Estimating the Redshift of Galaxies Using Image-Based Regression Techniques

Kim Graatrud, Simon Silverstein, and Nicholas Andrés Tran Rodriguez

Universitetet i Oslo

The widely accepted model for modern cosmology is the ΛCDM, but despite its success there is no consensus on the exact nature of dark matter or dark energy. Further improvements to the models need more data from the recent history of the universe, including the redshift of millions celestial bodies, where spectroscopy cannot keep up. We explore machine learning techniques for estimating redshift including convolutional neural networks, trees, and gradient boosted trees. A hybrid of both methods is also developed by first training our models through a convolutional neural networks and thereafter feeding its output to regression trees. In training our models we use the dataset GalaxiesML, which contains images of over 250 thousand galaxies in 5 different wavelength bands and their corresponding redshift. Compiled from multiple different astronomical surveys, GalaxiesML is the ideal dataset for developing methods of redshift estimation. We found that we could achieve close to a pure CNN performance using a hybrid CNN-gradient-boost model, which achieved a $R^2 = 0.814$ as opposed to the CNNs at a fraction of the computational cost. Our best model overall was a large CNN which achieved $R^2 = 0.876$, demonstrating potential for ML-based redshift estimation.

CONTENTS

I. INTRODUCTION

The principle prediction of modern cosmology is the distribution and evolution of matter and energy in our universe. The current best and most widely accepted model for this is the ΛCDM, which describes a universe dominated by a cosmological constant Λ and cold dark matter. Though decades of research have validated the ΛCDM across varying domains, no consensus has been reached on the exact nature of dark energy or dark matter. Worse still, constraints on the Hubble constant, H_0, from cosmic microwave background (CMB) experiments and from astronomical surveys are in 5.9-σ disagreement (Valentino, 2021). Resolving this disagreement and shedding light on dark energy is a major ongoing scientific endeavor.

Since the ΛCDM attempts to describe the evolution of the universe, it must be tested against accurate data from many different periods of the universe's history. The CMB provides us with data from early on, and has been precisely measured by experiments such as COBE, WMAP, and Planck (Fixsen *et al.* (1994); Komatsu *et al.* (2011); Planck Collaboration *et al.* (2020)). Data for the recent history of the universe has been harder to come by, since it requires precisely mapping the location and redshift of millions of celestial objects. A galaxy's redshift z is a measure of how its light has changed wavelength due to the expansion of the universe. In effect, z measures how far away a galaxy is, and thus how far in the past we see it. Ever since Vesto Silpher measured the redshift of a neighboring nebula using spectroscopy in 1912, spectrographs have been the tool for doing so, measuring the shift in a body's emission lines one object at a time. This has recently become more feasible with the success of the Dark Energy Spectroscopic Instrument (DESI) (DESI Collaboration *et al.*, 2025). Even so, the process remains costly and time consuming.

Since optical telescopes are capable of imaging many thousands of galaxies at once, an accurate method of calculating redshift from images would result in a significant increase in available cosmological data. We explore the possibility of doing so using machine learning in the form of convolutional neural networks and decision trees. Decision trees are interpretable methods which allow for . Convolutional neural networks (CNN's) have seen significant development since the beginning of the 21st century, and are now a go-to tool for many computer vision problems.

In Sec. II.A we explain the composition and goal of the dataset we have chosen to study. In Sec. II.B, II.C, and II.E we introduce the theory and methodology we

will use. And finally present our results and discuss in Sec. III and conclude in Sec. IV.

In this article, we will explore the use of decision trees, gradient boosting, and CNNs to try and accurately predict galaxy's redshift from it's optical image. Also, a hybrid of both CNNs and gradient boosting will be developed and tested. All methods will be evaluated by how they perform in estimating the of redshift of galaxies using the models' R^2 score. A breakdown on the time it takes to train and run the models will also be presented.

All code made for the production of the results for this paper can be found at our *GitHub* repository[1].

II. METHODS AND THEORY

A. The Dataset

All our models are trained, validated, and tested on the *GalaxiesML* dataset (Do *et al.*, 2024)[2]. GalaxiesML is split into 204,573 training, 40,914 testing, and 40,914 validation galaxies. Each galaxy has five 64×64 pixel images corresponding to intensity in the g, r, i, z, and y frequency bands (see Figure 1). Each galaxy also has a spectrographically-measured redshift in the range $0 < z < 4$ which we treat as the ground truth. Though most of the galaxies are similar in appearance, the dataset is not free of problems that can make training difficult. Figure 2 shows a few examples of the unusual and erroneous images our models must contend with.

Figure 3 shows that the GalaxiesML dataset is also dominated by low-z samples, with an order of magnitude more galaxies between $0 < z < 1$ than with $z > 1$. This is an unavoidable result of experimental observation: the further away an object is, the harder it is to detect. Because of the reduced training data, we expect all our models to perform worse at higher z.

To account for differences in intensity between bands and reduce overall bias, we scaled the dataset by subtracting each channel's mean and dividing by its standard deviation. We computed these from the training dataset and applied them to the testing and validation sets as well.

We trained all machine learning methods on the training dataset. We performed model selection with the validation set, and derived final results on the test set.

Throughout the report, we compare models based on the coefficient of determination R^2 of their predictions vs the target values.

