Application of Machine Learning Methods on Astronomical Databases

Predicting Galaxy Redshifts with Convolutional Neural Networks

Apostolos Kiraleos, AM: 711171037 Supervisor: Nikolaos Vasilas
Co-Supervisor: Emmanuel Bratsolis

Abstract

Galaxy redshift is a crucial parameter in astronomy that provides information on the distance, age, and evolution of galaxies. This dissertation investigates the application of machine learning for predicting galaxy redshifts. It involves the development and training of a neural network to analyze galaxy spectra sourced from the European Space Agency's Gaia mission and showcases the practical implementation of machine learning in astronomy.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Object, Purpose, and Objectives

This dissertation primarily addresses the critical issue of accurate galaxy redshift estimation. Galaxy redshifts are pivotal in modern astronomy, providing crucial insights into cosmic distances, universe evolution, and more. The topic is of significant interest and relevance within the field.

The main goal of this dissertation is to advance galaxy redshift estimation by applying advanced machine learning techniques, specifically Convolutional Neural Networks (CNNs). The objective is evaluating the performance and accuracy of a trained CNN model in predicting galaxy redshifts with the aim to understand the practical applications of this model in astronomical research.

The contribution of this work lies in advancing knowledge within the field of galaxy redshift estimation. By applying state-of-the-art machine learning techniques, we aim to provide an innovative approach to predicting galaxy redshifts, potentially improving accuracy and efficiency in astronomy.

1.2 Methodology

The methodology of this dissertation involves designing, training, and evaluating the performance of a trained Convolutional Neural Network (CNN). The CNN is trained on a dataset of galaxy spectra sourced from the European Space Agency's Gaia mission which came already preprocessed and cleaned, ready for model training. The CNN is then trained on the dataset and evaluated on a test set of galaxy spectra.

For the actual implementation of the CNN, the Python programming language was used, along with the Keras deep learning library.

1.3 Structure

The subsequent chapters of this dissertation are organized as follows:

1. **Introduction** (the current chapter): Provides an overview of the dissertation's focus, objectives, methodology, and innovation.
2. **Background**: Introduce foundational concepts related to the Gaia mission, galaxy redshift and machine learning methods.
3. **Data Preparation**: Details the process of selecting and cleaning galaxy spectra data for model training.
4. **Methodology**: Elaborates on the methodologies used for CNN model training and evaluation.
5. **Results**: Presents the findings of our experiments, assesses model performance, and discusses implications.
6. **Conclusion**: Summarizes key takeaways and outlines potential areas for future research.

2 Background

2.1 The Gaia Mission

The Gaia space mission, conducted by the European Space Agency (ESA), is a scientific endeavor designed to create an accurate three-dimensional map of the Milky Way galaxy and enhance our understanding of our cosmic surroundings. This mission relies on precise instrumentation and astrometry.

2.1.1 Objective and Methodology

Gaia employs an array of instruments onboard, with its primary tool being the Astrometric Instrument. This instrument is equipped with two telescopes and a complex set of detectors. Gaia's main objective is to measure the positions and motions of over a billion stars with unprecedented accuracy. By repeatedly observing these stars over time, Gaia constructs a precise 3D model of the Milky Way galaxy.[1]

2.1.2 Spectroscopy

One of Gaia's notable capabilities is its ability to disperse starlight into spectra. This is achieved through a dedicated Spectroscopic Instrument. This instrument allows Gaia to analyze the spectra of stars, providing valuable insights into their physical properties, such as composition, temperature, and luminosity. Spectroscopy enables the classification of stars into various categories, including main-sequence stars, giants, and white dwarfs.

The capacity to disperse starlight into spectra has implications for the study of galaxy redshift. Gaia's Spectroscopic Instrument can also be utilized to analyze the light emitted by galaxies. By measuring the redshift of galaxy spectra, astronomers can gain insight into the relative motion of galaxies and their distances from us. This redshift data contributes to our understanding of the expanding universe and connects our discussion to the subsequent section, where we delve into the concept of galaxy redshift in greater detail.

2.2 Galaxy redshift

Galaxy redshift is a fundamental astronomical property that describes the relative motion of a galaxy with respect to Earth. The redshift of a galaxy is measured by analyzing the spectrum of light emitted by the galaxy, which appears to be shifted towards longer wavelengths due to the Doppler effect. Redshift is a crucial parameter in astronomy, as it provides information about the distance, velocity, and evolution of galaxies.

The redshift of a galaxy has no units, and is defined as the fractional shift in the wavelength of light emitted by the galaxy. Specifically, the redshift "z" is defined as:

$$z = \frac{\lambda_{obsv} - \lambda_{emit}}{\lambda_{emit}}$$

where λ_{obsv} is the observed wavelength of light from the galaxy, and λ_{emit} is the wavelength of that same light as emitted by the galaxy. A redshift of $z = 0$ corresponds to no shift in the wavelength (i.e., the observed and emitted wavelengths are the same), while a redshift of $z = 1$ corresponds to a shift of 100% in the wavelength (i.e., the observed wavelength is twice as long as the emitted wavelength).

[1] Gaia Overview. ESA. September 26 2023. https://www.esa.int/Science_Exploration/Space_Science/Gaia/Gaia_overview

Accurate and efficient estimation of galaxy redshift is essential for a wide range of astronomical studies, including galaxy formation and evolution, large-scale structure of the universe, and dark matter distribution. However, measuring galaxy redshifts can be a challenging task due to various factors such as observational noise, instrumental effects, and variations in galaxy spectra.[2]

2.3 Artificial Intelligence

Artificial Intelligence, commonly abbreviated as AI, signifies the result of extensive research and development spanning several decades, aimed at equipping machines with intelligence and decision-making abilities resembling those of humans.

One of the defining features of AI is its adaptability—machines equipped with AI algorithms can analyze vast datasets, identify intricate patterns, and make decisions guided by these insights. This adaptability is particularly evident in the field of Machine Learning, a subset of AI that focuses on the development of algorithms capable of learning from data.

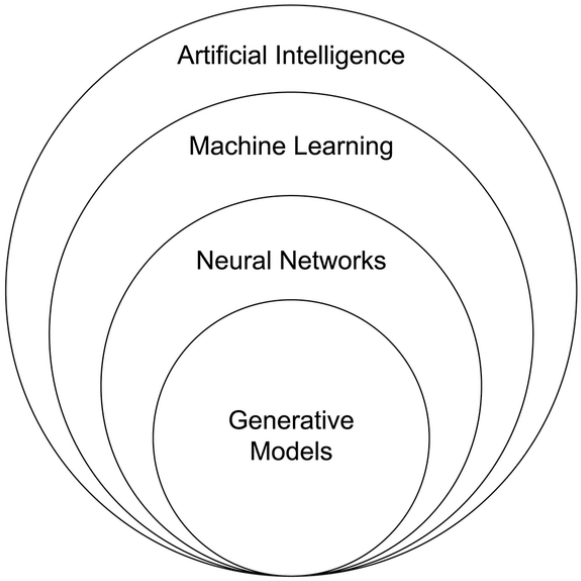

Figure 1: Venn diagram of Artificial Intelligence[3]

2.4 Machine Learning

Machine Learning (ML) is the driving force behind the remarkable progress witnessed in AI. At its core, ML provides the tools and methodologies for machines to learn from experience and iteratively improve their performance on a specific task. Unlike traditional programming, where explicit instructions dictate behavior, ML algorithms discern patterns and relationships within data, allowing them to generalize and make informed decisions when exposed to new information.

ML empowers machines to evolve autonomously, making it indispensable in an era characterized by ever-expanding datasets and complex problems. It encompasses various paradigms, including supervised

[2]What do redshifts tell astronomers. EarthSky. October 4 2023. https://earthsky.org/astronomy-essentials/what-is-a-redshift/

[3]Artificial Intelligence relation to Generative Models subset, Venn diagram. Wikipedia. September 28 2023. https://en.wikipedia.org/wiki/File:Artificial_Intelligence_relation_to_Generative_Models_subset,_Venn_diagram.png

learning, unsupervised learning, and reinforcement learning, each tailored to specific applications and data types.

2.4.1 Artificial Neural Networks

Artificial neural networks (ANNs) are a type of machine learning algorithm that is inspired by biological neural networks in animals. ANNs are composed of individual processing units called neurons, which are organized into layers. Each neuron receives input from other neurons in the previous layer, applies a mathematical operation to that input, and passes the output to the next layer. The output of the final layer of neurons is the predicted output of the network for a given input.

