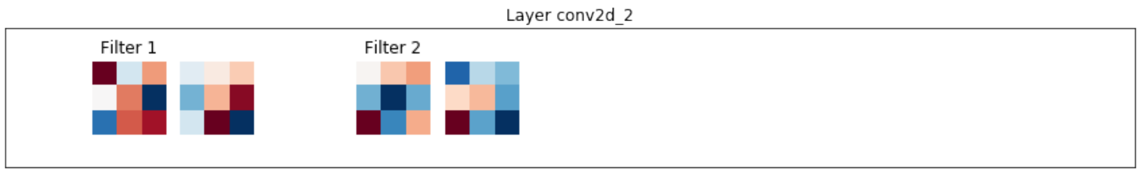
The simple inversion example that we did didn't have many layers or filters. So, we can actually go through and easily look at every single weight our model learned.

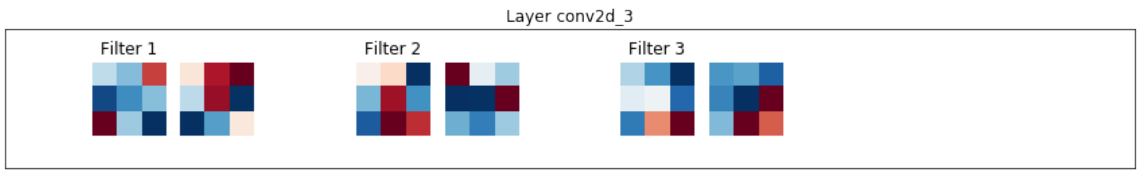
Keras saves the weights that all of our filters had, so it's easy to go through and display all of our model's filters:

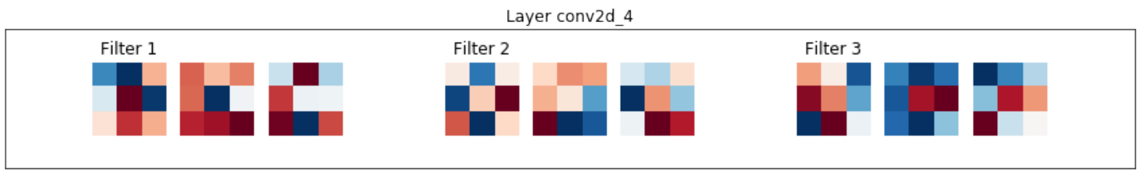
```
# iterate over all of our layers
for layer in simple_model.layers:
  if "conv" in layer.name:       # there are other (non-conv) layers,
    with no learnable params
    filters, biases = layer.get_weights()            # filters will have
     shape (filter_size, filter_size, number channels, number filter)
    f_min, f_max = filters.min(), filters.max()
    filters = (filters - f_min) / (f_max - f_min)    # normalize all
    filters for one layer

    # set up one figure per layer
    fig = plt.figure(figsize=(12,2))
    plt.title("Layer %s" %(layer.name)), plt.xticks([]), plt.yticks([])
    # say what layer we're on, and box in layers (but without axis ticks
    )
```

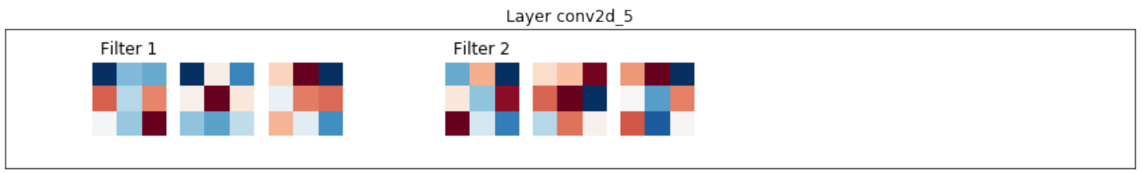
```
fig_count = 1    # to keep track of subplots

# Iterate over all the filters in the layer
for i in range(filters.shape[-1]):    # 'i' will iterate over the number of filters in that layer
 ax = fig.add_subplot(1,13,fig_count), plt.axis('off')    # this adds a dummy subplot, just to leave some whitespace between new filters
 fig_count += 1
 for j in range(filters.shape[-2]):     # 'j' will iterate over the channels of the filter
  ax = fig.add_subplot(1,13,fig_count)  # make a new subplot per channel per filter
  if j == 0:
   ax.set_title("Filter %d" %((i+1)))  # only title the first channel of the filter, to keep it neat
  plt.imshow(filters[:,:,j,i],cmap='RdBu'), plt.axis('off')
  fig_count += 1
plt.tight_layout()
```

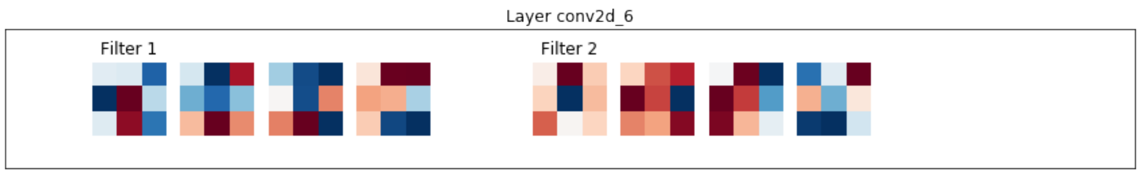


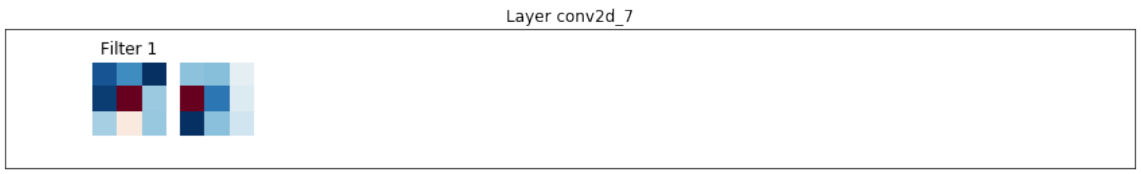












Looking at all of our filters *can* be useful, but it also can show us a bunch of seemingly random weights, as it does above. What's often more useful is to look at the layer **activations** - or the "version" of the image that convolving with a filter, adding its bias, and passing through an activation function gives us. For each of the filters above, we can look at the activation that would come from convolving an image with it.

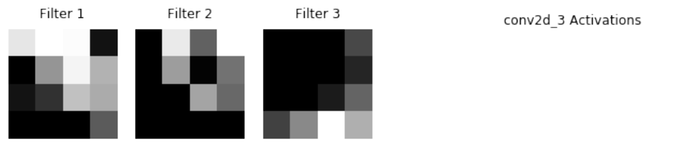
So, let's use our example image again, and see what the activations for that image are at each layer.

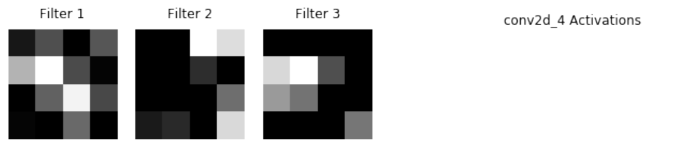
```
layer_outputs = [layer.output for layer in simple_model.layers[1:]]    # the first layer is input, but we want all layers after that
activation_model = Model(inputs=simple_model.input, outputs=layer_outputs)    # Make a model that returns each layer output given the input
activations = activation_model.predict(im_in.reshape(1,8,8,1))    # then, get all the activations for our test image


for layer_num in range(len(activations)):
  layer_activation = activations[layer_num]
  fig = plt.figure(figsize = (15,2*np.ceil(layer_activation.shape[-1]/9)))
  plt.title("%s Activations" %simple_model.layers[layer_num+1].name)
  plt.axis('off')
  for filter_num in range(layer_activation.shape[-1]):
    ax = fig.add_subplot(np.ceil(layer_activation.shape[-1]/9), 9, filter_num+1)
    ax.matshow(layer_activation[0, :, :, filter_num], cmap='Greys_r')
    if "conv" in simple_model.layers[layer_num+1].name:
      ax.set_title("Filter %d" %(filter_num+1))    # only have filters in the conv layers, else just call them Im 1/2/etc..
    else:
      ax.set_title("Im %d" %(filter_num+1))
    plt.axis('off')
  plt.tight_layout()
```

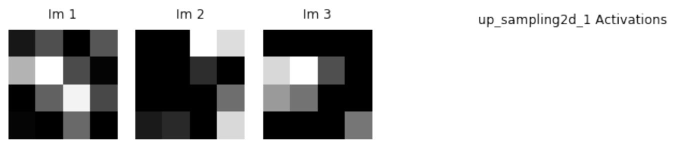
Filter 1
Filter 2
conv2d_1 Activations

Filter 1
Filter 2
conv2d_2 Activations

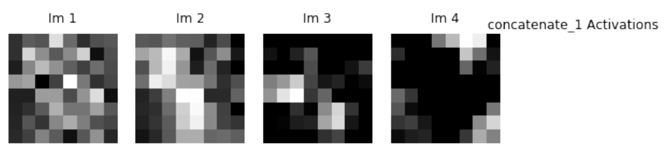
Im 1
Im 2
max_pooling2d_1 Activations


Filter 1
Filter 2
Filter 3
conv2d_3 Activations


Filter 1
Filter 2
Filter 3
conv2d_4 Activations


Im 1
Im 2
Im 3
up_sampling2d_1 Activations

Filter 1
Filter 2
conv2d_5 Activations



Filter 1
Filter 2
conv2d_6 Activations

Filter 1
conv2d_7 Activations

So, those aren't exactly interesting either. There are a few things to point out here:

- The pooling, upsampling, and concatenate ”activations” (there's no activation function in these layers so the name doesn't make as much sense for them) are plotted, so you can confirm these operations are acting as we expect.
- Because the activations are after the different channels of the filter have summed, we are showing one image per filter, not one image per filter channel.

But, even though we can look at all of our weights and activations, we aren't getting too much insight here on how our image is being inverted by these operations. A big part of this could be because the task we're asking the U-Net to perform - inverting an image - doesn't actually care too much about the features in the image. It isn't like identifying an eye to figure out that you're looking at a face - each pixel in this case could be inverted completely independently from those around it.

5.6.2 A Real-World Example

So instead, let's take a look at a real-world example, where the features of the image are much more important, and try to see what we can learn about what the U-Net is doing.

The example data and U-Net that we're going to be using are originally from:

https://github.com/zhixuhao/unet

But we're going to augment the data differently, and I've made the U-Net smaller (in both number of layers and number of filters) to make it more manageable to visualize.

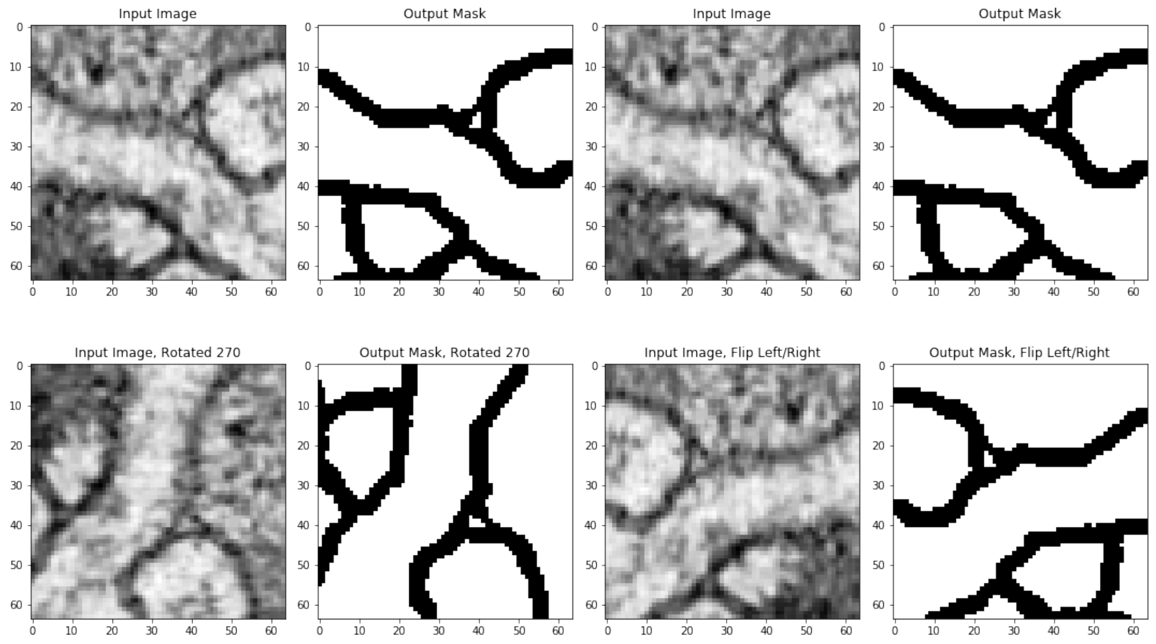
The task for this U-Net is image segmentation. Given a grayscale image, we want to create a black/white mask, where areas of interest are in black and all other areas are in white. Let's start by loading in the data so we can take a look at an example. We are also going to **augment** the data as we load it in. That means, we'll rotate and/or transpose images as we read them in, so that they can become multiple new images to use for training. We'll also chop each of our images (which begin as 512x512) into multiple smaller images, again, because smaller images will be easier to visualize later.

```
input_path = '%s/data/ims' %filepath
output_path = '%s/data/labels' %filepath

rotation_angles = [90,180,270]     # set degrees to rotate by, leave []
    if no augmentation
mirror_angles = [Image.TRANSPOSE,Image.FLIP_LEFT_RIGHT,Image.
    FLIP_TOP_BOTTOM]    # set types of mirroring/transposing the image

sz = 512   # image sizes (one dimension specified, images must be square
    )
```

```
n_crops = 8   #  number of images (along one dimension) to crop original to. Will end up with n_crops*n_crops images for every original image

cropped_sz = int(sz/n_crops)  # size of image we want to crop to. sz/n_crops must be an integer to divide images evenly
sz = cropped_sz   # set the new size to the size of the cropped images

X = []
for file in np.sort(os.listdir(input_path)):
  full_im = Image.open('%s/%s' %(input_path, file))
  for i in range(n_crops):
    for j in range(n_crops):
      box = (i*cropped_sz,j*cropped_sz,(i+1)*cropped_sz,(j+1)*cropped_sz)    # get the region of the full image that will become the cropped image
      im = full_im.crop(box)
      X.append(np.array(im).reshape(sz,sz,1))
      for angle in rotation_angles:     # rotate images on all angles
        rotated_im = im.rotate(angle)
        X.append(np.array(rotated_im).reshape(sz,sz,1))  # add rotated im as new one
      for angle in mirror_angles:    # transpose images on all angles
        mirror_im = im.transpose(angle)
        X.append(np.array(mirror_im).reshape(sz,sz,1))  # add transposed im as new one

# build y in the same way as X, with images and rotations in the same order to they match each other
y = []
for file in np.sort(os.listdir(output_path)):
  full_im = Image.open('%s/%s' %(output_path, file))
  for i in range(n_crops):
    for j in range(n_crops):
      box = (i*cropped_sz,j*cropped_sz,(i+1)*cropped_sz,(j+1)*cropped_sz
    )
      im = full_im.crop(box)
      y.append(np.array(im).reshape(sz,sz,1))
      for angle in rotation_angles:
        rotated_im = im.rotate(angle)
        y.append(np.array(rotated_im).reshape(sz,sz,1))
      for angle in mirror_angles:
        mirror_im = im.transpose(angle)
        y.append(np.array(mirror_im).reshape(sz,sz,1))

input_size = X[0].shape

X = np.array(X)
y = np.array(y)

# scale data to a 0-1 range
X = (X-np.min(X))/(np.max(X)-np.min(X))
y = (y-np.min(y))/(np.max(y)-np.min(y))
```