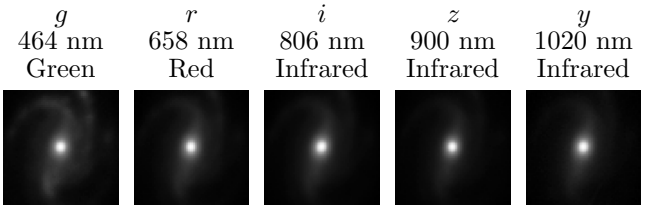

FIG. 1 A typical galaxy and each of its five channels. The center of each frequency band is included. Pictured galaxy has redshift $z = 0.13$.

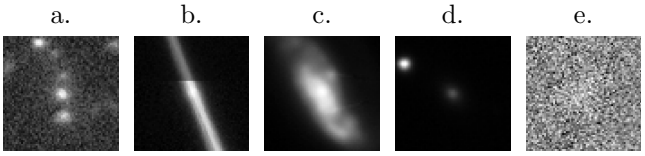

FIG. 2 A subset of the issues a particular channel or galaxy may have, including (a) multiple objects in the frame, (b) digital / experimental artifacts, (c) unusual size and brightness, (d) misalignment, and (e) overwhelming noise.

B. Regression Trees

As the name suggests, regression trees are regression methods belonging to the tree family of methods. Tree methods take the input dataset and splits the set into regions based on a criterion defined by the task at hand. This process is applied recursively to each split batch, each with different criterion, making a tree shape that can be seen in Fig. 4. Here each subsequent split divides the data w.r.t the criterion given.

The number of times the set has been split is referred to as *depth*. This depth directly corresponds to the complexity of the model, low depth meaning high bias and low variance, while high depth means low bias and high variance. Similar to other machine learning methods, high variance poses a problem for the generalization of the model, as it may simply overfit its training set. We will fit a shallow tree and a very deep tree to explicitly show this, it will also serve as the baseline for the more advanced methods. These trees will be built using using the squared error as the loss function for each split. More correctly the trees splits its input space into J_m disjoint regions $\{R_{jm}\}_{j=1}^{J_m}$, (Hastie *et al.*, 2009) goes more in-depth about input spaces and the mathematical dynamics of trees.

To combat high variance in the model there exists many variance-reducing methods such as *bagging*, and *random-forests* which use averaging and de-correlation to achieve lower variance. We will limit our attention to *Gradient boosting*.

[1] https://github.com/KimGraatrud/FYS-STK4155-Projects/tree/main/Project_3

[2] Available for download at https://zenodo.org/records/11117528. Note that the 127×127 pixel image datasets have impractically large file sizes; for this reason, we only use the 64×64 datasets.

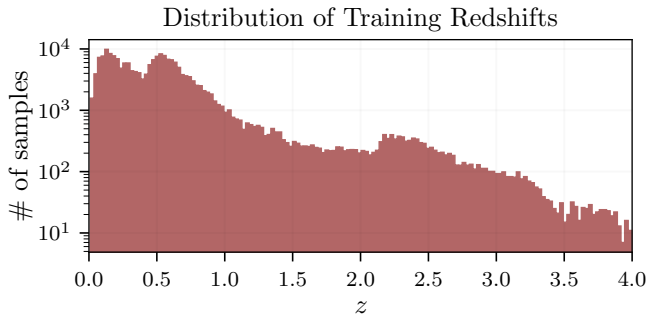

FIG. 3 Histogram showing the distribution in redshift of the 204,573 training samples. The testing and validation datasets have proportional distributions since they were derived from the same original dataset.

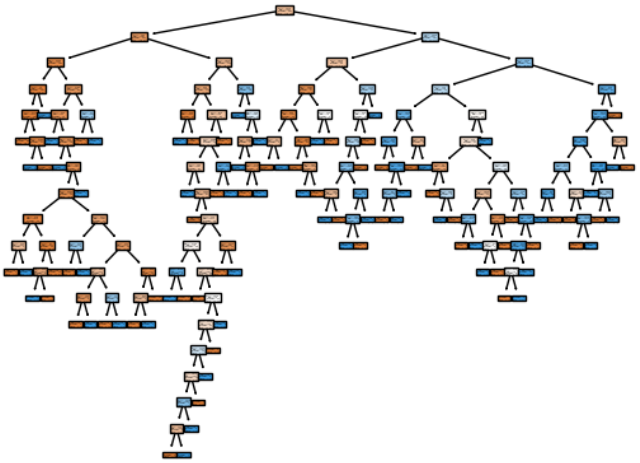

FIG. 4 Example of a typical tree structure. A classifier tree example made on real data, the *Pima Indians dataset*. Plotted using Scikit-learn's tree class.

C. Gradient Boosting

Gradient boosting is the process of using many smaller trees to learn some part of the data and eventually add them together to create a strong prediction. The boosting procedure is as follows; let

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)),$$

where $f(x)$ is the sum of regression trees, and we wish to minimize the loss function $L(f)$, yielding

$$\boldsymbol{f} = \sum_{m=0}^{M} \boldsymbol{h}_m, \quad \boldsymbol{h}_m \in \mathcal{R}^N.$$

Here M is the number of trees, and $\boldsymbol{h}$ are *step* vectors with $\boldsymbol{h}_0$ as an initial guess. The step vectors' formula usually takes the form of

$$\boldsymbol{h}_m = \rho_m(-\boldsymbol{g}_m),$$

where ρ_m is called the step length and is the solution of

$$\rho_m = \arg\min_{\rho} L(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m),$$

and $-\boldsymbol{g}_m$ is the negative gradient

$$-\boldsymbol{g}_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i}\right]_{f(x_i)=f_{m-1}(x_i)},$$
$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i}\right]_{f(x_i)=f_{m-1}(x_i)} \Big|_{f=f_{m-1}}.$$

Where we call r_{im} a pseudo residual. Then modeling the predictor as an additive expansion of trees:

$$f_M(x) = f_0 + \sum_{m=1}^{M} \eta h_m(x),$$

where η is the learning rate and f_0 is some initial constant which minimizes the loss. Just as we did for the regression trees, we will use the squared error as the loss function for our gradient boosting models. Resulting in our pseudo residuals reducing to

$$r_{im} = y_i - f_{m-1}(x_i),$$

at each iteration our trees are then obtained by solving

$$h_m = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{N} (r_{im} - h(x_i))^2.$$

The model then gets updated in a gradient descent step. This has been summarized in Alg. 1, inspired by the algorithm found in (Hastie *et al.*, 2009).