The input layer is the entry point of the neural network, accepting the initial data or features. Each neuron in this layer corresponds to a specific feature, and the values assigned to these neurons represent the input data.

Hidden layers, as the name suggests, are intermediary layers that lie between the input and output layers. These layers contain neurons that process and transform the data as it flows through the network. The presence of multiple hidden layers enables ANNs to capture complex patterns and relationships within the data.

The output layer is responsible for producing the final result, whether it's a classification, prediction, or decision. The number of neurons in this layer corresponds to the desired number of output classes or the nature of the prediction task.

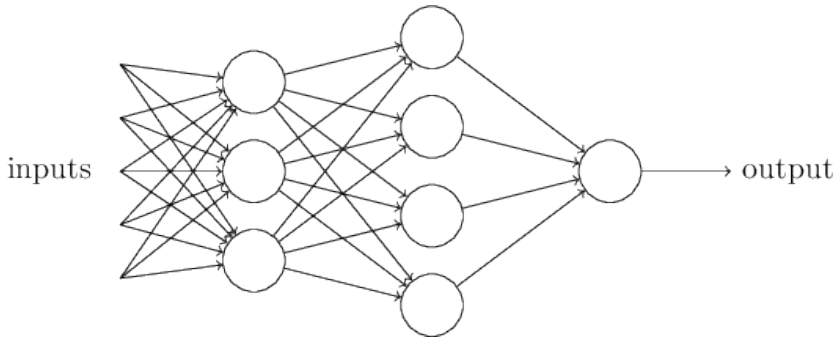

Figure 2: Visualization of an artificial neural network[4]

Inside every connection between neurons are numerical parameters known as weights and biases. These parameters are the essence of a neural network, as they determine the strength and significance of the connections between neurons.

Weights represent the strength of the connections between neurons. Each connection has an associated weight, which multiplies the output of one neuron before it's passed as input to the next neuron. During training, ANNs adjust these weights to minimize the difference between their predictions and the actual outcomes, effectively learning from the data.

Biases are additional parameters that are essential for fine-tuning the behavior of individual neurons. They ensure that neurons can activate even when the weighted sum of their inputs is zero. Biases enable ANNs to model complex functions and make predictions beyond linear relationships in the data.

A single neuron can be described as the sum of its inputs multiplied by their corresponding weights, plus the bias. This sum is then passed through an activation function, which determines the neuron's output. The activation function is a mathematical function that introduces non-linearity into the network,

[4]Michael A. Nielsen. Neural networks and Deep Learning. Determination Press. 2015

allowing it to learn complex patterns and relationships in the data. We will discuss activation functions in more detail later on.

Mathematically, the output of a neuron can be described as:

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

where y is the output of the neuron, f is the activation function, w_i is the weight of the ith input, x_i is the ith input, b is the bias, and n is the total number of inputs.

2.4.2 Mechanism of operation

The operation of an ANN is divided into two fundamental phases: the forward pass and the backward pass. During the forward pass, input data is propagated through the network from the input layer to the output layer. Each neuron processes its inputs, applies the activation function, and passes the result to the next layer.

The backward pass, also known as backpropagation, is where the magic of learning happens. In this phase, the network compares its predictions with the actual outcomes, calculating the discrepancy between the two. It then propagates this error backward through the network, adjusting the weights and biases to minimize a specified loss function. The loss function serves as a measure of the error between the network's predictions and the actual target values. By diminishing this loss, the neural network enhances its capacity to make increasingly accurate predictions.[5]

The backpropagation algorithm process depends on an optimization technique known as gradient descent. Gradient descent determines how the network's weights should be updated to minimize the loss function. To achieve this, it computes the gradient of the loss function with respect to each weight in the network. In other words, it calculates the rate of change of the loss concerning individual weights. This gradient offers crucial guidance, pointing the way toward the most rapid reduction in the loss function.

As the gradient highlights the direction of steepest descent, it becomes the "compass" for the adjustment of weights. Weight updates are made proportionally to the gradient, ensuring that changes are made more significantly in areas where the loss function is decreasing most rapidly. This iterative process of calculating gradients and updating weights continues until the network converges to a state where the loss is minimized to the greatest extent possible.[6]

2.4.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specific type of ANN that has proven to be highly effective for tasks involving image and video analysis, such as object detection, segmentation, and classification. CNNs are inspired by the structure and function of the visual cortex in animals that is tuned to detect specific visual features. In a similar way, CNNs are designed to learn and extract meaningful features from raw data.

The key differentiator of CNNs is their use of convolutional layers, which enable the network to automatically learn and extract local spatial features from raw input data. In a convolutional layer, each neuron is connected only to a small, localized region of the input data, known as the receptive field. By sharing weights across all neurons within a receptive field, the network can efficiently learn to detect local patterns and features, regardless of their location within the input image.

[5]Francois Chollet. Deep Learning with Python. Manning Publications. 2017

[6]What is Gradient Descent?. IBM. September 27 2023. https://www.ibm.com/topics/gradient-descent

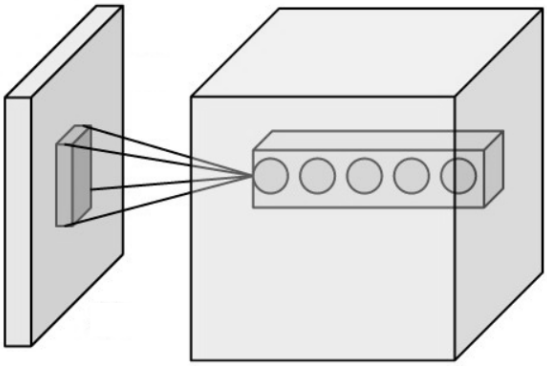

Figure 3: Neurons of a convolutional layer (right) connected to their receptive field (left)[7]

CNNs typically also include pooling layers, which downsample the output of the previous layer by taking the maximum or average value within small local regions. This helps to reduce the dimensionality of the input and extract higher-level features from the local features learned in the previous convolutional layer.

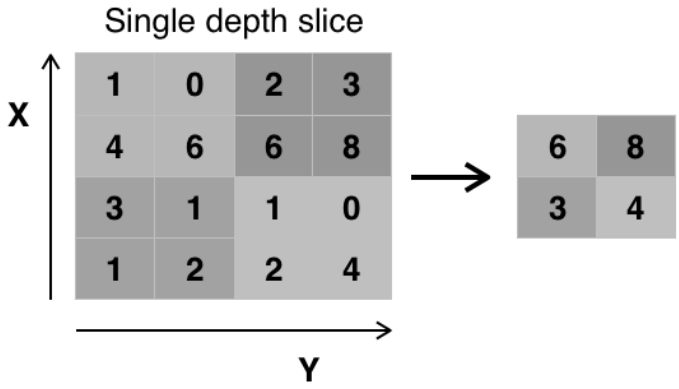

Figure 4: Pooling layer with a 2x2 filter[8]

The final layers of a CNN are fully connected layers, which take the outputs of the previous convolutional and pooling layers and use them to make a prediction. In the case of image classification, for example, the output of the final fully connected layer might be a vector of probabilities indicating the likelihood of each possible class. In our case, instead of class probabilities, the output of the final fully connected layer will yield a single numeric value representing the predicted redshift of the observed galaxy.[9]

[7]Input volume connected to a convolutional layer. Wikipedia Commons. September 28 2023. https://en.wikipedia.org/wiki/File:Conv_layer.png

[8]Pooling layer with a 2x2 filter and stride = 2. Wikipedia Commons. September 28 2023. https://en.wikipedia.org/wiki/File:Max_pooling.png