Now let's look at some examples of our images, and confirm that our augmentations worked properly.

```
fig = plt.figure(figsize=(15,9))
ax1 = fig.add_subplot(2,4,1)
ax1.imshow(X[0].reshape(sz,sz),cmap='Greys_r'), plt.title("Input Image")
ax2 = fig.add_subplot(2,4,2)
ax2.imshow(y[0].reshape(sz,sz),cmap='Greys_r'), plt.title("Output Mask")
ax3 = fig.add_subplot(2,4,5)
ax3.imshow(X[3].reshape(sz,sz),cmap='Greys_r'), plt.title("Input Image,
    Rotated 270")
ax4 = fig.add_subplot(2,4,6)
ax4.imshow(y[3].reshape(sz,sz),cmap='Greys_r'), plt.title("Output Mask,
```



```
    Rotated 270")
ax5 = fig.add_subplot(2,4,3)
ax5.imshow(X[0].reshape(sz,sz),cmap='Greys_r'), plt.title("Input Image")
ax6 = fig.add_subplot(2,4,4)
ax6.imshow(y[0].reshape(sz,sz),cmap='Greys_r'), plt.title("Output Mask")
ax7 = fig.add_subplot(2,4,7)
ax7.imshow(X[5].reshape(sz,sz),cmap='Greys_r'), plt.title("Input Image,
    Flip Left/Right")
ax8 = fig.add_subplot(2,4,8)
ax8.imshow(y[5].reshape(sz,sz),cmap='Greys_r'), plt.title("Output Mask,
    Flip Left/Right")
plt.tight_layout()

print("We now have %d pieces of data" %len(X))
```

And we now have 13440 pieces of data. Now, let's build the model that we'll train on this problem. We're going to build this model exactly the same way that we built the very simple model, but it will have more layers and more filters.

```
inputs = Input(input_size)
# conv block 1
conv1 = Conv2D(16, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(inputs)
conv1 = Conv2D(16, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
# conv block 2
conv2 = Conv2D(32, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(pool1)
conv2 = Conv2D(32, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
# conv block 3
conv3 = Conv2D(64, 5, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(pool2)
conv3 = Conv2D(64, 5, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
# conv block 4
conv4 = Conv2D(128, 5, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(pool3)
conv4 = Conv2D(128, 5, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)
# upconv block 1
up7 = Conv2D(64, 2, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv4))
concat7 = concatenate([conv3,up7], axis = 3)
conv7 = Conv2D(64, 5, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(concat7)
conv7 = Conv2D(64, 5, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv7)
```

```
# upconv block 2
up8 = Conv2D(32, 2, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv7))
concat8 = concatenate([conv2,up8], axis = 3)
conv8 = Conv2D(32, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(concat8)
conv8 = Conv2D(32, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv8)
# upconv block 3
up9 = Conv2D(16, 2, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv8))
concat9 = concatenate([conv1,up9], axis = 3)
conv9 = Conv2D(16, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(concat9)
conv9 = Conv2D(16, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv9)
conv9 = Conv2D(2, 3, activation = 'relu', padding = 'same',
    kernel_initializer = 'he_normal')(conv9)
# output
conv10 = Conv2D(1, 1, activation = 'sigmoid')(conv9)

model = Model(input = inputs, output = conv10)
model.compile(optimizer = Adam(lr = 1e-4), loss = 'mse', metrics = ['
    accuracy'])
model.summary()
```

```
Model: "model_3"
______________________________________________________________________________________

Layer (type)                    Output Shape         Param #
    Connected to
==================================================================================
input_2 (InputLayer)            (None, 64, 64, 1)    0

conv2d_8 (Conv2D)               (None, 64, 64, 16)   160         input_2
    [0][0]

conv2d_9 (Conv2D)               (None, 64, 64, 16)   2320
    conv2d_8[0][0]

max_pooling2d_2 (MaxPooling2D)  (None, 32, 32, 16)   0
    conv2d_9[0][0]

conv2d_10 (Conv2D)              (None, 32, 32, 32)   4640
    max_pooling2d_2[0][0]

conv2d_11 (Conv2D)              (None, 32, 32, 32)   9248
    conv2d_10[0][0]

max_pooling2d_3 (MaxPooling2D)  (None, 16, 16, 32)   0
    conv2d_11[0][0]

conv2d_12 (Conv2D)              (None, 16, 16, 64)   51264
    max_pooling2d_3[0][0]

conv2d_13 (Conv2D)              (None, 16, 16, 64)   102464
    conv2d_12[0][0]
```

```
__________________________________________________________________________________________________

max_pooling2d_4 (MaxPooling2D)  (None, 8, 8, 64)     0
    conv2d_13[0][0]
__________________________________________________________________________________________________

conv2d_14 (Conv2D)              (None, 8, 8, 128)    204928
    max_pooling2d_4[0][0]
__________________________________________________________________________________________________

conv2d_15 (Conv2D)              (None, 8, 8, 128)    409728
    conv2d_14[0][0]
__________________________________________________________________________________________________

up_sampling2d_2 (UpSampling2D)  (None, 16, 16, 128)  0
    conv2d_15[0][0]
__________________________________________________________________________________________________

conv2d_16 (Conv2D)              (None, 16, 16, 64)   32832
    up_sampling2d_2[0][0]
__________________________________________________________________________________________________

concatenate_2 (Concatenate)     (None, 16, 16, 128)  0
    conv2d_13[0][0]

    conv2d_16[0][0]
__________________________________________________________________________________________________

conv2d_17 (Conv2D)              (None, 16, 16, 64)   204864
    concatenate_2[0][0]
__________________________________________________________________________________________________

conv2d_18 (Conv2D)              (None, 16, 16, 64)   102464
```

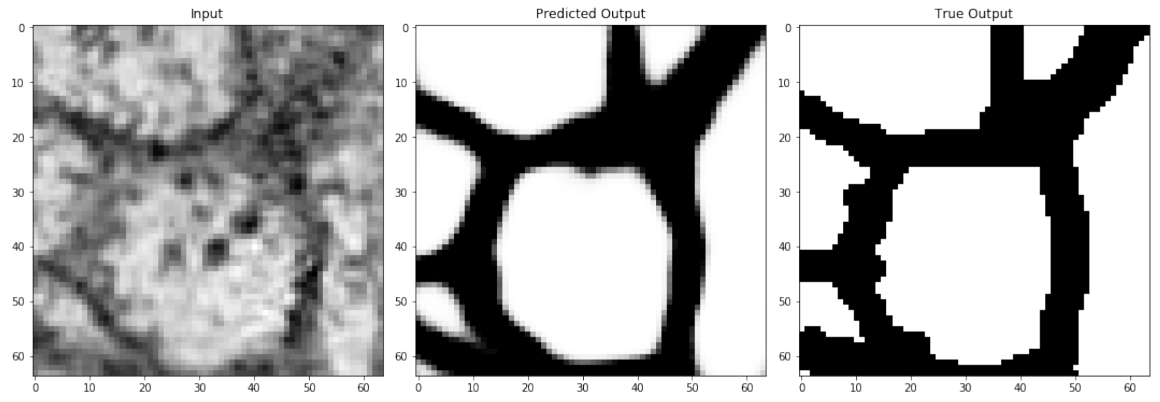
```
   conv2d_17[0][0]
________________________________________________________________________________

up_sampling2d_3 (UpSampling2D)  (None, 32, 32, 64)   0
   conv2d_18[0][0]
________________________________________________________________________________

conv2d_19 (Conv2D)              (None, 32, 32, 32)   8224
   up_sampling2d_3[0][0]
________________________________________________________________________________

concatenate_3 (Concatenate)     (None, 32, 32, 64)   0
   conv2d_11[0][0]

   conv2d_19[0][0]
________________________________________________________________________________

conv2d_20 (Conv2D)              (None, 32, 32, 32)   18464
   concatenate_3[0][0]
________________________________________________________________________________

conv2d_21 (Conv2D)              (None, 32, 32, 32)   9248
   conv2d_20[0][0]
________________________________________________________________________________

up_sampling2d_4 (UpSampling2D)  (None, 64, 64, 32)   0
   conv2d_21[0][0]
________________________________________________________________________________

conv2d_22 (Conv2D)              (None, 64, 64, 16)   2064
   up_sampling2d_4[0][0]
________________________________________________________________________________
```

```
concatenate_4 (Concatenate)     (None, 64, 64, 32)   0
    conv2d_9[0][0]

    conv2d_22[0][0]
__________________________________________________________________________________________

conv2d_23 (Conv2D)              (None, 64, 64, 16)   4624
    concatenate_4[0][0]
__________________________________________________________________________________________

conv2d_24 (Conv2D)              (None, 64, 64, 16)   2320
    conv2d_23[0][0]
__________________________________________________________________________________________

conv2d_25 (Conv2D)              (None, 64, 64, 2)    290
    conv2d_24[0][0]
__________________________________________________________________________________________

conv2d_26 (Conv2D)              (None, 64, 64, 1)    3
    conv2d_25[0][0]
==========================================================================================

Total params: 1,170,149
Trainable params: 1,170,149
Non-trainable params: 0
__________________________________________________________________________________________
```

Let's train this model:

```
model_history = model.fit(X, y, epochs = 100, verbose = 1,batch_size
    =100)
```

Now, before we take a closer look at what the model is doing, let's look at how accu-

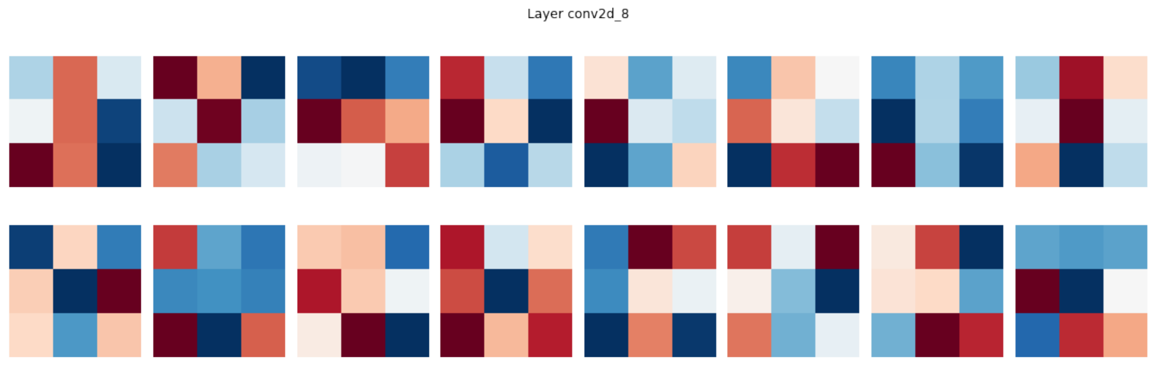


rately it was able to produce our desired masks to see if we've successfully completed our task.

```
data_num = 2500  # can change to look at a different image and its
    prediction

input_im = X[data_num].reshape(sz,sz)
predicted_im = model.predict(X[data_num:data_num+1]).reshape(sz,sz)
output_im = y[data_num].reshape(sz,sz)

fig = plt.figure(figsize=(15,5))
ax1, ax2, ax3 = fig.subplots(1,3)
ax1.imshow(input_im,cmap='Greys_r'), ax1.set_title("Input")
ax2.imshow(predicted_im,cmap='Greys_r'), ax2.set_title("Predicted Output
    ")
ax3.imshow(output_im,cmap='Greys_r'), ax3.set_title("True Output")
plt.tight_layout()
```

And it looks pretty good! Okay, so let's look at the filters again for this network, and see if any of them seem to stand out as picking out some features for us.