Its normal to constrain all the trees to be of the same maximum depth. The tuning parameters for gradient boosting is generally, the learning rate, the number of trees M, the L2 regularization parameter λ, the maximum number of leaves, and the minimum number of samples per leaf. We will vary these to find the best model.

D. Gradient Boosting on Images

Trees preform best when given data with features that are meaningful for the overall prediction its trying to make. In images each pixel is not meaningful on its own, only when studied in conjunction with the other pixels around it the image becomes meaningful. Trees, or even gradient boosting have no way of knowing the connection pixels have between each other. They have no way of seeing the edges, shapes or trends in a picture, since it treats each pixel individually. Its therefore very expected of trees and gradient boosting to be outperformed on this type of data. There are many ways of extracting features from images, though that is outside the scope of this paper.

Algorithm 1 Gradient Boosting for Regression

1. **Initialization.**
   Choose the squared error loss

$$L(y_i, f(x_i)) = \frac{1}{2}\big(y_i - f(x_i)\big)^2.$$

   Initialize the model with the constant minimizer of the loss:

$$f^{(0)}(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c) = \frac{1}{N}\sum_{i=1}^{N} y_i.$$

2. **For** $m = 1$ **to** M**:**

   (a) **Compute negative gradients (pseudo-residuals).**
   For each observation $i = 1, \ldots, N$:

$$r_{im} \leftarrow -\left.\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right|_{f(x_i)=f^{(m-1)}(x_i)} = y_i - f^{(m-1)}(x_i).$$

   (b) **Fit a regression tree.**
   Fit a regression tree $T_m(x)$ to the targets

$$\{r_{im}\}_{i=1}^{N},$$

   producing terminal regions

$$R_{jm}, \qquad j = 1, \ldots, J_m.$$

   (c) **Compute terminal-node values.**
   For each region R_{jm}, compute

$$\gamma_{jm} \leftarrow \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\Big(y_i,\ f^{(m-1)}(x_i) + \gamma\Big)$$
$$= \frac{1}{|R_{jm}|} \sum_{x_i \in R_{jm}} r_{im}.$$

   (d) **Update the model.**

$$f^{(m)}(x) = f^{(m-1)}(x) + \eta \sum_{j=1}^{J_m} \gamma_{jm}\, \mathbf{1}(x \in R_{jm}),$$

   where $\eta \in (0, 1]$ is the learning rate.

3. **Prediction.**
   For a new input x, predict

$$\hat{y}(x) = f^{(M)}(x).$$

Convolutional Neural Networks introduced in Sec. II.E, use kernel operations to reduce inputs and better learn the relationships between pixels, we will attempt to make the convolutional neural network feature extract and then take the data it extracted before it makes its prediction, then use as input that to train and test a regression tree and gradient boosting.

We have make such a pipeline and use compare it to the tree and gradient boosting models, as-well as the native CNNs models. The pipeline consists of a CNN which takes the input, trains a little on it, in turn transforming it to fewer features by using its pooling layers. This transformed data is then used to train and predict using a gradient boosted model, which gives us our final prediction. We then study the model performance and the computational cost in relation to the other methods used.

E. Convolutional Neural Networks

We also explore regression using convolutional neural networks (CNNs). Whereas a simple fully-connected network would require one feature for each of an image's pixel values, CNNs employ a kernel operation in at least one layer, reducing total inputs and exploiting the relationship between nearby pixels. Figure 5 diagrams the architecture of one such network.

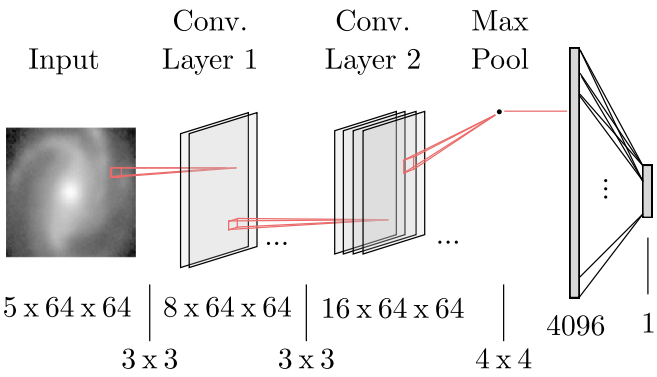

FIG. 5 Diagram of a CNN with two convolutional layers, one max pooling, and one fully connected hidden layer. Throughout this report it is implied that an activation function follows each layer.