[9]Francois Chollet. Deep Learning with Python. Manning Publications. 2017

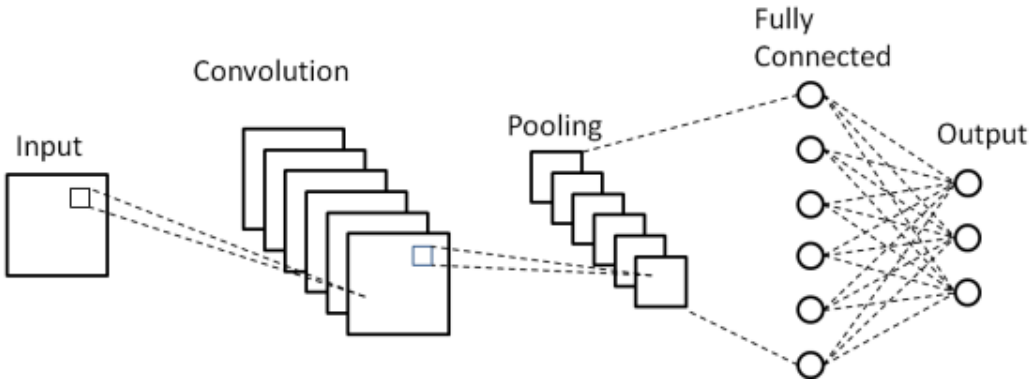

Figure 5: Typical CNN architecture[10]

2.4.4 Activation functions

Activation functions are mathematical functions that are applied to the output of each neuron in a neural network. They are used to introduce non-linearity into the network, which is essential for learning complex patterns and relationships in the data. The most common activation functions used in neural networks are the sigmoid, tanh, and ReLU functions. For *convolutional* neural networks, the ReLU function is typically used for all layers except the output layer, which uses a linear activation function.[11]

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

where x is the input to the function. The sigmoid function is a smooth, S-shaped function that returns a value between 0 and 1. It is commonly used in binary classification problems, where it is used to convert the output of the final layer to a probability between 0 and 1.[12]

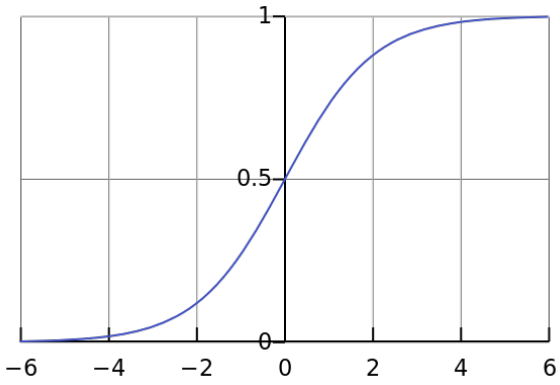

Figure 6: The sigmoid function[13]

[10]Phung, & Rhee,. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. Applied Sciences. 9. 4500. 10.3390/app9214500.

[11]Rectified Linear Units (ReLU) in Deep Learning. Kaggle. September 27 2023. https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning

[12]Activation Functions in Neural Networks. Towards Data Science. September 27 2023. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[13]Sigmoid function. Wikipedia Commons. September 28 2023. https://en.wikipedia.org/wiki/File:Logistic-curve.svg

The tanh function is defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

where x is the input to the function. The tanh function is also a smooth, S-shaped function that returns a value between -1 and 1. It is similar to the sigmoid function, but it is zero-centered.[14]

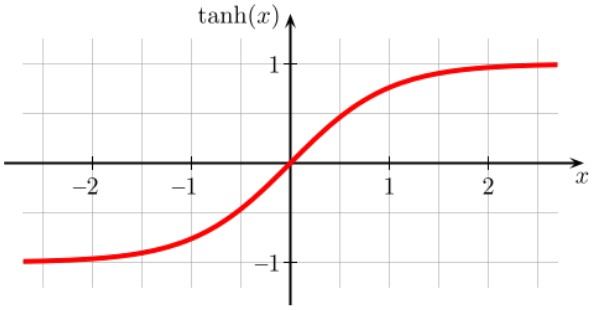

Figure 7: The tanh function[15]

Finally, the ReLU function is defined as:

$$ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

where x is the input to the function. The ReLU function is a simple function that returns 0 if the input is negative, and the input itself if the input is positive.

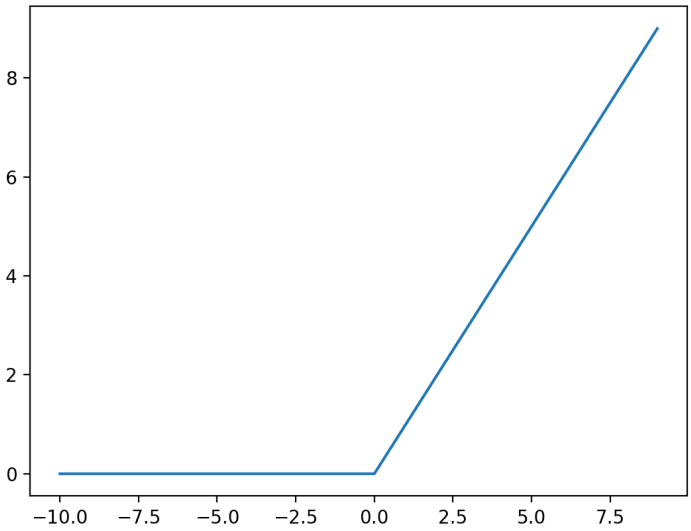

Figure 8: The ReLU function[16]

[14]Activation Functions in Neural Networks. Towards Data Science. September 27 2023. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[15]Tanh function. Wikipedia Commons. September 28 2023. https://en.wikipedia.org/wiki/File:Hyperbolic_Tangent.svg

2.4.5 Loss functions

Loss functions are mathematical functions that are used to measure the difference between the predicted output of a neural network and the actual output. They are used to guide the training process by indicating how well the network is performing. The most common loss functions used in neural networks are the mean squared error (MSE) and mean absolute error (MAE) functions.

The MSE function is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The MAE function is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i is the actual output and $\hat{y}_i$ is the predicted output for the ith sample, and n is the total number of samples.

Another loss function that is commonly used in neural networks is the Huber loss function. It is less sensitive to outliers in data than the squared error loss. It's defined as:

$$L_\delta = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{otherwise}\end{cases}$$

where δ is a small constant.

2.4.6 Optimizers

Optimizers serve as pivotal components in the training of neural networks, enabling the iterative adjustment of network weights to minimize the loss function and enhance overall performance. They play a crucial role in guiding the network's convergence toward optimal solutions. In the realm of CNNs, several optimizers are frequently employed to fine-tune model parameters. Notable among them are the stochastic gradient descent (SGD), Adam, and Adamax optimizers.

SGD is a foundational optimizer that forms the basis for many modern variants. It operates by updating weights in the direction that reduces the loss function. Although simple, SGD can be effective in optimizing neural networks, especially when coupled with proper learning rate schedules.[17]

The Adam optimizer, which stands for Adaptive Moment Estimation, is a powerful and widely adopted optimization algorithm. It combines the benefits of both momentum-based updates and adaptive learning rates. Adam maintains two moving averages for each weight, resulting in efficient and adaptive weight updates. This optimizer excels in handling non-stationary or noisy objective functions, making it a popular choice for training CNNs.[18]

Adamax is an extension of the Adam optimizer that offers certain advantages in terms of computational efficiency.

[16]Jason Brownlee. A Gentle Introduction to the Rectified Linear Unit (ReLU). Machine Learning Mastery. September 28 2023. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks

[17]Stochastic Gradient Descent (SGD). GeeksForGeeks. September 27 2023. https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

[18]Adam: A Method for Stochastic Optimization. arXiv. September 27 2023. https://arxiv.org/abs/1412.6980

2.4.7 Batch size

The batch size is a parameter in the training phase of a neural network. It signifies the number of data samples that are processed in a single forward and backward pass during each training iteration. The choice of batch size carries significant implications for the network's training dynamics.

Utilizing a larger batch size can expedite the training process, as more samples are processed in parallel. This can lead to faster convergence, especially on hardware optimized for parallel computations. However, there is a trade-off, as larger batch sizes can increase the risk of overfitting. The network might memorize the training data rather than learning to generalize from it.

Conversely, a smaller batch size entails processing fewer samples at once. This can result in slower training progress, particularly on hardware with limited parallelism. However, smaller batch sizes often lead to better generalization, as the network receives a more diverse set of samples during training. It is less likely to memorize the training data and is more likely to extract meaningful patterns.[19]

2.4.8 Epochs

Epochs refer to the number of times the entire training dataset is processed by the neural network. Each epoch represents a complete cycle through the dataset, during which the network updates its weights based on the observed errors. The choice of the number of epochs is another vital training hyperparameter.

Training for too few epochs may result in an underfit model, and, conversely, training for an excessive number of epochs can lead to overfitting, more about these two concepts in the next section.