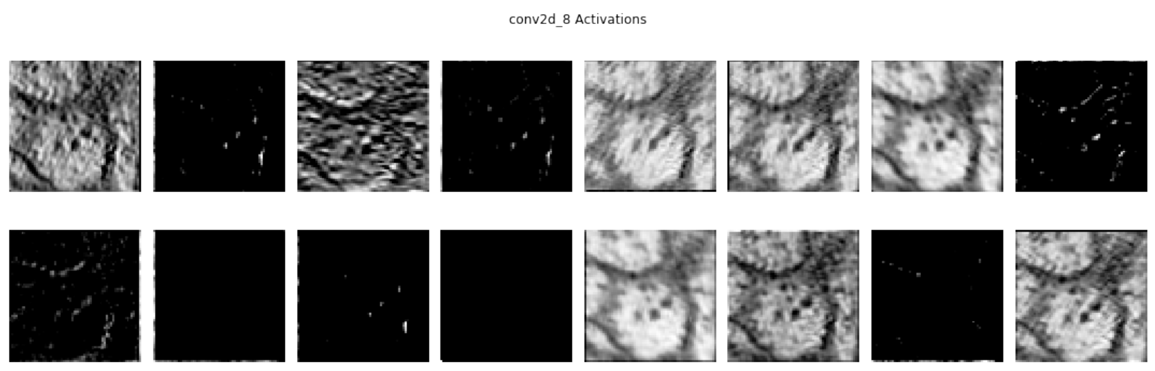
We'll do this in the exact same way that we did above, except we'll only look at the filters for the *first* convolutional layer of the *first* convolutional block. This is because we have way more filters in this example than we did in the last example, so even in the second



convolution of the first convolution block involves filters with 16 channels, so visualizing all of them would be extremely hard.

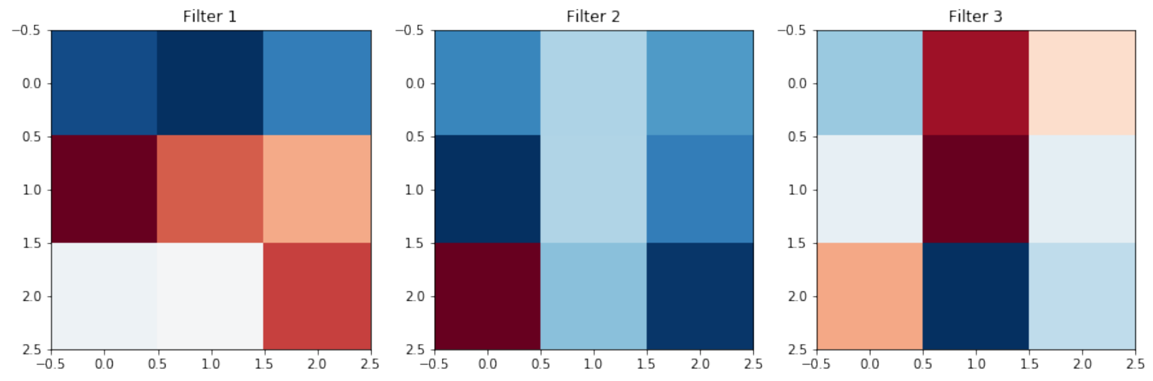
```
for layer in model.layers[1:2]:
  if "conv" in layer.name:
    filters, biases = layer.get_weights()
    f_min, f_max = filters.min(), filters.max()
    filters = (filters - f_min) / (f_max - f_min)

    fig = plt.figure(figsize=(15,5))
    plt.title("Layer %s" %layer.name)
    plt.axis('off')
    for i in range(filters.shape[-1]):
      ax = fig.add_subplot(np.ceil(filters.shape[-1]/8),8,i+1)
      plt.imshow(filters[:,:,:,i].reshape(filters.shape[0],filters.shape
[1]),cmap='RdBu')
      plt.axis('off')
    plt.tight_layout()
```

It might not be any more evident in this real-world example than it was in our very simple example what these filters are doing. But, now that our images have significant structure, we might be able to figure out what these filters are telling us. Let's look at the activations for this first convolutional layer to see if anything jumps out.



```
layer_outputs = [layer.output for layer in model.layers[1:]]
activation_model = Model(inputs=model.input, outputs=layer_outputs) #
    Creates a model that will return these outputs, given the model
    input
activations = activation_model.predict(input_im.reshape(1,64,64,1))

for layer_num in range(1):
  layer_activation = activations[layer_num]
  fig = plt.figure(figsize = (15,5))
  plt.title("%s Activations" %model.layers[layer_num+1].name)
  plt.axis('off')
  for filter_num in range(layer_activation.shape[-1]):
    ax = fig.add_subplot(np.ceil(layer_activation.shape[-1]/8), 8,
    filter_num+1)
    ax.imshow(layer_activation[0, :, :, filter_num], cmap='Greys_r')
    plt.axis('off')
  plt.tight_layout()
```

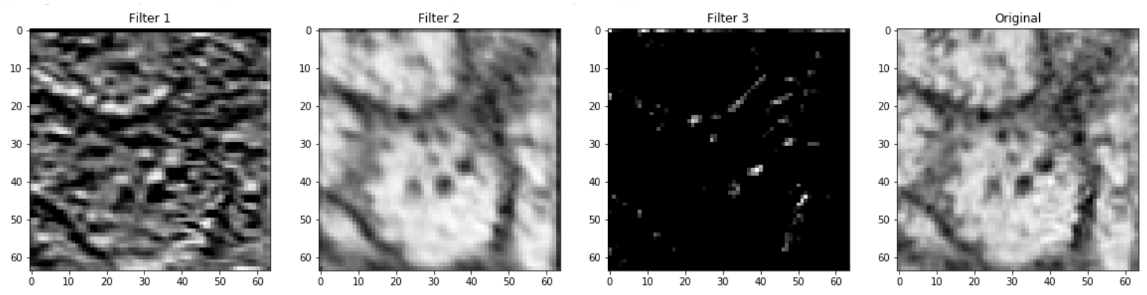
Now, if you've just been reading the cells of this notebook without re-running them, you have the same filters and activations as I do. Otherwise, if you've been re-running the cells, and you've re-trained the model, your filters and activations will be different than mine. But, you should see that the activations all look like versions of the image that are meant to pick out something.



In the cells below, I'm going to manually load in some of the filters and activations that I have in my notebook, and talk about them. If you didn't re-run the notebook, this will be filters and activations 2,6, and 7 (python-indexed) from above. Otherwise, your filters and activations will be different, but likely some will still be relatively similar, to mine.

So, here are the filters we're going to look at:

```
filt_2 = np.array([[0.6744713 , 0.69218516, 0.6413911 ],
                   [0.36131164, 0.426186  , 0.4645831 ],
                   [0.534594  , 0.5280584 , 0.4137286 ]])

filt_6 = np.array([[0.7352305,  0.61722165, 0.70441914],
                   [0.85438216, 0.61504716, 0.7480632 ],
                   [0.16859733, 0.65413386, 0.84601194]])

filt_7 = np.array([[0.6218604,  0.25640523, 0.45756572],
                   [0.5342595,  0.20999649, 0.53925157],
                   [0.3942946,  0.810168,   0.5870034 ]])

fig = plt.figure(figsize=(15,5))
ax1,ax2,ax3 = fig.subplots(1,3)
ax1.imshow(filt_2,cmap='RdBu'), ax1.set_title("Filter 1")
ax2.imshow(filt_6,cmap='RdBu'), ax2.set_title("Filter 2")
ax3.imshow(filt_7,cmap='RdBu'), ax3.set_title("Filter 3")
```



Now, let’s look at the activations these filters gave:

```
display(Im('%s/images/realWorld_example_activations.png' %filepath,
    height=270, width=1000))
```

So, let’s try to speculate what these filters might be doing.

- Filter 1 looks like it might be a horizontal/diagonal edge detector. It seems to pick out the strongest borders between black and and white pixels in the original image, provided that border isn’t vertical.

- Filter 2 doesn’t seem to change the image significantly, besides blurring it somewhat to smooth out the lighter sections.

- Filter 3 seems to only emphasize the darkest regions of the original image: the highest concentration of black pixels in the original image are highlighted, and the rest of the image is uniform.

If we go back and look at our filters, is there any evidence that this is what they’re doing? Well, maybe. If you look at the structure of them, you can pick out that Filter 1 might have differing values horizontally/diagonally which might be giving us the horizontal edges, or that Filter 2 might be keeping information from all the cells about equally which blurs the image out, but it’s easier to look at the activation and say what it’s doing based off of that than to make guesses based on the weights.

5.6.3 Deeper Layers

And what about the deeper layers?

Well, the activations are easy enough to visualize. Although in the layers with 64 or 128 filters, this would mean looking at 128 images, that's much easier than trying to look at the filters, which will end up with 128 channels.

However, it can prove more challenging to interpret later activations because they are now convolved versions of already convolved versions of our images. So while these activations can still provide valuable insight at every level, it's difficult to straightforwardly interpret what exactly the UNet is doing.

Still, it's clearly an incredibly powerful tool for image transformation, which hopefully you now have a little better of an understanding/intuition about!

References

Agarwal, S., Davé, R., and Bassett, B. A. (2018). Painting galaxies into dark matter haloes using machine learning. *Monthly Notices of the Royal Astronomical Society*, 478(3):3410–3422.

Angora, G., Rosati, P., Brescia, M., Mercurio, A., Grillo, C., Caminha, G., Meneghetti, M., Nonino, M., Vanzella, E., Bergamini, P., Biviano, A., and Lombardi, M. (2020). The search for galaxy cluster members with deep learning of panchromatic HST imaging and extensive spectroscopy. *Astronomy and Astrophysics*, 643:1–25.

Angulo, R. E. and Hahn, O. (2022). *Large-scale dark matter simulations*, volume 8.

Angulo, R. E., Lacey, C. G., Baugh, C. M., and Frenk, C. S. (2009). The fate of substructures in cold dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 399(2):983–995.

Angulo, R. E. and White, S. D. (2010). One simulation to fit them all - changing the background parameters of a cosmological N-body simulation. *Monthly Notices of the Royal Astronomical Society*, 405(1):143–154.

Avila, S., Crocce, M., Ross, A. J., García-Bellido, J., Percival, W. J., Banik, N., Camacho, H., Kokron, N., Chan, K. C., Andrade-Oliveira, F., Gomes, R., Gomes, D., Lima, M., Rosenfeld, R., Salvador, A. I., Friedrich, O., Abdalla, F. B., Annis, J., Benoit-Lévy, A., Bertin, E., Brooks, D., Carrasco Kind, M., Carretero, J., Castander, F. J., Cunha, C. E., da Costa, L. N., Davis, C., De Vicente, J., Doel, P., Fosalba, P., Frieman, J., Gerdes, D. W., Gruen, D., Gruendl, R. A., Gutierrez, G., Hartley, W. G., Hollowood, D., Honscheid, K., James, D. J., Kuehn, K., Kuropatkin, N., Miquel, R., Plazas, A. A., Sanchez, E., Scarpine, V., Schindler, R., Schubnell, M., Sevilla-Noarbe, I., Smith, M., Sobreira, F., Suchyta, E., Swanson, M. E., Tarle, G., Thomas, D., and Walker, A. R. (2018). Dark energy survey year-1 results: Galaxy mock catalogues for BAO. *Monthly Notices of the Royal Astronomical Society*, 479(1):94–110.

Avila, S., Knebe, A., Pearce, F. R., Schneider, A., Srisawat, C., Thomas, P. A., Behroozi, P., Elahi, P. J., Han, J., Mao, Y.-Y., Onions, J., Rodriguez-Gomez, V., and Tweed, D. (2014). SUSSING MERGER TREES: the influence of the halo finder. *Monthly Notices of the Royal Astronomical Society*, 441(4):3488–3501.

Banerji, M., Lahav, O., Lintott, C. J., Abdalla, F. B., Bamford, S. P., Andreescu, D., Murray, P., Jordan, M., Slosar, A., Szalay, A., Thomas, D., and Vandenberg, J. (2010). Galaxy Zoo : Reproducing Galaxy Morphologies via Machine Learning arXiv : 0908 . 2033v2 [astro-ph . CO] 22 Mar 2010. *Main*, 000(March).

Barchi, P., de Carvalho, R., Rosa, R., Sautter, R., Soares-Santos, M., Marques, B., Clua, E., Gonçalves, T., de Sá-Freitas, C., and Moura, T. (2020). Machine and Deep Learning applied to galaxy morphology - A comparative study. *Astronomy and Computing*, 30(January):100334.

Barnes, J. and Hut, P. (1986). A hierarchical O(N log N) force-calculation algorithm. $\backslash$*nat*, 324(6096):446–449.

Behroozi, P., Knebe, A., Pearce, F. R., Elahi, P., Han, J., Lux, H., Mao, Y.-Y., Muldrew, S. I., Potter, D., and Srisawat, C. (2015). Major mergers going Notts: challenges for modern halo finders. *Monthly Notices of the Royal Astronomical Society*, 454(3):3020–3029.