The kernel, or filter, is defined as a small grid of weights on a small patch, with dimensions $H' \times W'$, of the image; where it slides through each column representing a width W and through each row representing a height H. As it is assumed that similar patterns may appear in each image for each placement, the kernel takes the inner product with the image, where each inner product contains the same kernel. In this paper, the redshift of various galaxies will be determined through photo-z. Therefore, the weights calculated for each kernel are based on each of the five wavelength bands: 'g', 464 nm, 'r', 658 nm, 'i', 806 nm, 'z', 900 nm, and 'y', 1020 nm. In total, they form a 3-tensor of height H, width W, and total kernels C. Thereafter, the output will be applied to an activation function. As we want to design the architecture of the CNN with multiple layers, we first pool the output that is applied to the activation function. There are various methods in pooling the feature map, in this paper, we will use average pooling, where it defines a kernel of

size $n \times n$, which will map the whole feature map. For each kernel overlapping the feature map, the average is calculated and represented in a new reduced feature map. This is an important step in CNNs as it reduces the computation time between each layer. This general algorithm continues until the last layer. As we want from linear regression, we flatten the resulting feature map and use a linear activation function.

Between depth, kernel size, number of channels, and pooling, there are an overwhelming variety of shapes a CNN may take on. To limit the scope of our exploration, we fixed a number of hyper-parameters across all our models. For updating parameters, we use the *ADAM* gradient descent method with $\rho_1 = 0.9$ and $\rho_2 = 0.999$ (Hastie *et al.*, 2009). This was motivated by our previous work, which found this variable-learning-rate method to work well (Graatrud *et al.*, 2025). The same work showed *LeakyRELU* to be an efficient and effective activation function which avoided the problem of dead nodes; we used it after every hidden layer. We then explore different sizes of convolutional networks across two different overarching architectures, which we note 'deep' and 'wide.'

For deep models, we increased complexity primarily by adding more layers, often with multiple layers having the same smaller number of channels. All deep models used 3×3 convolutional kernels with a stride of 1 and padding of 1.

For wide models, we increased complexity by doubling the number of channels with each additional layer. We used 5×5 convolutional kernels for the smaller models, and 7×7 kernels for the larger. The specific shape of each of the twelve tested models is given in Appendix A.

After the different convolutional layers, all models performed max pooling with a 4×4 kernel with stride 4. Values were then flattened and passed through one final fully connected layer, producing a single output which estimated redshift.

The one hyperparameter we did vary was initial learning rate η, which was shown in a previous project to have ideal values which were dependent on application (Graatrud *et al.*, 2025).

As a demonstration of the convolution process, Fig. 6 shows the output of each layer of the smallest 'deep' convolutional network for a single sample.

F. Tools

During the production of results for this paper we have used the following tools[3]:

- *Numpy* - Numerical computing and efficient array operations (Harris *et al.*, 2020).
- *Matplotlib* - Visualization and plotting of data (Hunter, 2007).
- *PyTorch* - High-Performance Deep Learning Library (Paszke *et al.*, 2019).
- *Scikit-learn* - Machine learning algorithms and model utilities (Pedregosa *et al.*, 2011).

Scikit-learn has been used for its utilities and tree based methods. PyTorch has been used for its fast and reliable neural network methods, especially it's CNN implementation. Matplotlib has been used to create all the figures seen in this paper. Numpy has allowed for efficient storage and access of numbers.

For the purposes of timing comparison, all neural networks were trained on a NVIDIA GeForce RTX 4070 with 7168 CUDA cores and 12 GB of VRAM. The trees were trained on an AMD Ryzen 9 7900 4.0/5.2 GHz clock speed CPU. Our *PyTorch* package was compiled with the *CUDA* option enabled.

III. RESULTS AND DISCUSSION

Firstly we trained the tree-based models on the pixel data. Their performance can be seen in Fig. 7, where we have quantified each tree's performance its R^2 on the testing dataset. 'Dummy' in this case is a regressor which achieves an R^2 of 0, meaning it explains none of the variance in the dataset. It is used as a baseline against which to compare the relative performance of each model.

From Fig. 7 we clearly observe that the shallow tree and the deep tree both significantly outperform the dummy regressor. This of course means that both models learned meaningful predictors or structure from the images. The shallow tree outperformed the deep tree when predicting the test dataset; this is supported by the raw metrics in Table I. Here we observe that the deep tree had a $R^2 = 1$ and RMSE $= 0$ on its own training set, leading us to conclude that the difference in performance is due to the deep tree overfitting to its training set. This is expected as deeper trees have low bias and high variance. The shallow and deep trees had *depth*s of 8 and 204, 234 respectively. We expected our shallow tree, which is known to have high bias and low variance, to have the same hampered performance as the deep tree. However, it ended up performing better than the overfitted tree. We believe this is a case of us mistakenly thinking that a depth of 8 was shallow enough to underfit the data. We set out to demonstrate the bias-variance within our trees, but we have missing precision to do so. We elaborate further in Sec. IV.A.

Again looking at Fig. 7, we observe that the gradient boosted models had the highest calculated R^2 compared to the other tree methods, with the hyperparameter-tuned model being slightly more generalizable. The gradient boosted models were expected to perform the best

[3] This project was completed without the aid of any LLM.