Determining the ideal number of epochs involves a balance between achieving convergence and avoiding overfitting. Typically, researchers employ techniques like early stopping, which monitors validation performance and halts training when it starts to degrade, to guide epoch selection.

2.4.9 Common training roadblocks

Overfitting and underfitting where mentioned before as two common problems that can occur during the training of a neural network. There are also other common problems that can occur during training, such as vanishing and exploding gradients.

2.4.9.1 Overfitting & Underfitting Overfitting occurs when a model learns to fit the training data too closely. In other words, it captures not just the underlying patterns but also the noise in the data. As a result, the model performs exceptionally well on the training data but poorly on unseen data. Overfitting is a sign that the model has become too complex, often due to hyperparameters like a high degree of polynomial features or a large number of hidden layers in a neural network.

Underfitting on the other hand, occurs when a model is too simple to capture the underlying patterns in the data. It fails to learn from the training data effectively and, as a consequence, performs poorly both on the training set and unseen data. Underfitting can be a result of hyperparameters that restrict the model's capacity, such as a shallow architecture or a low learning rate.

To mitigate overfitting, regularization techniques such as dropout, L1/L2 regularization, and early stopping are commonly employed. These methods introduce constraints on the model's parameters, discouraging it from fitting noise in the data. In the case of underfitting, model architecture adjustments, including increasing the depth and complexity of neural networks, may be necessary. A careful balance

[19] How to Control the Stability of Training Neural Networks With the Batch Size. Machine Learning Mastery. September 27 2023. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

between model complexity and the size of the training dataset is crucial to combat both overfitting and underfitting effectively.[20]

2.4.9.2 Vanishing & Exploding Gradients Vanishing gradients are another issue in neural networks, particularly in networks with many layers. When backpropagating errors through deep architectures, gradients can become infinitesimally small. This phenomenon impedes the effective update of weights in earlier layers, causing slow convergence or stagnation in learning. Vanishing gradients restrict the network's capacity to capture long-range dependencies in sequential data or hierarchies of features in deep convolutional networks.

Conversely, exploding gradients occur when gradients become exceedingly large. This can lead to unstable training dynamics, as the weights are updated too drastically. Exploding gradients are often a result of poor weight initialization or a high learning rate.

Several strategies have been proposed to address vanishing and exploding gradients. Weight initialization techniques, such as Xavier (Glorot) initialization, are designed to ensure proper scaling of weights, helping to alleviate the vanishing gradient problem. Gradient clipping, which involves bounding gradients during training, prevents them from reaching extremely high values, mitigating the issue of exploding gradients. Additionally, the use of activation functions with derivatives that do not approach zero or infinity, such as the Rectified Linear Unit (ReLU), has become prevalent in deep neural networks, offering some resilience against vanishing and exploding gradients.[21]

[20]Overfitting and Underfitting With Machine Learning Algorithms. Machine Learning Mastery. October 6 2023. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[21]The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks. Analytics Vidhya. October 6 2023. https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/

3 Data Preparation

3.1 Source & Composition

*N.B: The data collection and cleaning was already done by Ioannis Bellas-Velidis and Despina Hatzidimitriou who worked on Gaia's Unresolved Galaxy Classifier (UGC) and who generously provided us with the dataset. What follows is **their** process of obtaining it.*[22]

The instances of the data set are selected galaxies with known redshifts. The target value is the redshift of the source galaxy or a specific value derived from it. The input data are the flux values of the sampled BP/RP (blue/red photometers, the instruments on board Gaia) spectrum of the galaxy[23]. The edges of the BP spectrum are truncated by removing the first 34 and the last 6 samples, to avoid low signal-to-noise data. Similarly, the first 4 and the last 10 samples are removed from the RP spectrum. The "truncated" spectra are then concatenated to form the vector of 186 (80 BP + 106 RP) fluxes in the 366nm to 996nm wavelength range.

The galaxies used for the dataset were selected from the Sloan Digital Sky Survey Data Release 16 (SDSS DR16) archive. Galaxies with bad or missing photometry, size, or redshift were rejected. The SDSS galaxies were cross-matched with the observed Gaia galaxies. The result was a dataset of SDSS galaxies that were also observed by Gaia. Due mainly to the photometric limit of the Gaia observations, most of the high-redshift galaxies ($z > 1.0$) are absent. The high redshift regime is very sparsely populated and would lead to a very unbalanced training set. Thus, an upper limit of $z = 0.6$ was imposed to the SDSS redshifts, rendering a total of 520.393 sources with $0 \leq z \leq 0.6$ forming the final dataset.

3.2 Analysis

The following section provides an overview of the dataset used for training the CNN model. It includes a sample of the data and a brief data analysis of the data's distribution and characteristics.

Table 1: Three randomly selected galaxies from the dataset.

z	$flux_0$	$flux_1$	$flux_2$	...	$flux_{183}$	$flux_{184}$	$flux_{185}$
0.1735581	0.539	0.514	0.439	...	2.909	2.705	2.386
0.2647779	-0.213	-0.14	0.052	...	13.063	12.639	12.429
0.1288678	0.394	0.329	0.287	...	5.484	5.281	5.123

This table shows a sample of the dataset which includes 3 randomly selected galaxies. The first column is the redshift z of the galaxy, and the remaining columns are its first and last 3 flux values.

[22] Bellas-Velidis & Hatzidimitriou. Unresolved Galaxy Classifier (UGC). September 30 2023. https://gea.esac.esa.int/archive/documentation/GDR3/Data_analysis/chap_cu8par/sec_cu8par_apsis/ssec_cu8par_apsis_ugc.html

[23] René Andrae. Sampled Mean Spectrum generator (SMSgen). Gaia Archive. September 30 2023. https://gea.esac.esa.int/archive/documentation/GDR3/Data_analysis/chap_cu8par/sec_cu8par_apsis/ssec_cu8par_apsis_smsgen.html

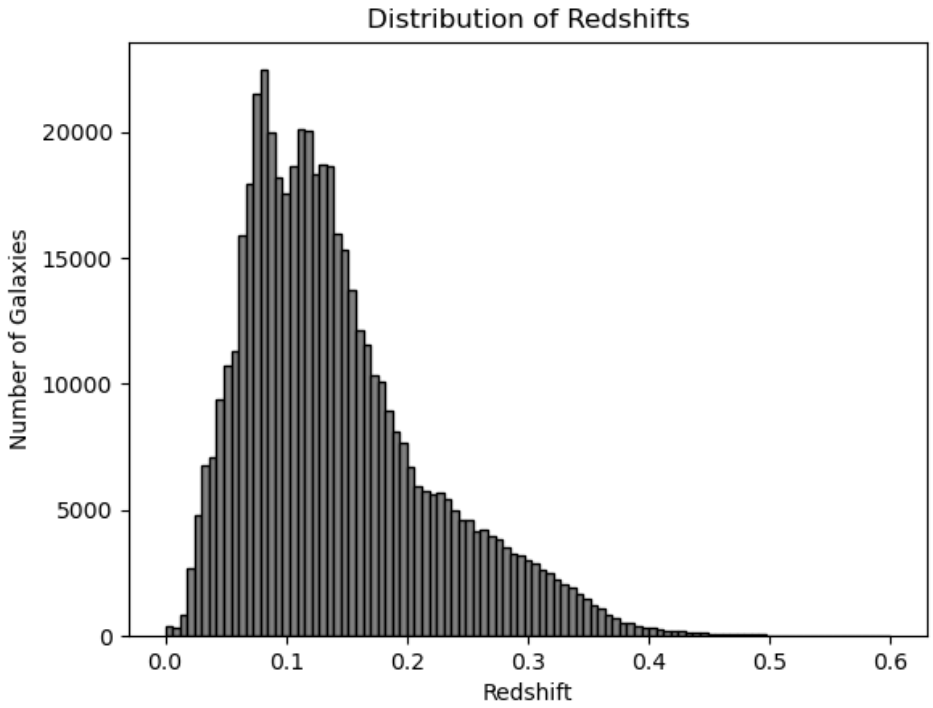

Figure 9: Redshift distribution per bin of 0.01

The figure above shows the distribution of the redshifts in the dataset. The x-axis represents the redshift, and the y-axis represents the number of galaxies in the dataset with that redshift. The redshifts are split into bins of 0.01, and the number of galaxies in each bin is plotted. The figure shows that the redshifts are not uniformly distributed, but instead, they follow a highly skewed (skewness of 1.094) normal distribution. The mean of the redshifts is 0.142, the median 0.126 and the standard deviation is 0.079.