Behroozi, P. S., Wechsler, R. H., and Wu, H.-Y. (2013a). THE ROCKSTAR PHASE-SPACE TEMPORAL HALO FINDER AND THE VELOCITY OFFSETS OF CLUSTER CORES. *The Astrophysical Journal*, 762(2):109.

Behroozi, P. S., Wechsler, R. H., Wu, H.-Y., Busha, M. T., Klypin, A. A., and Primack, J. R. (2013b). GRAVITATIONALLY CONSISTENT HALO CATALOGS AND MERGER TREES FOR PRECISION COSMOLOGY. *The Astrophysical Journal*, 763(1):18.

Berger, P. and Stein, G. (2019). A volumetric deep Convolutional Neural Network for simulation of mock dark matter halo catalogues. *Monthly Notices of the Royal Astronomical Society*, 482(3):2861–2871.

Berlind, A. A. and Weinberg, D. H. (2002). The Halo Occupation Distribution: Toward an Empirical Determination of the Relation between Galaxies and Mass. *The Astrophysical Journal*, 575(2):587–616.

Bernardini, M., Feldmann, R., Anglés-Alcázar, D., Boylan-Kolchin, M., Bullock, J., Mayer, L., and Stadel, J. (2022). From EMBER to FIRE*predicting high resolution baryon fields from dark matter silations with deep learning. *Monthly Notices of the Royal Astronomical Society*, 509(1):1323–1341.

Bernardini, M., Mayer, L., Reed, D., and Feldmann, R. (2020). Predicting dark matter halo formation in N-body simulations with deep regression networks. *Monthly Notices of the Royal Astronomical Society*, 496(4):5116–5125.

Bertschinger, E. (1998). Simulations of structure formation in the universe. *Annual Review of Astronomy and Astrophysics*, 36(1):599–654.

Blaauw, A. and Schmidt, M. (1965). *Galactic structure*.

Borgani, S. and Kravtsov, A. (2011). Cosmological simulations of galaxy clusters. *Advanced Science Letters*, 4(2):204–227.

Boylan-Kolchin, M., Ma, C.-P., and Quataert, E. (2008). Dynamical friction and galaxy merging time-scales. *Monthly Notices of the Royal Astronomical Society*, 383(1):93–101.

Boylan-Kolchin, M., Springel, V., White, S. D., Jenkins, A., and Lemson, G. (2009). Resolving cosmic structure formation with the Millennium-II Simulation. *Monthly Notices of the Royal Astronomical Society*, 398(3):1150–1164.

Brandenberger, R. H. (1985). Quantum field theory methods and inflationary universe models. *Reviews of Modern Physics*, 57(1):1–60.

Bretonnière, H., Boucaud, A., and Huertas-Company, M. (2021). Probabilistic segmentation of overlapping galaxies for large cosmological surveys. (NeurIPS 2021).

Brooks, A. M., Kuhlen, M., Zolotov, A., and Hooper, D. (2013). A BARYONIC SOLUTION TO THE MISSING SATELLITES PROBLEM. *The Astrophysical Journal*, 765(1):22.

Brooks, A. M. and Zolotov, A. (2014). WHY BARYONS MATTER: THE KINEMATICS OF DWARF SPHEROIDAL SATELLITES. *The Astrophysical Journal*, 786(2):87.

Bullock, J. S., Dekel, A., Kolatt, T. S., Kravtsov, A. V., Klypin, A. A., Porciani, C., and Primack, J. R. (2001). A Universal Angular Momentum Profile for Galactic Halos. *The Astrophysical Journal*, 555(1):240–257.

Caldeira, J., Wu, W. L., Nord, B., Avestruz, C., Trivedi, S., and Story, K. T. (2019). DeepCMB: Lensing reconstruction of the cosmic microwave background with deep neural networks. *Astronomy and Computing*, 28.

Calderon, V. F. and Berlind, A. A. (2019). Prediction of galaxy halo masses in SDSS DR7 via a machine learning approach. *Monthly Notices of the Royal Astronomical Society*, 490(2):2367–2379.

Campbell, D., van den Bosch, F. C., Padmanabhan, N., Mao, Y. Y., Zentner, A. R., Lange, J. U., Jiang, F., and Villarreal, A. (2018). The galaxy clustering crisis in abundance matching. *Monthly Notices of the Royal Astronomical Society*, 477(1):359–383.

Cañameras, R., Schuldt, S., Suyu, S. H., Taubenberger, S., Meinhardt, T., Leal-Taixé, L., Lemon, C., Rojas, K., and Savary, E. (2020). HOLISMOKES: II. Identifying galaxy-scale strong gravitational lenses in Pan-STARRS using convolutional neural networks. *Astronomy and Astrophysics*, 644:1–28.

Cavaglia, M., Staats, K., and Gill, T. (2019). Finding the Origin of Noise Transients in LIGO Data with Machine Learning. *Communications in Computational Physics*, 25(4).

Centrella, J. and Melott, A. L. (1983). Three-dimensional simulation of large-scale structure in the universe. \nat, 305:196–198.

Chardin, J., Uhlrich, G., Aubert, D., Deparis, N., Gillet, N., Ocvirk, P., and Lewis, J. (2019). A deep learning model to emulate simulations of cosmic reionization. *Monthly Notices of the Royal Astronomical Society*, 490(1):1055–1065.

Chua, K. T. E., Pillepich, A., Rodriguez-Gomez, V., Vogelsberger, M., Bird, S., and Hernquist, L. (2017). Subhalo demographics in the Illustris simulation: effects of baryons and halo-to-halo variation. *Monthly Notices of the Royal Astronomical Society*, 472(4):4343–4360.

Cole, S., Aragon-Salamanca, A., Frenk, C. S., Navarro, J. F., and Zepf, S. E. (1994). A recipe for galaxy formation. $\backslash$*mnras*, 271:781–806.

Coles, P. (2001). Large—scale Structure, Theory and Statistics. *Phase Transitions in the Early Universe: Theory and Observations*, pages 217–247.

Coles, P. and Chiang, L.-Y. (2000). Characterizing the nonlinear growth of large-scale structure in the Universe . *Nature*, 406(6794):376–378.

Crocce, M., Pueblas, S., and Scoccimarro, R. (2006). Transients from initial conditions in cosmological simulations. *Monthly Notices of the Royal Astronomical Society*, 373(1):369–381.

Croton, D. J., Springel, V., White, S. D. M., De Lucia, G., Frenk, C. S., Gao, L., Jenkins, A., Kauffmann, G., Navarro, J. F., and Yoshida, N. (2006). The many lives of active galactic nuclei: cooling flows, black holes and the luminosities and colours of galaxies. *Monthly Notices of the Royal Astronomical Society*, 365(1):11–28.

Cuéllar, S., Granados, P., Fabregas, E., Curé, M., Vargas, H., Dormido-Canto, S., and Farias, G. (2022). Deep learning exoplanets detection by combining real and synthetic data. *Plos One*, 17(5):e0268199.

Curtis, O., Brainerd, T. G., and Hernandez, A. (2022). Cosmic voids in GAN-generated maps of large-scale structure. *Astronomy and Computing*, 38.

Davis, M., Efstathiou, G., Frenk, C. S., and White, S. D. M. (1985). The evolution of large-scale structure in a universe dominated by cold dark matter. $\backslash$*apj*, 292:371–394.

de Andres, D., Cui, W., Ruppin, F., De Petris, M., Yepes, G., Lahouli, I., Aversano, G., Dupuis, R., and Jarraya, M. (2022). Mass Estimation of Planck Galaxy Clusters using Deep Learning. *EPJ Web of Conferences*, 257(1):00013.

De La Calleja, J. and Fuentes, O. (2004). Machine learning and image analysis for morphological galaxy classification. *Monthly Notices of the Royal Astronomical Society*, 349(1):87–93.

De Lucia, G. and Blaizot, J. (2007). The hierarchical formation of the brightest cluster galaxies. *Monthly Notices of the Royal Astronomical Society*, 375(1):2–14.

de Oliveira, R. A., Villaescusa-Navarro, F., Li, Y., Ho, S., and Spergel, D. N. (2020). Fast and accurate non-linear predictions of universes with deep learning. *arXiv*, (NeurIPS 2020):1–6.

Despali, G. and Vegetti, S. (2017). The impact of baryonic physics on the subhalo mass function and implications for gravitational lensing. *Monthly Notices of the Royal Astronomical Society*, 469(2):1997–2010.

Dey, B., Andrews, B. H., Newman, J. A., Mao, Y.-Y., Rau, M. M., and Zhou, R. (2021). Photometric Redshifts from SDSS Images with an Interpretable Deep Capsule Network. 18(December):1–18.

Diaferio, A., Kauffmann, G., Colberg, J. M., and White, S. D. M. (1999). Clustering of galaxies in a hierarchical universe – III. Mock redshift surveys. *Monthly Notices of the Royal Astronomical Society*, 307(3):537–552.

Diemand, J., Kuhlen, M., and Madau, P. (2007). Formation and Evolution of Galaxy Dark Matter Halos and Their Substructure. *The Astrophysical Journal*, 667(2):859–877.

Diemand, J. and Moore, B. (2011). The Structure and Evolution of Cold Dark Matter Halos. *Advanced Science Letters*, 4(2):297–310.

Diemand, J., Moore, B., and Stadel, J. (2004). Velocity and spatial biases in cold dark matter subhalo distributions. *Monthly Notices of the Royal Astronomical Society*, 352(2):535–546.

Diemer, B. and Kravtsov, A. V. (2014). Dependence of the Outer Density Profiles of Halos on Their Mass Accretion Rate. $\backslash$*apj*, 789(1):1.

Dolag, K., Borgani, S., Murante, G., and Springel, V. (2009). Substructures in hydrodynamical cluster simulations. *Monthly Notices of the Royal Astronomical Society*, 399(2):497–514.

Domínguez Sánchez, H., Margalef, B., Bernardi, M., and Huertas-Company, M. (2022). SDSS-IV DR17: Final release of MaNGA PyMorph photometric and deep-learning morphological catalogues. *Monthly Notices of the Royal Astronomical Society*, 509(3):4024–4036.

Dong, X., Ramachandra, N., Habib, S., Heitmann, K., Buehlmann, M., and Madireddy, S. (2021). Physical Benchmarking for AI-Generated Cosmic Web. (Dl):1–6.

Farrens, S., Lacan, A., Guinot, A., and Vitorelli, A. Z. (2022). Deep transfer learning for blended source identification in galaxy survey data. *Astronomy and Astrophysics*, 657:1–9.

Feder, R. M., Berger, P., and Stein, G. (2020). Nonlinear 3D cosmic web simulation with heavy-tailed generative adversarial networks. *Physical Review D*, 102(10).

Feldman, H. A., Kaiser, N., and Peacock, J. A. (1994). Power-spectrum analysis of three-dimensional redshift surveys. *The Astrophysical Journal*, 426:23.

Feng, Y., Chu, M. Y., Seljak, U., and McDonald, P. (2016). FastPM: A new scheme for fast simulations of dark matter and haloes. *Monthly Notices of the Royal Astronomical Society*, 463(3):2273–2286.

Franx, M., Illingworth, G., and de Zeeuw, T. (1991). The ordered nature of elliptical galaxies - Implications for their intrinsic angular momenta and shapes. *The Astrophysical Journal*, 383:112.

Fry, J. N. (1986). Statistics of Voids in Hierarchical Universes. $\backslash$*apj*, 306:358.

Gan, J., Kang, X., Van Den Bosch, F. C., and Hou, J. (2010). An improved model for the dynamical evolution of dark matter subhaloes. *Monthly Notices of the Royal Astronomical Society*, 408(4):2201–2212.

Gao, L., Frenk, C. S., Boylan-Kolchin, M., Jenkins, A., Springel, V., and White, S. D. M. (2011). The statistics of the subhalo abundance of dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 410(4):2309–2314.

Gao, L., White, S. D. M., Jenkins, A., Stoehr, F., and Springel, V. (2004). The subhalo populations of ΛCDM dark haloes. *Monthly Notices of the Royal Astronomical Society*, 355(3):819–834.

Garrison-Kimmel, S., Wetzel, A., Bullock, J. S., Hopkins, P. F., Boylan-Kolchin, M., Faucher-Giguère, C.-A., Kereš, D., Quataert, E., Sanderson, R. E., Graus, A. S., and Kelley, T. (2017). Not so lumpy after all: modelling the depletion of dark matter subhaloes by Milky Way-like galaxies. *Monthly Notices of the Royal Astronomical Society*, 471(2):1709–1727.

Ghigna, S., Moore, B., Governato, F., Lake, G., Quinn, T., and Stadel, J. (2000). Density Profiles and Substructure of Dark Matter Halos: Converging Results at Ultra-High Numerical Resolution. *The Astrophysical Journal*, 544(2):616–628.

Gill, S. P. D., Knebe, A., and Gibson, B. K. (2004a). The evolution of substructure - I. A new identification method. *\mnras*, 351(2):399–409.

Gill, S. P. D., Knebe, A., Gibson, B. K., and Dopita, M. A. (2004b). The evolution of substructure – II. Linking dynamics to environment. *Monthly Notices of the Royal Astronomical Society*, 351(2):410–422.

Giocoli, C., Tormen, G., Sheth, R. K., and van den Bosch, F. C. (2010). The substructure hierarchy in dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 18(April):1–18.