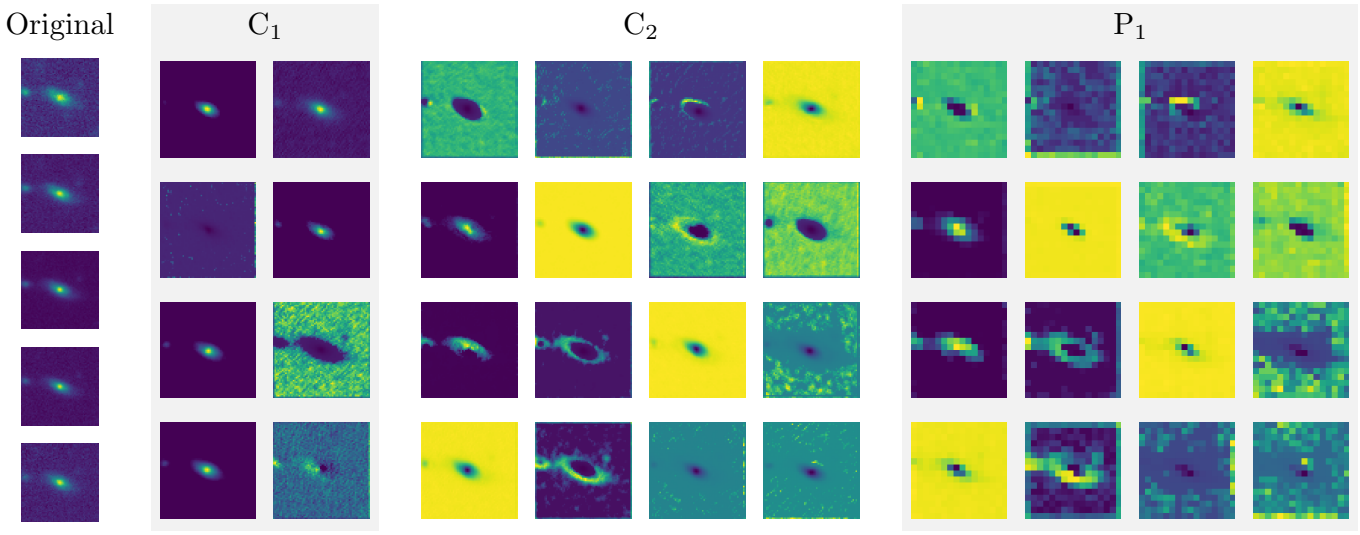

FIG. 6 Demonstration of the convolutional layer outputs for a small model (specifically 'd1,' see Appendix A).

TABLE I Performance per tree model given in RMSE and R^2. The models are as defined in Fig. 7.

Model	Train RMSE	Train R^2	Test RMSE	Test R^2
Dummy	0.573	0.000	0.565	0.000
Shallow	0.338	0.652	0.337	0.644
Deep	0.000	1.000	0.374	0.560
GB	0.222	0.850	0.254	0.797
Tuned GB	0.188	0.892	0.249	0.805
Hybrid GB	0.223	0.850	0.244	0.814

because of their ability to become very complex while not overfitting to the data. The hybrid gradient-boosting method preformed the best. With an $R^2 = 0.814$ it explained the variance in the test dataset better than the other tree methods. The difference in performance between the hybrid and tuned gradient boost models suggests that the act of encoding the image into fewer features helped the performance of the gradient boosted model.

TABLE II The time it took to train and predict using each each model given in minutes. Each model is defined as described in Fig. 7. Here the training time of the hybrid model is the combined training time of the CNN and the gradient boost and predict time is the time it took to extract the features of the test-dataset and predict using the gradient boost model.

Model	Train time	Predict time
Dummy	0.001	0.000
Shallow	23.38	0.008
Deep	124.943	0.007
GB	6.582	0.077
Tuned GB	9.047	0.108
Hybrid GB	3.098	0.388

What makes this result even more promising is the reduction in training time that came with the hybrid model. Whereas the tuned gradient boosted model took 9.047 minutes to train, the hybrid model took only 3.098 minutes for both the CNN and the gradient-boosted tree. The tradeoff with this is a marginally higher prediction time and a hardware requirement. Out of the ~ 3 minutes combined it took to train the CNN and the gradient-boosted tree, only 0.554 minutes of it was the tree. This highlights how tree based models perform well on fewer features, which can be provided by the use of an CNN.

Fig. 8 shows the performance of pure CNNs for different architectures and sizes. R^2 was calculated from predictions of the test dataset. Across both deep and wide architectures, R^2 increased with number of trainable parameters. Interestingly, this increase plateaued quickly, with the difference between the smallest ($R^2 = 0.747$) and largest ($R^2 = 0.876$) models not being dramatic. Notably, the different architectures did not seem to meaningfully change the result of the training. If we had a 'deep' model with number of trainable parameters as the largest 'wide' model we assume that it would preform about the same. The reason for our models' performance plateauing is currently not known. We speculate that our pure CNN models are not properly tuned to gain the best performance on this dataset. One thought is that our largest models over-fitted to our training data and that we did not have enough computational power to train larger models and could only observe the first barely over-fitted model. Another possibility is that we have reached the noise floor for this dataset and cannot gain any more performance. Again we are limited by computational power and time to explore the possibilities.

Fig. 8 also shows the training time for each model. Training time increases exponentially with number of