The 99th percentile of the redshifts is approximately 0.37, which means that *99%* of the galaxies have a redshift of $z \leq 0.37$. As we will see later on, this will have an impact on the performance of the model in the redshifts of that range.

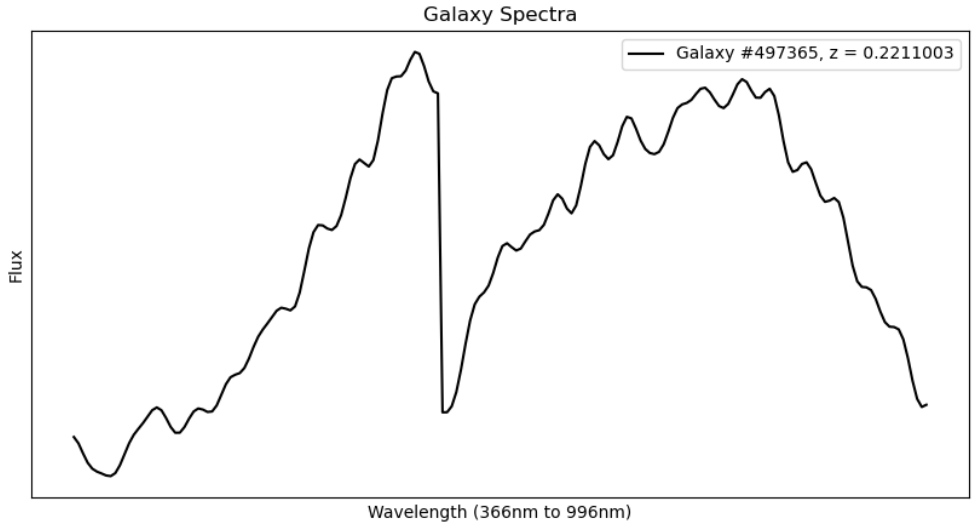

Figure 10: A randomly selected galaxy spectra

This figure shows a randomly selected galaxy spectra. The x-axis represents the wavelength in nanometers, and the y-axis represents the flux. The dip in the middle of the spectrum is the point where the BP and RP spectra are concatenated.

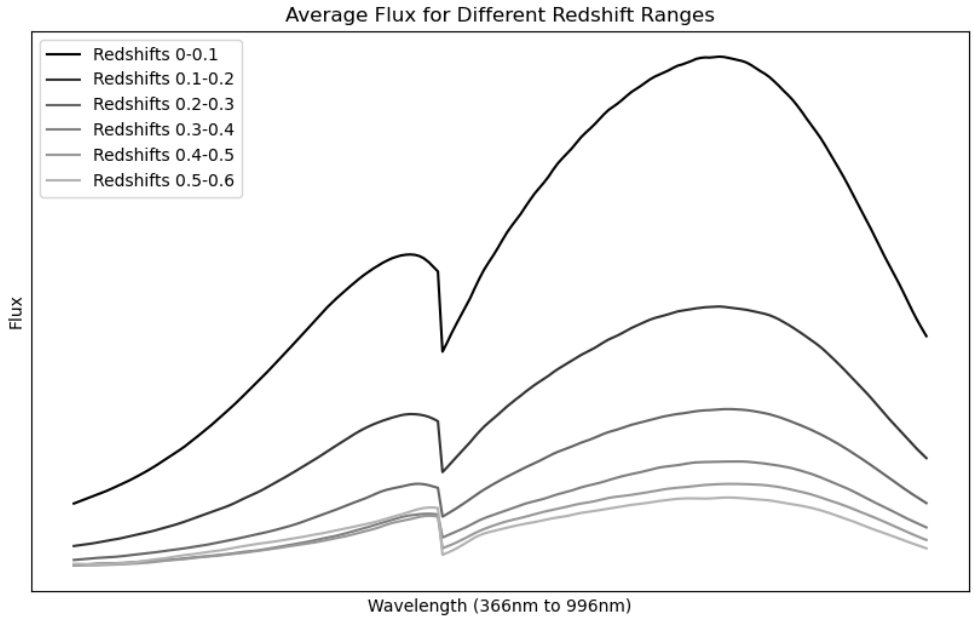

Figure 11: Average flux values for galaxy spectra within various redshift ranges

This figure shows the average flux values for galaxy spectra within various redshift ranges.

We split the redshifts into 6 ranges: $[0, 0.1)$, $[0.1, 0.2)$, $[0.2, 0.3)$, $[0.3, 0.4)$, $[0.4, 0.5)$, and $[0.5, 0.6]$. For each range, we calculated the average flux values of 1000 randomly chosen galaxies (or all available, if fewer than 1000 were in a range).

The figure shows that the average flux values decrease as the redshift increases. This might also explain why our model performs worse on higher redshifts, as the flux values are "flatter", and their features and characteristics less pronounced.

3.3 Preprocessing

Data preprocessing is a crucial step in machine learning, as it can significantly impact the performance of a model. It involves transforming raw data into a format that is more suitable for machine learning algorithms. This process can include various steps, such as data cleaning, feature scaling and normalization, and data splitting.[24]

Feature scaling is a preprocessing step to standardize the range of independent variables or features in the dataset. It ensures that no single feature disproportionately influences the learning process. Common scaling techniques include min-max scaling, Z-score normalization, and robust scaling. Properly scaled features promote faster convergence and more stable training.

On our dataset, since the data was already cleaned, the only preprocessing step we had to apply was min-max scaling to the flux values, which rescales them to the range $[0, 1]$. Other scaling techniques were also tested, but min-max scaling yielded the best results.

It is mathematically defined as:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x is the original data and x_{min} and x_{max} are the minimum and maximum values of x, respectively.

3.4 Splitting

Data is typically split into three sets: a training set, a validation set, and a test set. The training set is used to train the neural network, the validation set is used to fine-tune hyperparameters and monitor model performance during training, and the test set evaluates the model's performance on unseen data. This separation ensures that the model's performance metrics are reliable indicators of its generalization ability.

Our dataset was first split into a training and a test set. The training set contained 90% of the data, while the test set contained the remaining 10%. Then of the training set, 30% was used as a validation set. This resulted in a training set of 63% of the data, a validation set of 27% of the data, and a test set of 10% of the data.

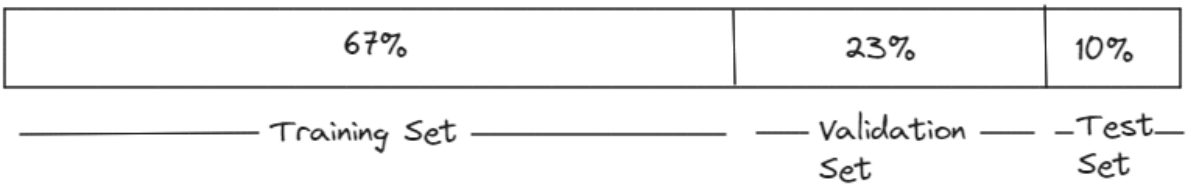

Figure 12: Data split

[24]Why Data should be Normalized before Training a Neural Network. Towards Data Science. September 26 2023. https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d

4 Methodology

4.1 Why convolutional neural networks?

The basis of the decision to use a convolutional neural network is within the nature of the input data. Galaxy spectra, represented as a one dimensional array of flux values have similar characteristics to those of images. It is essentially a one dimensional image, where each "pixel", or, flux at a particular wavelength is correlated with its neighbours. One could thing of this problem as trying to predict the frequency (or wavelength) of a sine wave.

Convolutional neural networks excel at capturing local patterns within data. Unlike traditional neural networks that treat each data point independently, CNNs use convolutional layers to slide over the input, extracting relevant features through a receptive field.

Another significant advantage of CNNs is their ability to perform dimensionality reduction effectively. By applying convolutional and pooling layers, these networks reduce the length of the input while retaining essential features. This is particularly advantageous when working with one dimensional arrays, as it helps in condensing the spectral information while simultaneously preserving the most critical spectral features.

4.2 Hyperparameter Optimization

Hyperparameters are the knobs and levers of a machine learning model. While regular parameters are learned from the training data (e.g., the weights in a neural network), hyperparameters are predefined settings that dictate how a model learns. These settings influence various aspects of the learning process, including the learning rate, the number of hidden layers and their units, the batch size, etc.