Giocoli, C., Tormen, G., and van den Bosch, F. C. (2008). The population of dark matter subhaloes: mass functions and average mass-loss rates. *Monthly Notices of the Royal Astronomical Society*, 386(4):2135–2144.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Greig, B., Wyithe, J. S. B., Murray, S. G., Mutch, S. J., and Trott, C. M. (2022). Generating extremely large-volume reionisation simulations. 000(May).

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377.

Guo, Q. and White, S. (2014). Numerical resolution limits on subhalo abundance matching. *Monthly Notices of the Royal Astronomical Society*, 437(4):3228–3235.

Guo, Z. and Martini, P. (2019). Classification of Broad Absorption Line Quasars with a Convolutional Neural Network. *The Astrophysical Journal*, 879(2):72.

Gupta, N. and Reichardt, C. L. (2020). Mass Estimation of Galaxy Clusters with Deep Learning. I. Sunyaev–Zel'dovich Effect. *The Astrophysical Journal*, 900(2):110.

Guth, A. H. and Pi, S. Y. (1982). Fluctuations in the New Inflationary Universe. *\prl*, 49(15):1110–1113.

Guzman, E. and Meyers, J. (2022). Reconstructing cosmic polarization rotation with ResUNet-CMB. *Journal of Cosmology and Astroparticle Physics*, 2022(1):1–15.

Han, J., Cole, S., Frenk, C. S., and Jing, Y. (2016). A unified model for the spatial and mass distribution of subhaloes. *Monthly Notices of the Royal Astronomical Society*, 457(2):1208–1223.

Hand, N., Feng, Y., Beutler, F., Li, Y., Modi, C., Seljak, U., and Slepian, Z. (2018). nbodykit: An Open-source, Massively Parallel Toolkit for Large-scale Structure. *The Astronomical Journal*, 156(4):160.

Harnois-Déraps, J. and Van Waerbeke, L. (2015). Simulations of weak gravitational lensing - II. Including finite support effects in cosmic shear covariance matrices. *Monthly Notices of the Royal Astronomical Society*, 450(3):2857–2873.

Hayashi, E., Navarro, J. F., Taylor, J. E., Stadel, J., and Quinn, T. (2003). The Structural Evolution of Substructure. *The Astrophysical Journal*, 584(2):541–558.

Hayashi, H. and Chiba, M. (2009). ON THE SIZE EVOLUTION OF A GALACTIC DISK IN HIERARCHICAL MERGING OF COLD DARK MATTER HALOS. *The Astrophysical Journal*, 702(2):871–879.

He, S., Li, Y., Feng, Y., Ho, S., Ravanbakhsh, S., Chen, W., and Póczos, B. (2019). Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences of the United States of America*, 116(28):13825–13832.

He, Z., Er, X., Long, Q., Liu, D., Liu, X., Li, Z., Liu, Y., Deng, W., and Fan, Z. (2020). Deep learning for strong lensing search: Tests of the convolutional neural networks and new candidates from KiDS DR3. *Monthly Notices of the Royal Astronomical Society*, 497(1):556–571.

Henghes, B., Thiyagalingam, J., Pettitt, C., Hey, T., and Lahav, O. (2022). Deep learning methods for obtaining photometric redshift estimations from images. *Monthly Notices of the Royal Astronomical Society*, 512(2):1696–1709.

Henriques, B. M. B. and Thomas, P. A. (2010). Tidal disruption of satellite galaxies in a semi-analytic model of galaxy formation. *Monthly Notices of the Royal Astronomical Society*, 403(2):768–779.

Hiroshima, N., Ando, S., and Ishiyama, T. (2018). Modeling evolution of dark matter substructure and annihilation boost. *Physical Review D*, 97(12):123002.

Hirschmann, M., Naab, T., Somerville, R. S., Burkert, A., and Oser, L. (2012). Galaxy formation in semi-analytic models and cosmological hydrodynamic zoom simulations. *Monthly Notices of the Royal Astronomical Society*, 419(4):3200–3222.

Hong, S. E., Park, S., Jee, M. J., Bak, D., and Cha, S. (2021). Weak-lensing Mass Reconstruction of Galaxy Clusters with a Convolutional Neural Network. *The Astrophysical Journal*, 923(2):266.

Hoyle, B. (2016). Measuring photometric redshifts using galaxy images and Deep Neural Networks. *Astronomy and Computing*, 16:34–40.

Huang, X., Storfer, C., Ravi, V., Pilon, A., Domingo, M., Schlegel, D. J., Bailey, S., Dey, A., Gupta, R. R., Herrera, D., Juneau, S., Landriau, M., Lang, D., Meisner, A., Moustakas, J., Myers, A. D., Schlafly, E. F., Valdes, F., Weaver, B. A., Yang, J., and Yèche, C. (2020). Finding Strong Gravitational Lenses in the DESI DECam Legacy Survey. *The Astrophysical Journal*, 894(1):78.

Isola, P., Zhu, J. Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:5967–5976.

Jacobs, C., Glazebrook, K., Collett, T., More, A., and McCarthy, C. (2017). Finding strong lenses in CFHTLS using convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, 471(1):167–181.

Jenkins, A., Frenk, C. S., White, S. D. M., Colberg, J. M., Cole, S., Evrard, A. E., Couchman, H. M. P., and Yoshida, N. (2001). The mass function of dark matter haloes. *\mnras*, 321(2):372–384.

Jiang, C. Y., Jing, Y. P., Faltenbacher, A., Lin, W. P., and Li, C. (2008). A Fitting Formula for the Merger Timescale of Galaxies in Hierarchical Clustering. *The Astrophysical Journal*, 675(2):1095–1105.

Jiang, F. and van den Bosch, F. C. (2014). Generating merger trees for dark matter haloes: a comparison of methods. *Monthly Notices of the Royal Astronomical Society*, 440(1):193–207.

Jiang, F. and van den Bosch, F. C. (2016). Statistics of dark matter substructure – I. Model and universal fitting functions. *Monthly Notices of the Royal Astronomical Society*, 458(3):2848–2869.

Jiang, F. and van den Bosch, F. C. (2017). Statistics of dark matter substructure – III. Halo-to-halo variance. *Monthly Notices of the Royal Astronomical Society*, 472(1):657–674.

Jo, Y. and Kim, J.-h. (2019). Machine-assisted Semi-Simulation Model (MSSM): Estimating Galactic Baryonic Properties from Their Dark Matter Using A Machine Trained on Hydrodynamic Simulations. *Monthly Notices of the Royal Astronomical Society*, 17(August):1–17.

Johnson, J. W., Maller, A. H., Berlind, A. A., Sinha, M., and Holley-Bockelmann, J. K. (2019). The secondary spin bias of dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 486(1):1156–1166.

Joyce, M., Garrison, L., and Eisenstein, D. (2021). Quantifying resolution in cosmological N-body simulations using self-similarity. *Monthly Notices of the Royal Astronomical Society*, 501(4):5051–5063.

Kahn, F. D. and Woltjer, L. (1959). Intergalactic Matter and the Galaxy. *\apj*, 130:705.

Kamdar, H. M., Turk, M. J., and Brunner, R. J. (2016). Machine learning and cosmological simulations – II. Hydrodynamical simulations. *Monthly Notices of the Royal Astronomical Society*, 457(2):1162–1179.

Kampakoglou, M. and Benson, A. J. (2007). Tidal mass loss from collisionless systems. *Monthly Notices of the Royal Astronomical Society*, 374(3):775–786.

Kasmanoff, N., Villaescusa-Navarro, F., Tinker, J., and Ho, S. (2020). dm2gal: Mapping Dark Matter to Galaxies with Neural Networks. (NeurIPS).

Kauffmann, G., White, S. D. M., and Guiderdoni, B. (1993). The formation and evolution of galaxies within merging dark matter haloes. *\mnras*, 264:201–218.

Kazantzidis, S., Zentner, A. R., Kravtsov, A. V., Bullock, J. S., and Debattista, V. P. (2009). COLD DARK MATTER SUBSTRUCTURE AND GALACTIC DISKS. II. DYNAMICAL EFFECTS OF HIERARCHICAL SATELLITE ACCRETION. *The Astrophysical Journal*, 700(2):1896–1920.

Khalifa, N. E. M., Taha, M. H. N., Hassanien, A. E., and Selim, I. M. (2017). Deep Galaxy: Classification of Galaxies based on Deep Convolutional Neural Networks.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*.

Klimentowski, J., Łokas, E. L., Knebe, A., Gottlöber, S., Martinez-Vaquero, L. A., Yepes, G., and Hoffman, Y. (2010). The grouping, merging and survival of subhaloes in the simulated Local Group. *Monthly Notices of the Royal Astronomical Society*, 402(3):1899–1910.

Klypin, A. and Holtzman, J. (1997). Particle-Mesh code for cosmological simulations. *arXiv e-prints*, pages astro–ph/9712217.

Klypin, A. and Prada, F. (2018). Dark matter statistics for large galaxy catalogues: Power spectra and covariance matrices. *Monthly Notices of the Royal Astronomical Society*, 478(4):4602–4621.

Knebe, A., Devriendt, J. E. G., Mahmood, A., and Silk, J. (2002). Merger histories in warm dark matter structure formation scenarios. *Monthly Notices of the Royal Astronomical Society*, 329(4):813–828.

Knebe, A., Knollmann, S. R., Muldrew, S. I., Pearce, F. R., Aragon-Calvo, M. A., Ascasibar, Y., Behroozi, P. S., Ceverino, D., Colombi, S., Diemand, J., Dolag, K., Falck, B. L., Fasel, P., Gardner, J., Gottlöber, S., Hsu, C.-H., Iannuzzi, F., Klypin, A., Lukić, Z., Maciejewski, M., McBride, C., Neyrinck, M. C., Planelles, S., Potter, D., Quilis, V., Rasera, Y., Read, J. I., Ricker, P. M., Roy, F., Springel, V., Stadel, J., Stinson, G., Sutter, P. M., Turchaninov, V., Tweed, D., Yepes, G., and Zemp, M. (2011). Haloes gone MAD: The Halo-Finder Comparison Project. *Monthly Notices of the Royal Astronomical Society*, 415(3):2293–2318.

Knebe, A., Power, C., Gill, S. P. D., and Gibson, B. K. (2006). The importance of interactions for mass loss from satellite galaxies in cold dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 368(2):741–750.

Kosiba, M., Lieu, M., Altieri, B., Clerc, N., Faccioli, L., Kendrew, S., Valtchanov, I., Sadibekova, T., Pierre, M., Hroch, F., Werner, N., Burget, L., Garrel, C., Koulouridis, E., Gaynullina, E., Molham, M., Ramos-Ceja, M. E., and Khalikova, A. (2020). Multiwavelength classification of X-ray selected galaxy cluster candidates using convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, 496(4):4141–4153.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Landy, S. D. and Szalay, A. S. (1993). Bias and Variance of Angular Correlation Functions. $\backslash$*apj*, 412:64.

Lanusse, F., Ma, Q., Li, N., Collett, T. E., Li, C. L., Ravanbakhsh, S., Mandelbaum, R., and Póczos, B. (2018). CMU DeepLens: Deep learning for automatic image-based galaxy-galaxy strong lens finding. *Monthly Notices of the Royal Astronomical Society*, 473(3):3895–3906.

Lazanu, A., Giannantonio, T., Schmittfull, M., and Shellard, E. P. (2017). Matter bispectrum of large-scale structure with Gaussian and non-Gaussian initial conditions: Halo models, perturbation theory, and a three-shape model. *Physical Review D*, 95(8):1–8.

Li, R., Napolitano, N. R., Spiniello, C., Tortora, C., Kuijken, K., Koopmans, L. V. E., Schneider, P., Getman, F., Xie, L., Long, L., Shu, W., Vernardos, G., Huang, Z., Covone, G., Dvornik, A., Heymans, C., Hildebrandt, H., Radovich, M., and Wright, A. H. (2021). High-quality Strong Lens Candidates in the Final Kilo-Degree Survey Footprint. *The Astrophysical Journal*, 923(1):16.

Li, Y., Ni, Y., Croft, R. A. C., Di Matteo, T., Bird, S., and Feng, Y. (2020). AI-assisted super-resolution cosmological simulations. 12(October):1–12.

Lin, Q., Fouchez, D., Pasquet, J., Treyer, M., Ait Ouahmed, R., Arnouts, S., and Ilbert, O. (2022). Photometric redshift estimation with convolutional neural networks and galaxy images: Case study of resolving biases in data-driven methods. *Astronomy & Astrophysics*.

Lucie-Smith, L., Peiris, H. V., and Pontzen, A. (2019). An interpretable machine-learning framework for dark matter halo formation. *Monthly Notices of the Royal Astronomical Society*, 490(1):331–342.

Lucie-Smith, L., Peiris, H. V., Pontzen, A., and Lochner, M. (2018). Machine learning cosmological structure formation. *Monthly Notices of the Royal Astronomical Society*, 479(3):3405–3414.

Lucie-Smith, L., Peiris, H. V., Pontzen, A., Nord, B., and Thiyagalingam, J. (2020). Deep learning insights into cosmological structure formation.

Maksimova, N. A., Garrison, L. H., Eisenstein, D. J., Hadzhiyska, B., Bose, S., and Satterthwaite, T. P. (2021). AbacusSummit: A Massive Set of High-Accuracy, High-Resolution N -Body Simulations.