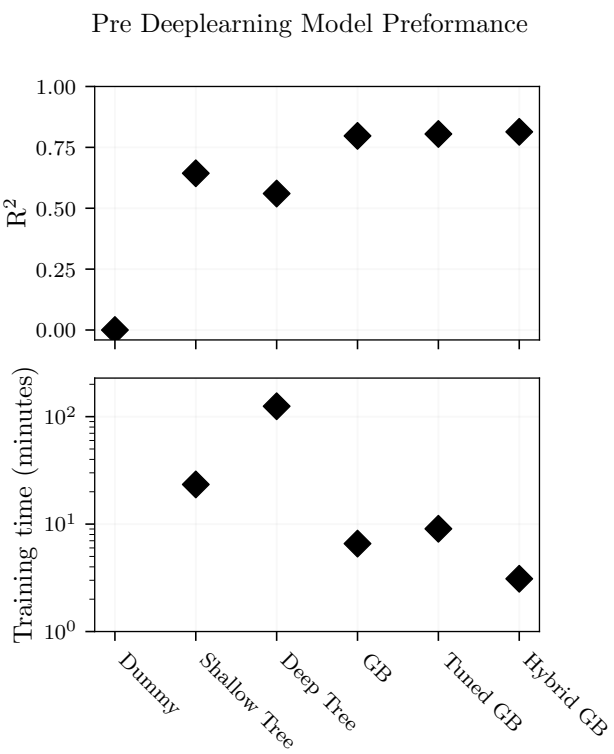

FIG. 7 Shows five different models trained on the same dataset and predicting the same test set. The models are 'dummy tree' which does not learn anything from the data, a 'shallow tree' which had its *depth* constrained at 8, a 'deep tree' which was allowed to grow as deep as it can, 'GB' which stands for gradient boosting and is not hyperparameter-tuned, 'tuned GB' which is a hyperparameter-tuned model with L2 = 0.04642, 63 leaves, and minimum 5 samples per leaf. The performance is quantified in the R^2 metric, with the corresponding values are reported in Table I. The time it took to train each model is also included in the bottom part of the figure, here dummy is excluded due to its near nonexistent training time.

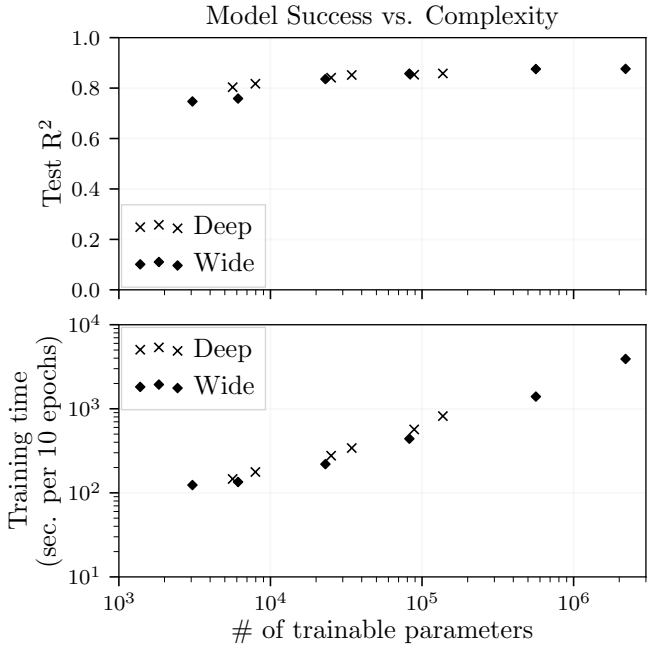

FIG. 8 Shows the success of different sized CNNs. Crosses (Deep) have more layers, but fewer channels per layer while diamonds (Wide) have more channels, but fewer layers. The top graph shows the R^2 of each model as calculated from predictions of the test dataset, while the bottom graph shows the training time in seconds per 10 epochs. All networks were trained for a total of 15 epochs with a batch size of 256 and an initial learning rate of $\eta = 2 \times 10^{-4}$. The specific shapes of all models may be found in Appendix A.

trainable parameters (note the logarithmic scale for the axes). This relationship is primarily what limited us from exploring more and larger models. Since training time increases significantly faster with model complexity than R^2 does, smaller models are leagues more efficient for the architectures explored here. Therefore it would be worthwhile to chose a smaller model and tune it properly.

Fig. 9 shows the training behavior of the smallest 'deep' model for four different learning rates. As Graatrud *et al.* (2025) showed, the largest learning rates learned faster but were significantly more unstable. Motivated by these findings, we trained all CNNs with learning rate $\eta = 2 \times 10^{-4}$.

Figures 12, 10, and 11 show how our smallest CNN, best CNN, and hybrid model predicted redshift as compared to the target values. The black dotted line across the plots represents a perfect model with prediction equal to target.

The smallest model (Fig. 10) performs poorly across all redshifts, and does especially bad for larger values ($z > 1.2$). We include it as a reference against which we may judge the larger models.

The hybrid model (Fig. 11) performs better. Notice that the distribution at small redshift ($z < 1$) is more constrained than it was for the smallest model. However, this model still struggles at high z. This is most likely because of the comparatively fewer training samples we have at these z values. Considering Figure 3, we expect our networks do much better at lower z.

The best CNN (Fig. 12) has by far the tightest prediction spread, demonstrating decent low-z success and even successfully predicting some of the highest redshift samples. Again, this model also has trouble predicting redshift for galaxies with $z \approx 3$ due to the limited training samples.

These methods show promise, but would need to much more accurately predict redshift to be usable for cosmological data purposes.

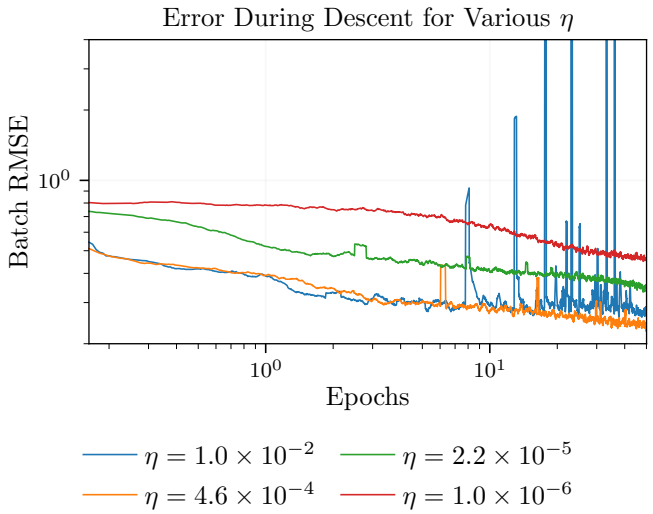

FIG. 9 Shows the batch RMSE of the smallest deep CNN network (specifically 'd1,' see Appendix A) as a function of the epochs the model trained over. Four different learning rates η were tested. A rolling average with a window of size 50 was applied to each.