4.2.1 Methodologies of Optimization

Hyperparameter optimization involves exploring various combinations of hyperparameters to find the optimal configuration. Several methods exist for this, including grid search, random search, and Bayesian optimization.

The traditional way of performing hyperparameter optimization has been grid search, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a model.

Secondly, random search simply selects hyperparameters randomly. While it might not explore every combination, it often reaches near-optimal configurations faster than grid search, especially when only a small number of hyperparameters affects the final performance of the model.[25]

Finally, Bayesian optimization employs probabilistic models to guide the search efficiently. It smartly explores the space of potential choices of hyperparameters by deciding which combination to explore next based on previous observations. This makes it more efficient than random *and* grid search, as it can reach near-optimal configurations faster.[26] For this reason, and since Keras provides a simple API for it, Bayesian optimization was chosen for hyperparameter optimization.

The search space of hyperparameters was chosen as: the number of convolutional layers, the number of filters in each convolutional layer, the kernel size of each convolutional layer, the number of dense

[25] Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research. https://www.cs.ubc.ca/labs/algorithms/Projects/SMAC/papers/11-LION5-SMAC.pdf

[26] Sequential model-based optimization for general algorithm configuration. Learning and Intelligent Optimization. Lecture Notes in Computer Science. https://www.cs.ubc.ca/labs/algorithms/Projects/SMAC/papers/11-LION5-SMAC.pdf

layers, the number of units in each dense layer, their activation functions, the loss function, and the optimizer.

4.2.2 Results

The final architecture of the model (after Bayesian hyperparameter optimization) is as follows:

Table 2: Final architecture of the model.

Layer type	Filters / Units	Kernel Size	# of Params
Convolutional	256	5	1536
Max Pooling	-	-	0
Convolutional	256	5	327936
Max Pooling	-	-	0
Convolutional	128	3	98432
Max Pooling	-	-	0
Convolutional	64	2	16448
Max Pooling	-	-	0
Flatten	-	-	0
Dense	256	-	147712
Dense	256	-	65792
Dense	128	-	32896
Dense	64	-	8256
Dense	1	-	65
Total			699.073

The optimizer and loss function used (after hyperparameter optimization) was the Adamax optimizer with the default starting learning rate of 0.001 and the huber loss function with the default parameters, respectively.

The batch size and number of epochs were not included in the search space of hyperparameters, but instead were chosen manually after experimentation using the early stopping technique. The batch size was chosen as 16 and the number of epochs as 20.

4.3 Training & Validation

The model was trained on a single AMD RX 6600 GPU with 8GB of VRAM. The training process took approximately 1 hour and 15 minutes. The training and validation loss and mean absolute error (MAE) are shown in the figures below.

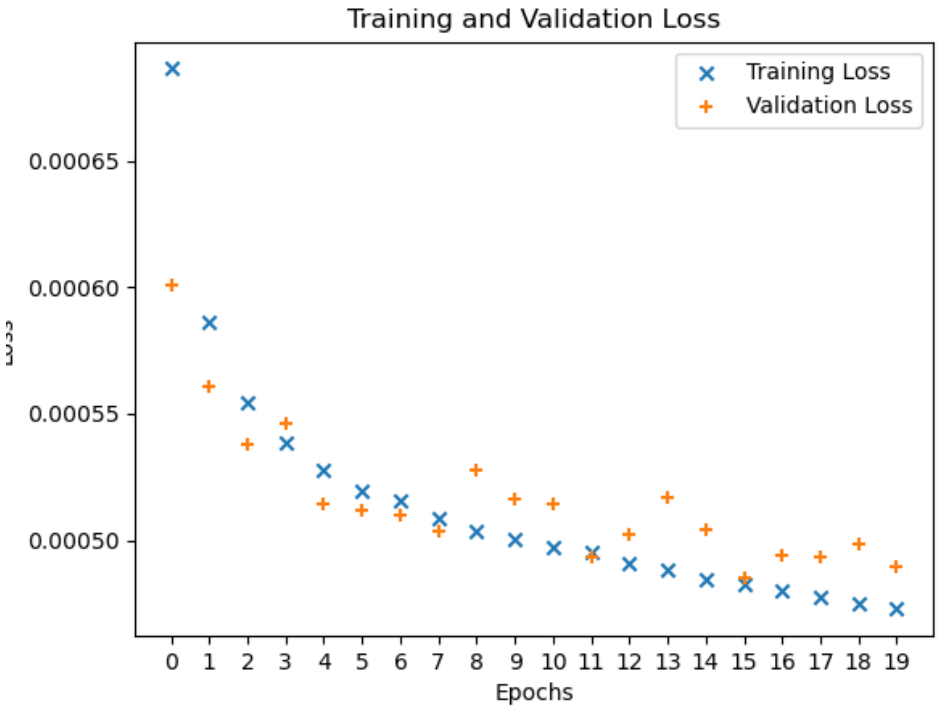

Figure 13: Training and validation loss after 20 epochs

We can see from the training and validation loss figure that the model doesn't overfit, as the training loss decreases monotonically the validation loss closely follows it. The training loss is slightly lower than the validation loss, which is expected, as the validation set is data that the model hasn't seen before.

It might seem from this figure that if we were to train the model for more epochs, the training and validation losses would continue to decrease. However, this is not the case, as the model has already converged, and training it for more epochs would only lead to overfitting, as shown by the next figure.

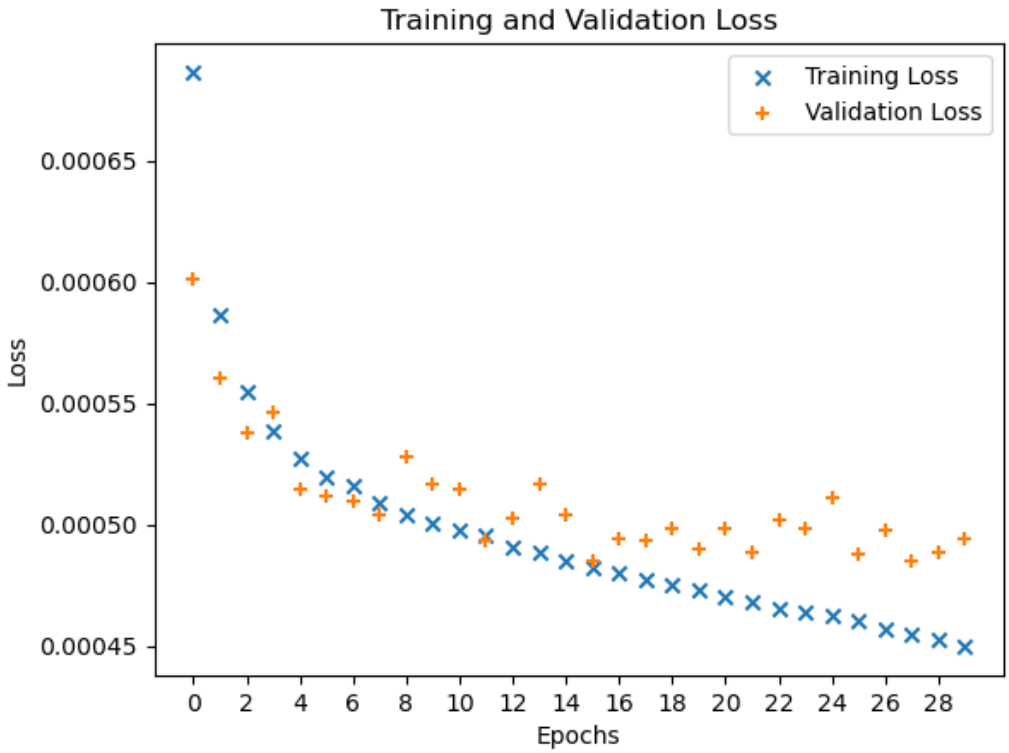

Figure 14: Training and validation loss after 30 epochs

From this figure we can see that the training loss continues to decrease after 20 epochs, but the validation loss keeps hovering around the same value (0.00050) as with 20 epochs. This is a clear sign that the model has converged and is now overfitting.