Manera, M., Scoccimarro, R., Percival, W. J., Samushia, L., McBride, C. K., Ross, A. J., Sheth, R. K., White, M., Reid, B. A., Sánchez, A. G., de Putter, R., Xu, X., Berlind, A. A., Brinkmann, J., Maraston, C., Nichol, B., Montesano, F., Padmanabhan, N., Skibba, R. A., Tojeiro, R., and Weaver, B. A. (2013). The clustering of galaxies in the SDSS-III Baryon Oscillation Spectroscopic Survey: A large sample of mock galaxy catalogues. *Monthly Notices of the Royal Astronomical Society*, 428(2):1036–1054.

Mansfield, P. (2020). gotetra: Cosmic velocity fields tracking through the use of tetrahedra. Astrophysics Source Code Library, record ascl:2005.011.

Mao, Y. Y., Williamson, M., and Wechsler, R. H. (2015). the Dependence of Subhalo Abundance on Halo Concentration. *Astrophysical Journal*, 810(1).

Maslej-Krešňáková, V., El Bouchefry, K., and Butka, P. (2021). Morphological classification of compact and extended radio galaxies using convolutional neural networks and data augmentation techniques. *Monthly Notices of the Royal Astronomical Society*, 505(1):1464–1475.

McBride, C., Berlind, A., Scoccimarro, R., Wechsler, R., Busha, M., Gardner, J., and van den Bosch, F. (2009). Las damas mock galaxy catalogs for SDSS. 41.

McCauliff, S. D., Jenkins, J. M., Catanzarite, J., Burke, C. J., Coughlin, J. L., Twicken, J. D., Tenenbaum, P., Seader, S., Li, J., and Cote, M. (2015). AUTOMATIC CLASSIFICATION of KEPLER PLANETARY TRANSIT CANDIDATES. *Astrophysical Journal*, 806(1).

McCavana, T., Micic, M., Lewis, G. F., Sinha, M., Sharma, S., Holley-Bockelmann, K., and Bland-Hawthorn, J. (2012). The lives of high-redshift mergers. *Monthly Notices of the Royal Astronomical Society*, 424(1):361–371.

Milletari, F., Navab, N., and Ahmadi, S. A. (2016). V-Net: Fully convolutional neural networks for volumetric medical image segmentation. *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, pages 565–571.

Mirza, M. and Osindero, S. (2014). Conditional Generative Adversarial Nets. pages 1–7.

Mishra, A., Reddy, P., and Nigam, R. (2019). CMB-GAN: Fast Simulations of Cosmic Microwave background anisotropy maps using Deep Learning.

Mitchell, P. D., Lacey, C. G., Lagos, C. D. P., Frenk, C. S., Bower, R. G., Cole, S., Helly, J. C., Schaller, M., Gonzalez-Perez, V., and Theuns, T. (2018). Comparing galaxy formation in semi-analytic models and hydrodynamical simulations. *Monthly Notices of the Royal Astronomical Society*, 474(1):492–521.

Monaco, P., Sefusatti, E., Borgani, S., Crocce, M., Fosalba, P., Sheth, R. K., and Theuns, T. (2013). An accurate tool for the fast generation of dark matter halo catalogues. *Monthly Notices of the Royal Astronomical Society*, 433(3):2389–2402.

Moster, B. P., Naab, T., Lindström, M., and O'Leary, J. A. (2020). GalaxyNet: Connecting galaxies and dark matter haloes with deep neural networks and reinforcement learning in large volumes. 21(May):1–21.

Munshi, F., Brooks, A. M., Applebaum, E., Weisz, D. R., Governato, F., and Quinn, T. R. (2017). Going, going, gone dark: Quantifying the scatter in the faintest dwarf galaxies. *arXiv: Astrophysics of Galaxies*.

Munshi, F., Brooks, A. M., Christensen, C., Applebaum, E., Holley-Bockelmann, K., Quinn, T. R., and Wadsley, J. (2019). Dancing in the Dark: Uncertainty in Ultrafaint Dwarf Galaxy Predictions from Cosmological Simulations. *The Astrophysical Journal*, 874(1):40.

Mustafa, M., Bard, D., Bhimji, W., Lukić, Z., Al-Rfou, R., and Kratochvil, J. M. (2019). CosmoGAN: creating high-fidelity weak lensing convergence maps using Generative Adversarial Networks. *Computational Astrophysics and Cosmology*, 6(1).

Nadler, E. O., Mao, Y.-Y., Wechsler, R. H., Garrison-Kimmel, S., and Wetzel, A. (2018). Modeling the Impact of Baryons on Subhalo Populations with Machine Learning. *The Astrophysical Journal*, 859(2):129.

Navarro, J. F., Frenk, C. S., and White, S. D. M. (1996). The Structure of Cold Dark Matter Halos. *The Astrophysical Journal*, 462:563.

Nelson, D., Springel, V., Pillepich, A., Rodriguez-Gomez, V., Torrey, P., Genel, S., Vogelsberger, M., Pakmor, R., Marinacci, F., Weinberger, R., Kelley, L., Lovell, M., Diemer, B., and Hernquist, L. (2019). The IllustrisTNG simulations: public data release. *Computational Astrophysics and Cosmology*, 6(1):1–30.

Neyman, J. and Scott, E. L. (1952). A Theory of the Spatial Distribution of Galaxies. \apj, 116:144.

Ni, Y., Li, Y., Lachance, P., Croft, R. A., Di Matteo, T., Bird, S., and Feng, Y. (2021). AI-assisted superresolution cosmological simulations - II. Halo substructures, velocities, and higher order statistics. *Monthly Notices of the Royal Astronomical Society*, 507(1):1021–1033.

Nipoti, C., Giocoli, C., and Despali, G. (2018). Accretion of satellites on to central galaxies in clusters: merger mass ratios and orbital parameters. *Monthly Notices of the Royal Astronomical Society*, 476(1):705–714.

Nolte, A., Wang, L., Bilicki, M., Holwerda, B., and Biehl, M. (2019). Galaxy classification: A machine learning analysis of GAMA catalogue data. *Neurocomputing*, 342(January 2019):172–190.

Ntampaka, M., Trac, H., Sutherland, D. J., Battaglia, N., Póczos, B., and Schneider, J. (2015). A MACHINE LEARNING APPROACH FOR DYNAMICAL MASS MEASUREMENTS OF GALAXY CLUSTERS. *The Astrophysical Journal*, 803(2):50.

Ntampaka, M., ZuHone, J., Eisenstein, D., Nagai, D., Vikhlinin, A., Hernquist, L., Marinacci, F., Nelson, D., Pakmor, R., Pillepich, A., Torrey, P., and Vogelsberger, M. (2019). A Deep Learning Approach to Galaxy Cluster X-Ray Masses. *The Astrophysical Journal*, 876(1):82.

Nurmi, P., Heinämäki, P., Saar, E., Einasto, M., Holopainen, J., Martínez, V. J., and Einasto, J. (2006). Subhaloes in $\Lambda$CDM cosmological simulations: I. Masses and abundances. *astro-ph/0611941*.

Nusser, A., Dekel, A., Bertschinger, E., and Blumenthal, G. R. (1991). Cosmological Velocity-Density Relation in the Quasi-linear Regime. \apj, 379:6.

Onions, J., Knebe, A., Pearce, F. R., Muldrew, S. I., Lux, H., Knollmann, S. R., Ascasibar, Y., Behroozi, P., Elahi, P., Han, J., Maciejewski, M., Merchán, M. E., Neyrinck, M., Ruiz, A. N., Sgró, M. A., Springel, V., and Tweed, D. (2012). Subhaloes going Notts: the subhalo-finder comparison project. *Monthly Notices of the Royal Astronomical Society*, 423(2):1200–1214.

Ostriker, J. P. and Peebles, P. J. E. (1973). A Numerical Study of the Stability of Flattened Galaxies: or, can Cold Galaxies Survive? \apj, 186:467–480.

Paczynski, B. (1987). Giant luminous arcs discovered in two clusters of galaxies. \nat, 325(6105):572–573.

Pasquet, J., Bertin, E., Treyer, M., Arnouts, S., and Fouchez, D. (2019). Photometric redshifts from SDSS images using a convolutional neural network. *Astronomy and Astrophysics*, 621:1–14.

Pasquet-Itam, J. and Pasquet, J. (2018). Deep learning approach for classifying, detecting and predicting photometric redshifts of quasars in the Sloan Digital Sky Survey stripe 82. *Astronomy and Astrophysics*, 611(1986):1–13.

Peebles, P. J. E. (1974). A model for continuous clustering in the large-scale distribution of matter. *Astrophysics and Space Science*, 31(2):403–410.

Penarrubia, J. and Benson, A. J. (2005). Effects of dynamical evolution on the distribution of substructures. *Monthly Notices of the Royal Astronomical Society*, 364(3):977–989.

Perez, L. A., Malhotra, S., Rhoads, J. E., and Tilvi, V. (2021). Void Probability Function of Simulated Surveys of High-redshift Ly α Emitters . *The Astrophysical Journal*, 906(1):58.

Perraudin, N., Marcon, S., Lucchi, A., and Kacprzak, T. (2021). Emulation of Cosmological Mass Maps with Conditional Generative Adversarial Networks. *Frontiers in Artificial Intelligence*, 4:1–19.

Perraudin, N., Srivastava, A., Lucchi, A., Kacprzak, T., Hofmann, T., and Réfrégier, A. (2019). Cosmological N-body simulations: a challenge for scalable generative models. *Computational Astrophysics and Cosmology*, 6(1).

Petrillo, C. E., Tortora, C., Chatterjee, S., Vernardos, G., Koopmans, L. V., Kleijn, G. V., Napolitano, N. R., Covone, G., Kelvin, L. S., and Hopkins, A. M. (2019). Testing convolutional neural networks for finding strong gravitational lenses in KiDS. *Monthly Notices of the Royal Astronomical Society*, 482(1):807–820.

Petulante, A., Berlind, A. A., Kelly Holley-Bockelmann, J., and Sinha, M. (2021). Machine learning the fates of dark matter subhaloes: A fuzzy crystal ball. *Monthly Notices of the Royal Astronomical Society*, 504(1):248–266.

Planck Collaboration, Ade, P. A. R., Aghanim, N., Armitage-Caplan, C., Arnaud, M., Ashdown, M., Atrio-Barandela, F., Aumont, J., Baccigalupi, C., Banday, A. J., Barreiro, R. B., Bartlett, J. G., Battaner, E., Benabed, K., Beno\^\it, A., Benoit-Lévy, A., Bernard, J.-P., Bersanelli, M., Bielewicz, P., Bobin, J., Bock, J. J., Bonaldi, A., Bond, J. R., Borrill, J., Bouchet, F. R., Bridges, M., Bucher, M., Burigana, C., Butler, R. C., Calabrese, E., Cappellini, B., Cardoso, J.-F., Catalano, A., Challinor, A., Chamballu, A., Chary, R.-R., Chen, X., Chiang, H. C., Chiang, L.-Y., Christensen, P. R., Church, S., Clements, D. L., Colombi, S., Colombo, L. P. L., Couchot, F., Coulais, A., Crill, B. P., Curto, A., Cuttaia, F., Danese, L., Davies, R. D., Davis, R. J., de Bernardis, P., de Rosa, A., de Zotti, G., Delabrouille, J., Delouis, J.-M., Désert, F.-X., Dickinson, C., Diego, J. M., Dolag, K., Dole, H., Donzelli, S., Doré, O., Douspis, M., Dunkley, J., Dupac, X., Efstathiou, G., Elsner, F., En\sslin, T. A., Eriksen, H. K., Finelli, F., Forni, O., Frailis, M., Fraisse, A. A., Franceschi, E., Gaier, T. C., Galeotta, S., Galli, S., Ganga, K., Giard, M., Giardino, G., Giraud-Héraud, Y., Gjerl\ow, E., González-Nuevo, J., Górski, K. M., Gratton, S., Gregorio, A., Gruppuso, A., Gudmundsson, J. E., Haissinski, J., Hamann, J., Hansen, F. K., Hanson, D., Harrison, D., Henrot-Versillé, S., Hernández-Monteagudo, C., Herranz, D., Hildebrandt, S. R., Hivon, E., Hobson, M., Holmes, W. A., Hornstrup, A., Hou, Z., Hovest, W., Huffenberger, K. M., Jaffe, A. H., Jaffe, T. R., Jewell, J., Jones, W. C., Juvela, M., Keihänen, E., Keskitalo, R., Kisner, T. S., Kneissl, R., Knoche, J., Knox, L., Kunz, M., Kurki-Suonio, H., Lagache, G., Lähteenmäki, A., Lamarre, J.-M., Lasenby, A., Lattanzi, M., Laureijs, R. J., Lawrence, C. R., Leach, S., Leahy, J. P., Leonardi, R., León-Tavares, J., Lesgourgues, J., Lewis, A., Liguori, M., Lilje, P. B., Linden-V\ornle, M., López-Caniego, M., Lubin, P. M., Mac\'\ias-Pérez, J. F., Maffei, B., Maino, D., Mandolesi, N., Maris, M., Marshall, D. J., Martin, P. G., Mart\'\inez-González, E., Masi, S., Massardi, M., Matarrese, S., Matthai, F., Mazzotta, P., Meinhold, P. R., Melchiorri, A., Melin, J.-B., Mendes, L., Menegoni, E., Mennella, A., Migliaccio, M., Millea, M., Mitra, S., Miville-Deschênes, M.-A., Moneti, A., Montier, L., Morgante, G., Mortlock, D., Moss, A., Munshi, D., Murphy, J. A., Naselsky, P., Nati, F., Natoli, P., Netterfield, C. B., N\orgaard-Nielsen, H. U., Noviello, F., Novikov, D., Novikov, I., O\'Dwyer, I. J., Osborne, S., Oxborrow, C. A., Paci, F., Pagano, L., Pajot, F., Paladini, R., Paoletti, D., Partridge, B., Pasian, F., Patanchon, G., Pearson, D., Pearson, T. J., Peiris, H. V., Perdereau, O., Perotto, L., Perrotta, F., Pettorino, V., Piacentini, F., Piat, M., Pierpaoli, E., Pietrobon, D., Plaszczynski, S., Platania, P., Pointecouteau, E., Polenta, G., Ponthieu, N., Popa, L., Poutanen, T., Pratt, G. W., Prézeau, G., Prunet, S., Puget, J.-L., Rachen, J. P., Reach, W. T., Rebolo, R., Reinecke, M., Remazeilles, M., Renault, C., Ricciardi, S., Riller, T., Ristorcelli, I., Rocha, G., Rosset, C., Roudier, G., Rowan-Robinson, M., Rubiño-Mart\'\in, J. A., Rusholme, B., Sandri, M., Santos, D., Savelainen, M., Savini, G., Scott, D., Seiffert, M. D., Shellard, E. P. S., Spencer, L. D., Starck, J.-L., Stolyarov, V., Stompor, R., Sudiwala, R., Sunyaev, R., Sureau, F., Sutton, D., Suur-Uski, A.-S., Sygnet, J.-F., Tauber, J. A., Tavagnacco, D., Terenzi, L., Toffolatti, L., Tomasi, M., Tristram, M., Tucci, M., Tuovinen, J., Türler, M., Umana, G., Valenziano, L., Valiviita, J., Van Tent, B., Vielva, P., Villa, F., Vittorio, N., Wade, L. A., Wandelt, B. D., Wehus, I. K., White, M., White, S. D. M., Wilkinson, A., Yvon, D., Zacchei, A., and Zonca, A. (2014). Planck 2013 results. XVI. Cosmological parameters. *A&A*, 571:A16.