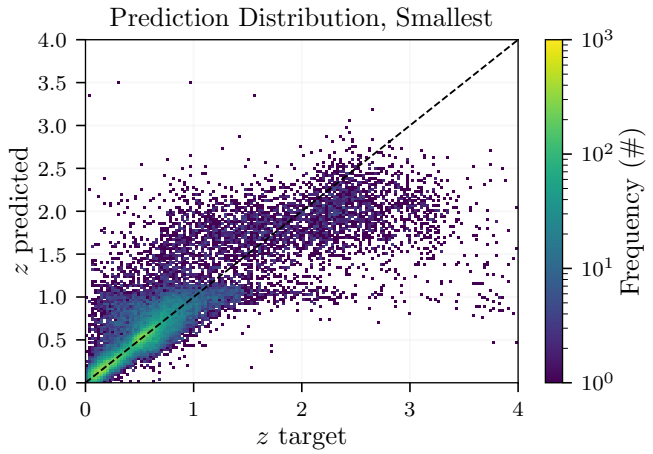

FIG. 10 Shows a 2D histogram of the the predicted redshifts from our worst performing CNN ('w1' from Appendix A, which was also the model with the smallest number of trainable parameters) compared to the real redshifts. The black dotted line represent where the predicted redshift is equal to the real redshift. The color bar represents the frequency of target to predicted redshifts.

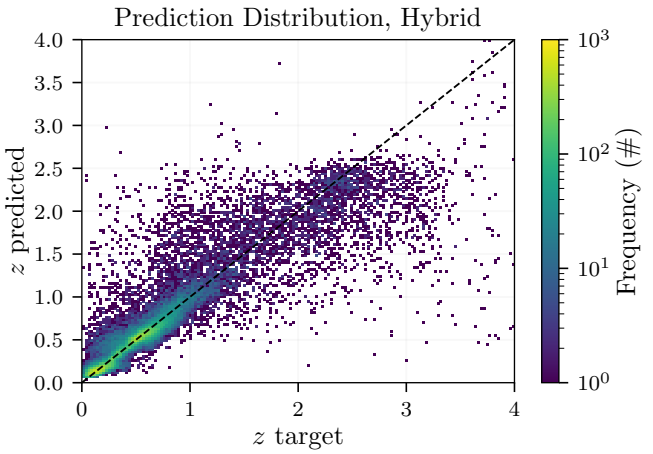

FIG. 11 Shows a 2D histogram of the the predicted redshifts from our tree-based model, the CNN-gradient-boost-hybrid, compared to the real redshifts. The black dotted line represent where the predicted redshift is equal to the real redshift. The color bar represents the frequency of target to predicted redshifts.

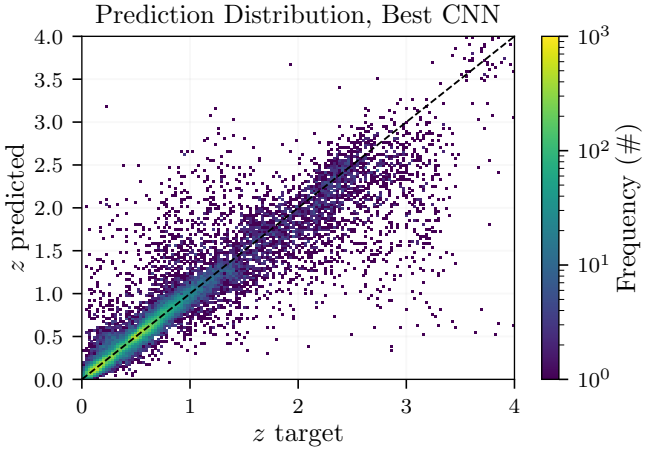

FIG. 12 Shows a 2D histogram of the the predicted redshifts from our best performing CNN ('w6' from Appendix A, which was also the model with the largest number of trainable parameters) compared to the real redshifts. The black dotted line represent where the predicted redshift is equal to the real redshift. The color bar represents the frequency of target to predicted redshifts.

IV. CONCLUSION

To have any hope of resolving the Hubble tension and uncovering the nature of dark energy and matter, the field of cosmology needs considerably more data mapping the recent history of the universe, including the location of the redshift of millions of celestial objects. As an alternative to spectroscopy, we used machine learning methods to estimate redshifts of galaxies from images of different wavelength filter bands from the GalaxiesML dataset.

We specifically applied CNNs of different sizes and architectures to the problem. As an alternative method, we also used regression trees with varying depth and with gradient boosting. We compared methods using R^2 on the test dataset.

Lastly, we also explored a model which is a hybrid of both CNNs and regression tress by first training our model through a CNN and thereafter feeding the prediction of the trained model to a gradient boosted model which resulted in better model performance, as compared

to the other tree-based models. The best traditional tree model achieved a $R^2 = 0.805$ with a computational time of 9.047 minutes. The hybrid CNN-gradient-boost model achieved a $R^2 = 0.814$ with a computational time of 3.098 minutes. Our best CNN achieved $R^2 = 0.876$, but took a much longer 65.24 minutes to train.