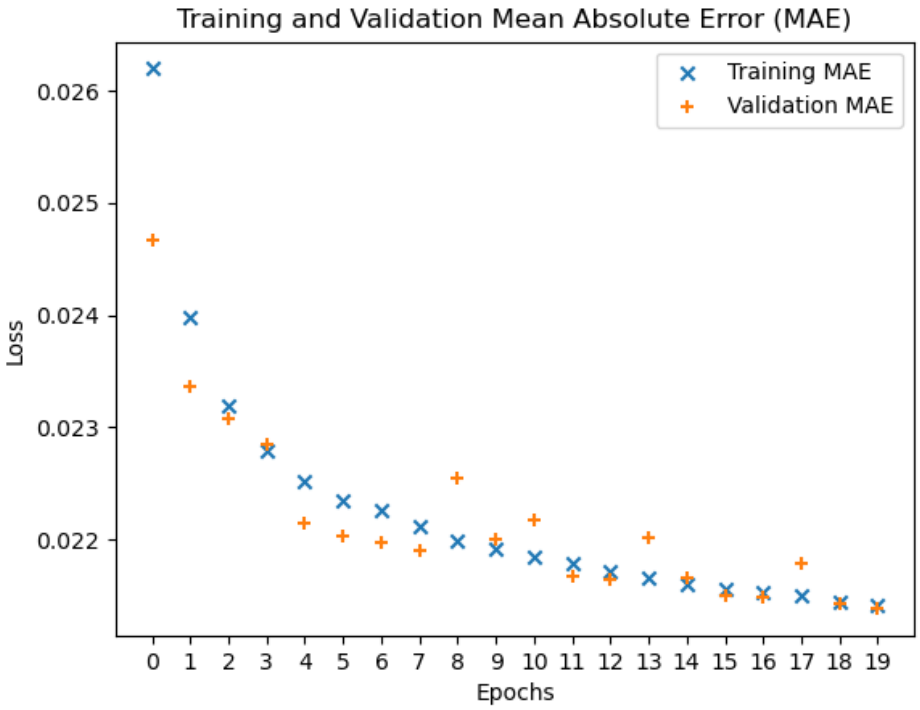

Figure 15: Training and validation Mean Absolute Error

This figure shows the Mean Absolute Error (MAE) of the training and validation sets. The MAE is a metric that measures the average difference between the model's predictions and the actual values. It's a common metric used to evaluate the accuracy of regression models.

The training loss seems to converge to a value of approximately 0.021 after 20 epochs. The validation loss is also exactly the same, which means that the model performs equally well on the training and validation sets. This is a good sign, as it means that the model generalizes well to unseen data.

5 Results & Discussion

Next we will discuss the results of the model and its performance on the test set by analyzing its performance on the different redshift ranges through looking at different figures and metrics.

5.1 Error & Standard Deviation

The model was evaluated on the test set, which contained 10% of the data (around 52.000 samples). The test set was not used during training, so it represents unseen data.

It achieved a mean absolute error (MAE) of 0.021 on the test set which, practically, means that the model's predictions are on average 0.021 away from the actual values. For example, if the actual redshift of a galaxy is 0.142 (the dataset's mean redshift), the model's prediction will be either 0.163, or 0.121, on average. Now, let's look at the model's performance on more specific different redshift ranges.

Table 3: Mean error, standard deviation and percentage of data per redshift bin of 0.05

redshift bin	mean error	std (σ)	% of data
0.00 - 0.05	0.0216	0.0236	6.62
0.05 - 0.10	0.0117	0.0208	27.77
0.10 - 0.15	0.0013	0.0243	28.99
0.15 - 0.20	-0.0081	0.0296	16.61
0.20 - 0.25	-0.0110	0.0346	8.80
0.25 - 0.30	-0.0122	0.0333	6.00
0.30 - 0.35	-0.0248	0.0351	3.55
0.35 - 0.40	-0.0458	0.0416	1.15
0.40 - 0.45	-0.0793	0.0593	0.30
0.45 - 0.50	-0.1327	0.0806	0.11
0.50 - 0.55	-0.1804	0.1128	0.06
0.55 - 0.60	-0.2100	0.0978	0.04

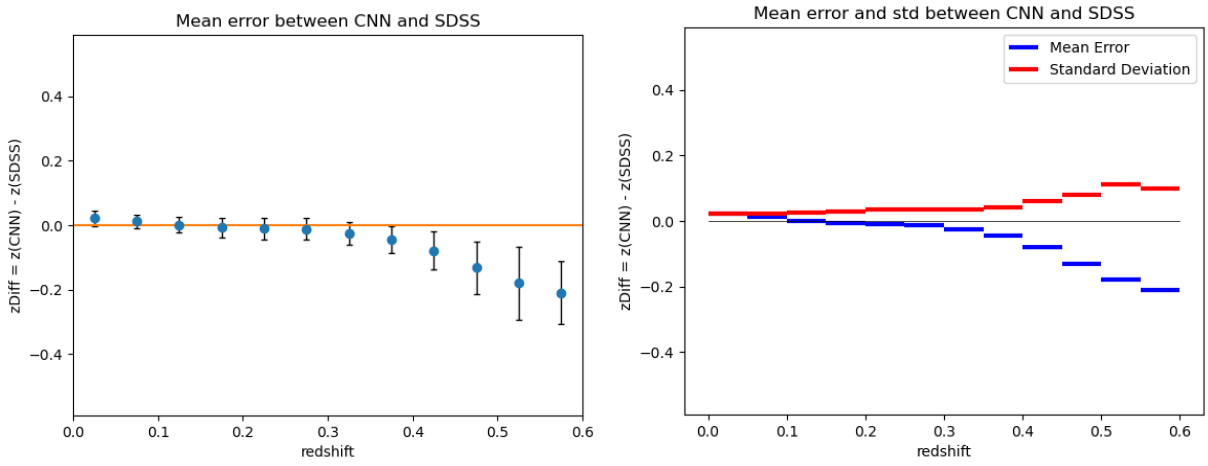

Figure 16: Mean error and standard deviation per redshift bin

The absolute mean error is $\leq |0.21|$ for the whole dataset, and the standard deviation is ≤ 0.1. As the redshift increases, the absolute mean error also increases, and the standard deviation increases as well. This is expected, as the model was trained on a dataset that contained mostly low redshift galaxies.

Since, though, 99% of the galaxies have a redshift of $z \leq 0.37$, we can calculate the mean absolute error for the first 7 bins (0 to 0.35) to determine the model's performance on the overwhelming majority of the data. The mean absolute error for these bins is 0.013 and the mean standard deviation is 0.032. This means that the model's predictions are on average 0.013 away from the actual values, which is a very good result.

We can also pick out the best performing bins which are the 2nd, 3rd, 4th, 5th and 6th redshift bins, and which constitute 88.17% of the data. The mean error in these bins is 0.00886.

The best performing bin is the 3rd bin (0.10 - 0.15) with 29% of the data and a mean error of 0.0013 and a standard deviation of 0.0243. This means that the model's predictions are on average 0.0013 away from the actual values, which is an excellent result.

The worst performing bins are the 9th, 10th, 11th and 12th redshift bins, which constitute 0.51% of the data. The mean absolute error in these bins is 0.15.

The first bin (0.00 - 0.05) is also a relatively badly performing bin, at least compared to the next 6, with a mean error of 0.0216. This is not entirely expected, since galaxies with such a low redshift, and therefore close to Earth, should have low signal-to-noise spectra and thus have cleaner data for a model to learn from.

A final note is that the model's bias is negative, which means that it tends to predict lower redshifts than the actual values.