Planck Collaboration, Aghanim, N., Akrami, Y., Ashdown, M., Aumont, J., Baccigalupi, C., Ballardini, M., Banday, A. J., Barreiro, R. B., Bartolo, N., Basak, S., Battye, R., Benabed, K., Bernard, J.-P., Bersanelli, M., Bielewicz, P., Bock, J. J., Bond, J. R., Borrill, J., Bouchet, F. R., Boulanger, F., Bucher, M., Burigana, C., Butler, R. C., Calabrese, E., Cardoso, J.-F., Carron, J., Challinor, A., Chiang, H. C., Chluba, J., Colombo, L. P. L., Combet, C., Contreras, D., Crill, B. P., Cuttaia, F., de Bernardis, P., de Zotti, G., Delabrouille, J., Delouis, J.-M., Di Valentino, E., Diego, J. M., Doré, O., Douspis, M., Ducout, A., Dupac, X., Dusini, S., Efstathiou, G., Elsner, F., En\sslin, T. A., Eriksen, H. K., Fantaye, Y., Farhang, M., Fergusson, J., Fernandez-Cobos, R., Finelli, F., Forastieri, F., Frailis, M., Fraisse, A. A., Franceschi, E., Frolov, A., Galeotta, S., Galli, S., Ganga, K., Génova-Santos, R. T., Gerbino, M., Ghosh, T., González-Nuevo, J., Górski, K. M., Gratton, S., Gruppuso, A., Gudmundsson, J. E., Hamann, J., Handley, W., Hansen, F. K., Herranz, D., Hildebrandt, S. R., Hivon, E., Huang, Z., Jaffe, A. H., Jones, W. C., Karakci, A., Keihänen, E., Keskitalo, R., Kiiveri, K., Kim, J., Kisner, T. S., Knox, L., Krachmalnicoff, N., Kunz, M., Kurki-Suonio, H., Lagache, G., Lamarre, J.-M., Lasenby, A., Lattanzi, M., Lawrence, C. R., Le Jeune, M., Lemos, P., Lesgourgues, J., Levrier, F., Lewis, A., Liguori, M., Lilje, P. B., Lilley, M., Lindholm, V., López-Caniego, M., Lubin, P. M., Ma, Y.-Z., Mac\'\ias-Pérez, J. F., Maggio, G., Maino, D., Mandolesi, N., Mangilli, A., Marcos-Caballero, A., Maris, M., Martin, P. G., Martinelli, M., Mart\'\inez-González, E., Matarrese, S., Mauri, N., McEwen, J. D., Meinhold, P. R., Melchiorri, A., Mennella, A., Migliaccio, M., Millea, M., Mitra, S., Miville-Deschênes, M.-A., Molinari, D., Montier, L., Morgante, G., Moss, A., Natoli, P., N\orgaard-Nielsen, H. U., Pagano, L., Paoletti, D., Partridge, B., Patanchon, G., Peiris, H. V., Perrotta, F., Pettorino, V., Piacentini, F., Polastri, L., Polenta, G., Puget, J.-L., Rachen, J. P., Reinecke, M., Remazeilles, M., Renzi, A., Rocha, G., Rosset, C., Roudier, G., Rubiño-Mart\'\in, J. A., Ruiz-Granados, B., Salvati, L., Sandri, M., Savelainen, M., Scott, D., Shellard, E. P. S., Sirignano, C., Sirri, G., Spencer, L. D., Sunyaev, R., Suur-Uski, A.-S., Tauber, J. A., Tavagnacco, D., Tenti, M., Toffolatti, L., Tomasi, M., Trombetti, T., Valenziano, L., Valiviita, J., Van Tent, B., Vibert, L., Vielva, P., Villa, F., Vittorio, N., Wandelt, B. D., Wehus, I. K., White, M., White, S. D. M., Zacchei, A., and Zonca, A. (2020). Planck 2018 results - VI. Cosmological parameters. *A&A*, 641:A6.

Press, W. H. and Schechter, P. (1974). Formation of Galaxies and Clusters of Galaxies by Self-Similar Gravitational Condensation. *\apj*, 187:425–438.

Ramanah, D. K., Charnock, T., and Lavaux, G. (2019). Painting halos from 3D dark matter fields using Wasserstein mapping networks.

Ramanah, D. K., Charnock, T., Villaescusa-Navarro, F., and Wandelt, B. D. (2020). Super-resolution emulator of cosmological simulations using deep physical models. *Monthly Notices of the Royal Astronomical Society*, 495(4):4227–4236.

Ramanah, D. K., Wojtak, R., and Arendse, N. (2021). Simulation-based inference of dynamical galaxy cluster masses with 3D convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, 501(3):4080–4091.

Reed, D., Governato, F., Quinn, T., Gardner, J., Stadel, J., and Lake, G. (2005). Dark matter subhaloes in numerical simulations. *Monthly Notices of the Royal Astronomical Society*, 359(4):1537–1548.

Richings, J., Frenk, C., Jenkins, A., Robertson, A., Fattahi, A., Grand, R. J. J., Navarro, J., Pakmor, R., Gomez, F. A., Marinacci, F., and Oman, K. A. (2020). Subhalo destruction in the Apostle and Auriga simulations. *Monthly Notices of the Royal Astronomical Society*, 492(4):5780–5793.

Rodríguez, A. C., Kacprzak, T., Lucchi, A., Amara, A., Sgier, R., Fluri, J., Hofmann, T., and Réfrégier, A. (2018). Fast cosmic web simulations with generative adversarial networks. *Computational Astrophysics and Cosmology*, 5(1):4.

Romano-Díaz, E., Shlosman, I., Heller, C., and Hoffman, Y. (2010). DISSECTING GALAXY FORMATION. II. COMPARING SUBSTRUCTURE IN PURE DARK MATTER AND BARYONIC MODELS. *The Astrophysical Journal*, 716(2):1095–1104.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing.

Rubin, V. C., Ford W.˜K., J., and Thonnard, N. (1980). Rotational properties of 21 SC galaxies with a large range of luminosities and radii, from NGC 4605 (R=4kpc) to UGC 2885 (R=122kpc). *apj*, 238:471–487.

Savary, E., Rojas, K., Maus, M., Clément, B., Courbin, F., Gavazzi, R., Chan, J. H. H., Lemon, C., Vernardos, G., Cañameras, R., Schuldt, S., Suyu, S. H., Cuillandre, J. C., Fabbro, S., Gwyn, S., Hudson, M. J., Kilbinger, M., Scott, D., and Stone, C. (2021). A search for galaxy-scale strong gravitational lenses in the Ultraviolet Near Infrared Optical Northern Survey (UNIONS).

Sawala, T., Pihajoki, P., Johansson, P. H., Frenk, C. S., Navarro, J. F., Oman, K. A., and White, S. D. M. (2017). Shaken and stirred: the Milky Way's dark substructures. *Monthly Notices of the Royal Astronomical Society*, 467(4):4383–4400.

Schanche, N., Cameron, A. C., Hébrard, G., Nielsen, L., Triaud, A. H., Almenara, J. M., Alsubai, K. A., Anderson, D. R., Armstrong, D. J., Barros, S. C., Bouchy, F., Boumis, P., Brown, D. J., Faedi, F., Hay, K., Hebb, L., Kiefer, F., Mancini, L., Maxted, P. F., Palle, E., Pollacco, D. L., Queloz, D., Smalley, B., Udry, S., West, R., and Wheatley, P. J. (2019a). Machine-learning approaches to exoplanet transit detection and candidate validation in wide-field ground-based surveys. *Monthly Notices of the Royal Astronomical Society*, 483(4):5534–5547.

Schanche, N., Cameron, A. C., Hébrard, G., Nielsen, L., Triaud, A. H. M. J., Almenara, J. M., Alsubai, K. A., Anderson, D. R., Armstrong, D. J., Barros, S. C. C., Bouchy, F., Boumis, P., Brown, D. J. A., Faedi, F., Hay, K., Hebb, L., Kiefer, F., Mancini, L., Maxted, P. F. L., Palle, E., Pollacco, D. L., Queloz, D., Smalley, B., Udry, S., West, R., and Wheatley, P. J. (2019b). Machine-learning approaches to exoplanet transit detection and candidate validation in wide-field ground-based surveys. *Monthly Notices of the Royal Astronomical Society*, 483(4):5534–5547.

Schaurecker, D., Li, Y., Tinker, J., Ho, S., and Refregier, A. (2021). Super-resolving Dark Matter Halos using Generative Deep Learning.

Schechter, P. (1976). An analytic expression for the luminosity function for galaxies. *apj*, 203:297–306.

Schuldt, S., Suyu, S. H., Cañameras, R., Taubenberger, S., Meinhard, T., Leal-Taixé, L., and Hsieh, B. C. (2021). Photometric redshift estimation with a convolutional neural network: NetZ. *Astronomy and Astrophysics*, 651(2016).

Scoccimarro, R. (1998). Transients from initial conditions: A perturbative analysis. *Monthly Notices of the Royal Astronomical Society*, 299(4):1097–1118.

Scoccimarro, R. and Sheth, R. K. (2002). PTHALOS: A fast method for generating mock galaxy distributions. *Monthly Notices of the Royal Astronomical Society*, 329(3):629–640.

Seljak, U. and Zaldarriaga, M. (1996). A Line-of-Sight Integration Approach to Cosmic Microwave Background Anisotropies. *The Astrophysical Journal*, 469:437.

Shallue, C. J. and Vanderburg, A. (2018). Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90. *The Astronomical Journal*, 155(2):94.

Sharp, N. A., Bonometto, S. A., and Lucchin, F. (1984). Higher order correlation functions in the Zwicky catalogue. *aap*, 130(1):79–85.

Shuntov, M., Pasquet, J., Arnouts, S., Ilbert, O., Treyer, M., Bertin, E., De La Torre, S., Dubois, Y., Fouchez, D., Kraljic, K., Laigle, C., Pichon, C., and Vibert, D. (2020). PhotoWeb redshift: Boosting photometric redshift accuracy with large spectroscopic surveys. *Astronomy and Astrophysics*, 636.

Simha, V. and Cole, S. (2017). Modelling galaxy merger time-scales and tidal destruction. *Monthly Notices of the Royal Astronomical Society*, 472(2):1392–1400.

Sinha, M. and Holley-Bockelmann, K. (2012). A FIRST LOOK AT GALAXY FLYBY INTERACTIONS. I. CHARACTERIZING THE FREQUENCY OF FLYBYS IN A COSMOLOGICAL CONTEXT. *The Astrophysical Journal*, 751(1):17.

Somerville, R. S., Hopkins, P. F., Cox, T. J., Robertson, B. E., and Hernquist, L. (2008). A semi-analytic model for the co-evolution of galaxies, black holes and active galactic nuclei. *Monthly Notices of the Royal Astronomical Society*, 391(2):481–506.

Soucail, G., Fort, B., Mellier, Y., and Picat, J. P. (1987). A blue ring-like structure in the center of the A 370 cluster of galaxies. *aap*, 172:L14–L16.

Spergel, D. N., Verde, L., Peiris, H. V., Komatsu, E., Nolta, M. R., Bennett, C. L., Halpern, M., Hinshaw, G., Jarosik, N., Kogut, A., Limon, M., Meyer, S. S., Page, L., Tucker, G. S., Weiland, J. L., Wollack, E., and Wright, E. L. (2003). First-Year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Determination of Cosmological Parameters. *The Astrophysical Journal Supplement Series*, 148(1):175–194.