The success of our hybrid model is exciting, as it resulted in a non-trivial increase in model performance with a significant reduction in the computational time required. Time is money, especially when using rented computational power, which we have found a method to minimize depending on the priority and use-case of the model.

In their current state, the predictions from our models are inadequate for deriving cosmological results due to their inaccuracy. They do however have enough predictive power to find use in some part of the cosmological data pipeline, perhaps as pre-classification or quick dataset analysis.

A. Future work

As seen in Sec. III, there is much more to explore here. The scope of our results have been bounded by the computational hardware we have access to and the time required to train these models. A simple tree could take anywhere from 10 minutes to a couple hours to train greatly limiting our exploration of the parameter spaces of these models. While we found that a hybrid between a CNN and gradient boosted regression tree trained fast and preformed well, we could not tune this pipeline to preform its best. Similarly the pure CNN models with their astronomical training time also made it hard for us to tune properly. These time constraints have also dissuaded us from substantiating our results by running many times or using techniques like bootstrapping or k-fold cross validation.

We were also limited in the specific CNN architectures we were able to test, and suspect that a better-performing model exists within the same number of trainable parameters as was tested here. Specifically, future work should include the independent effects of pooling type and frequency, padding, kernel size, and number of channels. More sophisticated feature extraction techniques could also be employed.

Our success in this exploration was also limited by the quality of images in our dataset. Aspects such as the clarity of a galaxy or noise from the measurement equipment could of course be improved. Having knowledge of these errors may also allow us to qualitatively discuss how much of the spread is due to measurement errors.

We could also incorporate the experimental uncertainty in the redshift into our calculation of loss and model success.

The dataset we decided to use for the training of our models has many more metrics including galaxy masks and morphology that represent avenues for future projects. Metrics could be predicted, some with our current models, and others with minimal change to its architecture (both regression and classification). There may be factors such as the brightness of the galaxy which if included in the training of out models may improve R^2.

To potentially have a smaller calculated RMSE for our CNN, we could have designed its architecture with an autoencoder. As this may be computationally taxing and given the lack of time in doing this project this was unachievable.

Appendix A: Index of Network Shapes

C_n^k denotes a convolutional layer with an $n \times n$ kernel, k channels, and padding 1 followed by a LeakyRELU activation. P_m indicates max pooling with an $m \times m$ kernel and stride m. An F flattens the output of the previous layer and passes it through a fully connected layer with one output. Data is passed through the network from left to right.

The CNNs used for Figure 8 are:

'Deep' networks:

d1 - $C_3^8 : C_3^{16} : P_4 : F$
d2 - $C_3^8 : C_3^{16} : C_3^{16} : P_4 : F$
d3 - $C_3^{16} : C_3^{16} : C_3^{32} : C_3^{32} : P_4 : F$
d4 - $C_3^{16} : C_3^{16} : C_3^{32} : C_3^{32} : C_3^{32} : P_4 : F$
d5 - $C_3^{16} : C_3^{16} : C_3^{32} : C_3^{32} : C_3^{64} : C_3^{64} : P_4 : F$
d6 - $C_3^{16} : C_3^{16} : C_3^{16} : C_3^{32} : C_3^{32} : C_3^{32} : C_3^{64} : C_3^{64} : C_3^{64} : P_4 : F$

'Wide' networks:

w1 - $C_5^8 : P_4 : F$
w2 - $C_5^16 : P_4 : F$
w3 - $C_5^{16} : C_5^{32} : P_4 : F$
w4 - $C_5^{16} : C_5^{32} : C_5^{64} : P_4 : F$
w5 - $C_7^{16} : C_7^{32} : C_7^{64} : C_7^{128} : P_4 : F$
w6 - $C_7^{16} : C_7^{32} : C_7^{64} : C_7^{128} : C_7^{256} : P_4 : F$

The CNN used for the hybrid model is:

hybridCNN - $C_3^8 : P_2 : C_5^8 : P_2 : C_5^8 : P_3 : F$

Note that hybridCNN does not use padding on its convolutional layers.

REFERENCES

E. D. Valentino, Monthly Notices of the Royal Astronomical Society **502**, 2065 (2021), arXiv:2011.00246 [astro-ph].

D. J. Fixsen, E. S. Cheng, D. A. Cottingham, *et al.*, The Astrophysical Journal **420**, 445 (1994), publisher: IOP ADS Bibcode: 1994ApJ...420..445F.

E. Komatsu, K. M. Smith, J. Dunkley, *et al.*, The Astrophysical Journal Supplement Series **192**, 18 (2011), arXiv:1001.4538 [astro-ph].

Planck Collaboration *et al.*, Astronomy & Astrophysics **641**, A1 (2020).

DESI Collaboration *et al.*, Physical Review D **112**, 083515 (2025), arXiv:2503.14738 [astro-ph].

T. Do, B. Boscoe, E. Jones, Y. Q. Li, and K. Alfaro, (2024), preprint, under review, arXiv:2410.00271 [astro-ph.CO].

T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)* (2009).

K. Graatrud, S. Silverstein, and N. Andrés Tran Rodriguez, Universitetet i Oslo (2025).

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. Del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Nature **585**, 357 (2020).

J. D. Hunter, Computing in Science & Engineering **9**, 90 (2007).

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," (2019), arXiv:1912.01703 [cs.LG].

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).