Springel, V. (2005). The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society*, 364(4):1105–1134.

Springel, V., Frenk, C. S., and White, S. D. (2006). The large-scale structure of the Universe. *Nature*, 440(7088):1137–1144.

Springel, V., White, S. D. M., Tormen, G., and Kauffmann, G. (2001). Populating a cluster of galaxies - I. Results at \fontshape{it}{z}=0. *Monthly Notices of the Royal Astronomical Society*, 328(3):726–750.

Srisawat, C., Knebe, A., Pearce, F. R., Schneider, A., Thomas, P. A., Behroozi, P., Dolag, K., Elahi, P. J., Han, J., Helly, J., Jing, Y., Jung, I., Lee, J., Mao, Y.-Y., Onions, J., Rodriguez-Gomez, V., Tweed, D., and Yi, S. K. (2013). Sussing Merger Trees: The Merger Trees Comparison Project. *Monthly Notices of the Royal Astronomical Society*, 436(1):150–162.

Stirling, A. J. and Peacock, J. A. (1996). Power correlations in cosmology: limits on primordial non-Gaussian density fields. $\backslash$*mnras*, 283(4):L99–L104.

Su, Y., Zhang, Y., Liang, G., ZuHone, J. A., Barnes, D. J., Jacobs, N. B., Ntampaka, M., Forman, W. R., Nulsen, P. E., Kraft, R. P., and Jones, C. (2020). A deep learning view of the census of galaxy clusters in IllustrisTNG. *Monthly Notices of the Royal Astronomical Society*, 498(4):5620–5628.

Szewciw, A. O., Beltz-Mohrmann, G. D., Berlind, A. A., and Sinha, M. (2022). Toward Accurate Modeling of Galaxy Clustering on Small Scales: Constraining the Galaxy-halo Connection with Optimal Statistics. *The Astrophysical Journal*, 926(1):15.

Taffoni, G., Mayer, L., Colpi, M., and Governato, F. (2003). On the life and death of satellite haloes. *Monthly Notices of the Royal Astronomical Society*, 341(2):434–448.

Tamosiunas, A., Winther, H. A., Koyama, K., Bacon, D. J., Nichol, R. C., and Mawdsley, B. (2021). Investigating cosmological GAN emulators using latent space interpolation. *Monthly Notices of the Royal Astronomical Society*, 506(2):3049–3067.

Tang, H., Scaife, A. M. M., Wong, O. I., and Shabala, S. S. (2022). Radio Galaxy Zoo: giant radio galaxy classification using multidomain deep learning. *Monthly Notices of the Royal Astronomical Society*, 510(3):4504–4524.

Tassev, S., Zaldarriaga, M., and Eisenstein, D. J. (2013). Solving large scale structure in ten easy steps with COLA. *Journal of Cosmology and Astroparticle Physics*, 2013(6).

Taylor, J. E. and Babul, A. (2001). The Dynamics of Sinking Satellites around Disk Galaxies: A Poor Man's Alternative to High-Resolution Numerical Simulations. *The Astrophysical Journal*, 559(2):716–735.

Taylor, J. E. and Babul, A. (2004). The evolution of substructure in galaxy, group and cluster haloes - I. Basic dynamics. *Monthly Notices of the Royal Astronomical Society*, 348(3):811–830.

Taylor, J. E. and Babul, A. (2005). The evolution of substructure in galaxy, group and cluster haloes — II. Global properties. *Monthly Notices of the Royal Astronomical Society*, 364(2):515–534.

Tinker, J. L., Sheldon, E. S., Wechsler, R. H., Becker, M. R., Rozo, E., Zu, Y., Weinberg, D. H., Zehavi, I., Blanton, M. R., Busha, M. T., and Koester, B. P. (2012). Cosmological constraints from galaxy clustering and the mass-to-number ratio of galaxy clusters. *Astrophysical Journal*, 745(1).

Tormen, G., Diaferio, A., and Syer, D. (1998). Survival of substructure within dark matter haloes. *Monthly Notices of the Royal Astronomical Society*, 299(3):728–742.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). Learning spatiotemporal features with 3D convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:4489–4497.

Tröster, T., Ferguson, C., Harnois-Déraps, J., and McCarthy, I. G. (2019). Painting with baryons: Augmenting N-body simulations with gas using deep generative models. *Monthly Notices of the Royal Astronomical Society: Letters*, 487(1):L24–L29.

Tweed, D., Devriendt, J., Blaizot, J., Colombi, S., and Slyz, A. (2009). Building merger trees from cosmological N -body simulations. *Astronomy & Astrophysics*, 506(2):647–660.

Ullmo, M. and Aghanim, N. (2021). Adversarial Networks. *Computer Vision*, (2018):40–40.

van den Bosch, F. C. (2017). Dissecting the evolution of dark matter subhaloes in the Bolshoi simulation. *Monthly Notices of the Royal Astronomical Society*, 468(1):885–909.

van den Bosch, F. C. and Jiang, F. (2016). Statistics of dark matter substructure – II. Comparison of model with simulation results. *Monthly Notices of the Royal Astronomical Society*, 458(3):2870–2884.

van den Bosch, F. C., Lewis, G. F., Lake, G., and Stadel, J. (1999). Substructure in Dark Halos: Orbital Eccentricities and Dynamical Friction. *The Astrophysical Journal*, 515(1):50–68.

van den Bosch, F. C. and Ogiya, G. (2018). Dark matter substructure in numerical simulations: a tale of discreteness noise, runaway instabilities, and artificial disruption. *Monthly Notices of the Royal Astronomical Society*, 475(3):4066–4087.

van den Bosch, F. C., Ogiya, G., Hahn, O., and Burkert, A. (2018). Disruption of dark matter substructure: fact or fiction? *Monthly Notices of the Royal Astronomical Society*, 474(3):3043–3066.

van den Bosch, F. C., Tormen, G., and Giocoli, C. (2005). The mass function and average mass-loss rate of dark matter subhaloes. *Monthly Notices of the Royal Astronomical Society*, 359(3):1029–1040.

van Kampen, E. (2000). The survival of subhaloes in galaxies and galaxy clusters. *eprint arXiv:astro-ph/0008453*, (August).

Wang, L., Li, C., Kauffmann, G., and De Lucia, G. (2006). Modelling galaxy clustering in a high-resolution simulation of structure formation. *Monthly Notices of the Royal Astronomical Society*, 371(2):537–547.

Watson, D. F., Berlind, A. A., and Zentner, A. R. (2011). A COSMIC COINCIDENCE: THE POWER-LAW GALAXY CORRELATION FUNCTION. *The Astrophysical Journal*, 738(1):22.

Weinberg, M. D. (1989). Self-gravitating response of a spherical galaxy to sinking satellites. *Monthly Notices of the Royal Astronomical Society*, 239(2):549–569.

Wetzel, A. R. (2011). On the orbits of infalling satellite haloes. *Monthly Notices of the Royal Astronomical Society*, 412(1):49–58.

Wetzel, A. R., Cohn, J. D., and White, M. (2009). Simulating subhaloes at high redshift: merger rates, counts and types. *Monthly Notices of the Royal Astronomical Society*, 395(3):1376–1390.

Wetzel, A. R. and White, M. (2010). What determines satellite galaxy disruption? *Monthly Notices of the Royal Astronomical Society*, 403(2):1072–1088.

White, S. D. M. (1979). The hierarchy of correlation functions and its relation to other measures of galaxy clustering. *\mnras*, 186:145–154.

White, S. D. M. and Frenk, C. S. (1991). Galaxy Formation through Hierarchical Clustering. *\apj*, 379:52.

Wilde, J., Serjeant, S., Bromley, J. M., Dickinson, H., Koopmans, L. V. E., and Metcalf, R. B. (2022). Detecting gravitational lenses using machine learning: exploring interpretability and sensitivity to rare lensing configurations. *Monthly Notices of the Royal Astronomical Society*, 512(3):3464–3479.

Wu, C., Wong, O. I., Rudnick, L., Shabala, S. S., Alger, M. J., Banfield, J. K., Ong, C. S., White, S. V., Garon, A. F., Norris, R. P., Andernach, H., Tate, J., Lukic, V., Tang, H., Schawinski, K., and Diakogiannis, F. I. (2019). Radio Galaxy Zoo: CLARAN - A deep learning classifier for radio morphologies. *Monthly Notices of the Royal Astronomical Society*, 482(1):1211–1230.

Wu, Z., Zhang, Z., Pan, S., Miao, H., Luo, X., Wang, X., Sabiu, C. G., Forero-Romero, J., Wang, Y., and Li, X.-D. (2021). Cosmic Velocity Field Reconstruction Using AI. *The Astrophysical Journal*, 913(1):2.

Yan, Z., Mead, A. J., Van Waerbeke, L., Hinshaw, G., and McCarthy, I. G. (2020). Galaxy cluster mass estimation with deep learning and hydrodynamical simulations. *Monthly Notices of the Royal Astronomical Society*, 499(3):3445–3458.

Yip, J. H. T., Zhang, X., Wang, Y., Zhang, W., Sun, Y., Contardo, G., Villaescusa-Navarro, F., He, S., Genel, S., and Ho, S. (2019). From Dark Matter to Galaxies with Convolutional Neural Networks.

York, D. G., Adelman, J., John E. Anderson, J., Anderson, S. F., Annis, J., Bahcall, N. A., Bakken, J. A., Barkhouser, R., Bastian, S., Berman, E., Boroski, W. N., Bracker, S., Briegel, C., Briggs, J. W., Brinkmann, J., Brunner, R., Burles, S., Carey, L., Carr, M. A., Castander, F. J., Chen, B., Colestock, P. L., Connolly, A. J., Crocker, J. H., Csabai, I., Czarapata, P. C., Davis, J. E., Doi, M., Dombeck, T., Eisenstein, D., Ellman, N., Elms, B. R., Evans, M. L., Fan, X., Federwitz, G. R., Fiscelli, L., Friedman, S., Frieman, J. A., Fukugita, M., Gillespie, B., Gunn, J. E., Gurbani, V. K., de Haas, E., Haldeman, M., Harris, F. H., Hayes, J., Heckman, T. M., Hennessy, G. S., Hindsley, R. B., Holm, S., Holmgren, D. J., Huang, C.-h., Hull, C., Husby, D., Ichikawa, S.-I., Ichikawa, T., Ivezić,

, Kent, S., Kim, R. S. J., Kinney, E., Klaene, M., Kleinman, A. N., Kleinman, S., Knapp, G. R., Korienek, J., Kron, R. G., Kunszt, P. Z., Lamb, D. Q., Lee, B., Leger, R. F., Limmongkol, S., Lindenmeyer, C., Long, D. C., Loomis, C., Loveday, J., Lucinio, R., Lupton, R. H., MacKinnon, B., Mannery, E. J., Mantsch, P. M., Margon, B., McGehee, P., McKay, T. A., Meiksin, A., Merelli, A., Monet, D. G., Munn, J. A., Narayanan, V. K., Nash, T., Neilsen, E., Neswold, R., Newberg, H. J., Nichol, R. C., Nicinski, T., Nonino, M., Okada, N., Okamura, S., Ostriker, J. P., Owen, R., Pauls, A. G., Peoples, J., Peterson, R. L., Petravick, D., Pier, J. R., Pope, A., Pordes, R., Prosapio, A., Rechenmacher, R., Quinn, T. R., Richards, G. T., Richmond, M. W., Rivetta, C. H., Rockosi, C. M., Ruthmansdorfer, K., Sandford, D., Schlegel, D. J., Schneider, D. P., Sekiguchi, M., Sergey, G., Shimasaku, K., Siegmund, W. A., Smee, S., Smith, J. A., Snedden, S., Stone, R., Stoughton, C., Strauss, M. A., Stubbs, C., SubbaRao, M., Szalay, A. S., Szapudi, I., Szokoly, G. P., Thakar, A. R., Tremonti, C., Tucker, D. L., Uomoto, A., Berk, D. V., Vogeley, M. S., Waddell, P., Wang, S.-i., Watanabe, M., Weinberg, D. H., Yanny, B., and Yasuda, N. (2000). The Sloan Digital Sky Survey: Technical Summary. *The Astronomical Journal*, 120(3):1579–1587.

Zaldarriaga, M. and Seljak, U. (2000). CMBFAST for Spatially Closed Universes. *The Astrophysical Journal Supplement Series*, 129(2):431–434.

Zaldarriaga, M., Seljak, U., and Bertschinger, E. (1998). Integral Solution for the Microwave Background Anisotropies in Nonflat Universes. *The Astrophysical Journal*, 494(2):491–502.

Zel'dovich, Y. B. (1970). Gravitational instability: An approximate theory for large density perturbations. \aap, 5:84–89.

Zentner, A. R., Berlind, A. A., Bullock, J. S., Kravtsov, A. V., and Wechsler, R. H. (2005). The Physics of Galaxy Clustering. I. A Model for Subhalo Populations. *The Astrophysical Journal*, 624(2):505–525.

Zentner, A. R. and Bullock, J. S. (2003). Halo Substructure and the Power Spectrum. *The Astrophysical Journal*, 598(1):49–72.

Zhang, Z., Zou, Z., Li, N., and Chen, Y. (2022). Classifying Galaxy Morphologies with Few-shot Learning. *Research in Astronomy and Astrophysics*, 22(5):1–23.

Zwicky, F. (1933). Die Rotverschiebung von extragalaktischen Nebeln. *Helvetica Physica Acta*, 6:110–127.