5.2 Visual Analysis

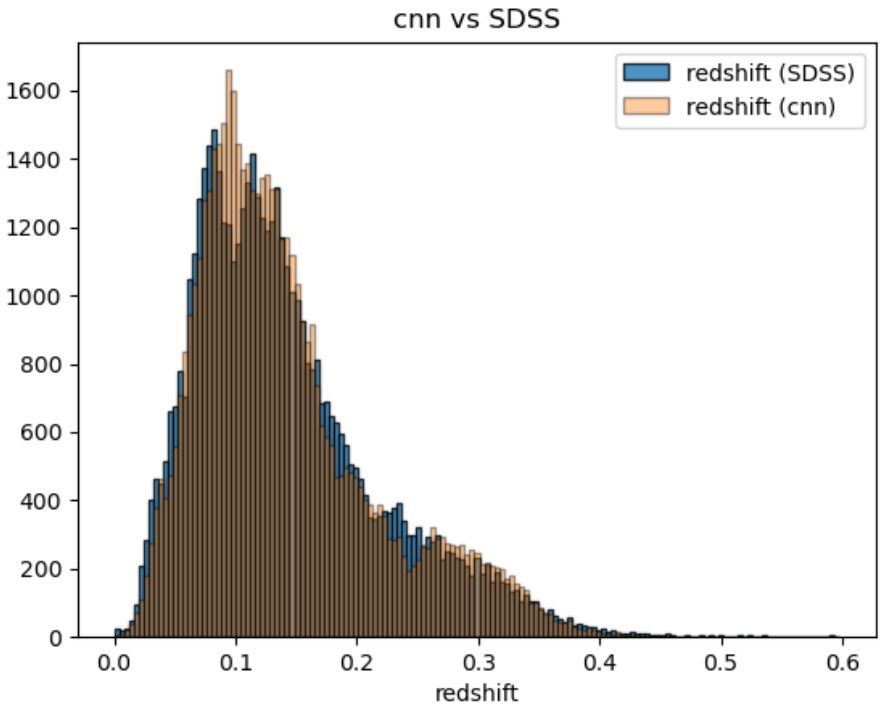

Figure 17: Histogram of the predicted and true redshifts

This figure shows a histogram of the predicted and true redshifts. The x-axis represents the redshift, and the y-axis represents the number of galaxies with that redshift. The blue bars represent the true redshifts, and the beige bars represent the predicted redshifts.

If the model could predict the redshifts perfectly, the beige bars would be exactly on top of the blue bars. However, we can see that the beige bars are not exactly on top of the blue bars. The model misses the double peaks around the 0.1 redshift mark and instead has a single higher peak, and there are some ranges where the model predicts lower redshifts than the true values for example around the 0.2 redshift mark.

Overall, though, as we also saw in the previous section, the model's predictions are very close to the true values, and the histogram shows that the model performs very well.

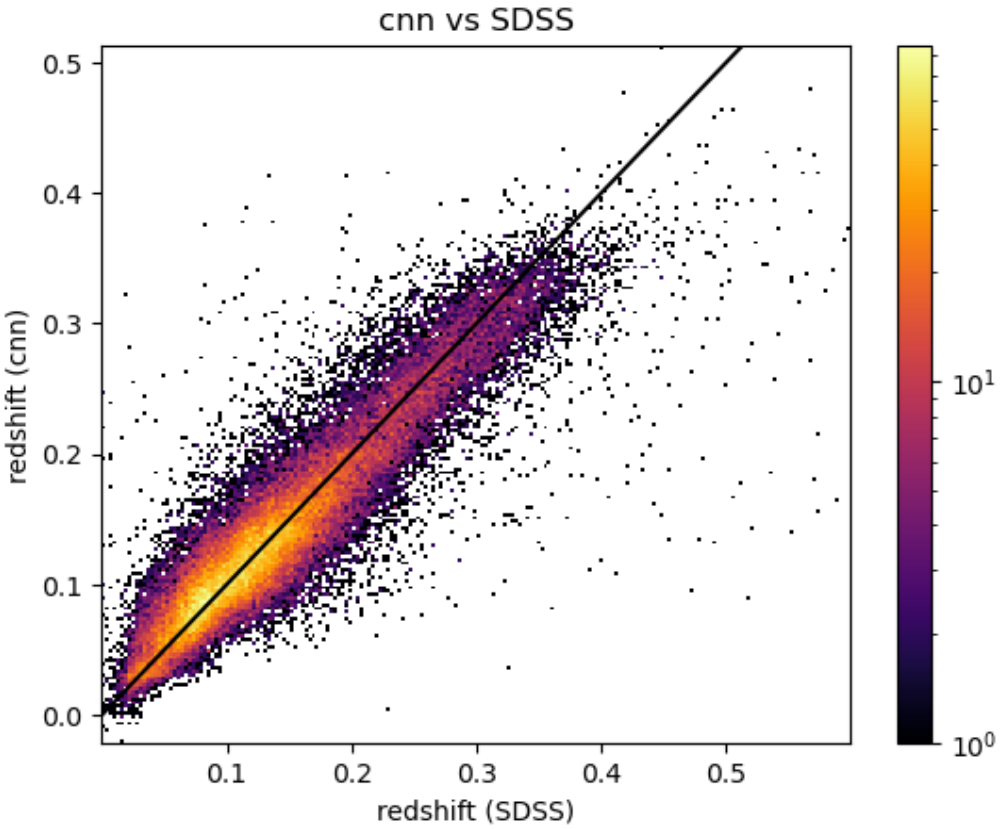

Figure 18: Two dimensional histogram of the predicted and true redshifts

This figure shows a two dimensional histogram of the predicted and true redshifts. The x-axis represents the true redshift, and the y-axis represents the predicted redshift. The color of each point represents the number of galaxies with that true and predicted redshift. Worth noting is that the color scale is logarithmic, so the redder the color, the (exponentially) more galaxies there are with that true and predicted redshift.

For a perfect fit, all of the points would be on the black diagonal line. Obviously, we can see that the points are not exactly on it but are very close to it.

This type of plot is also useful for seeing the outliers of the model. The outliers are the points that are far away from the diagonal line. We can see that there are some outliers, but they are very few (in the order of 100s) compared to the total number of galaxies (in the order of 10.000s).

We can also see the model's negative bias in the upper redshift ranges as the points are, on average,

slightly below the diagonal line the more you go up in redshift. We can also note that most of the outliers are also below the diagonal line, so they could be a contributing factor to the model's negative bias.

This concludes the results and discussion chapter. Next we will look at the conclusions of this dissertation.

6 Conclusions

In this dissertation, we trained and evaluated a Convolutional Neural Network that predicts galaxy redshifts. This work leveraged the dataset generously provided by Ioannis Bellas-Velidis and Despina Hatzidimitriou who worked on Gaia's Unresolved Galaxy Classifier (UGC).

The findings reveal that the CNN exhibits a commendable performance in predicting galaxy redshifts. The model achieved a mean absolute error (MAE) of 0.021 on the test set, signifying a high level of accuracy. This result showcases the model's capability to make predictions with an average deviation of only 0.021 from actual values, on average.

Furthermore, a detailed examination of the model's performance across different redshift ranges demonstrated its robustness. Notably, for redshift ranges within the 0.00 - 0.35 interval, encompassing the overwheling majority of the data, the model displayed an even more impressive MAE of 0.013.

While the model's predictions exhibited a slight negative bias, especially in higher redshift ranges, this bias remains a potential area for future refinement. The two-dimensional histogram analysis revealed that outliers were relatively scarce, affirming the model's consistency in predicting redshifts.

This research contributes to the field of astronomy and machine learning by showcasing the effectiveness of CNNs in redshift prediction tasks, particularly for galaxies with low to moderate redshifts. These results have significant implications for astronomical studies, facilitating more precise estimations of galaxy redshifts and consequently enhancing our understanding of the cosmos.

In closing, this work serves as a foundation for further exploration into machine learning applications in astrophysics. Future endeavors could explore methods for mitigating the model's bias and bad performance in higher redshift ranges and investigate the integration of additional data sources to increase predictive accuracy.

Appendix

Sources

Books

1. Francois Chollet. Deep Learning with Python. Manning Publications. 2017
2. Michael A. Nielsen. Neural networks and Deep Learning. Determination Press. 2015

Papers

1. Adam: A Method for Stochastic Optimization. arXiv. September 27 2023. https://arxiv.org/abs/1412.6980
2. Phung, & Rhee,. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. Applied Sciences. 9. 4500. 10.3390/app9214500.
3. Bergstra, James & Bengio, Y.. (2012). Random Search for Hyper-Parameter Optimization. The Journal of Machine Learning Research. 13. 281-305.
4. Sequential model-based optimization for general algorithm configuration. Learning and Intelligent Optimization. Lecture Notes in Computer Science. https://www.cs.ubc.ca/labs/algorithms/Projects/SMAC/papers/11-LION5-SMAC.pdf

Website sources

1. Gaia Overview. ESA. September 26 2023. https://www.esa.int/Science_Exploration/Space_Science/Gaia/Gaia_overview
2. What do redshifts tell astronomers. EarthSky. October 4 2023. https://earthsky.org/astronomy-essentials/what-is-a-redshift/
3. What is Gradient Descent?. IBM. September 27 2023. https://www.ibm.com/topics/gradient-descent
4. Why Data should be Normalized before Training a Neural Network. Towards Data Science. September 26 2023. https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d
5. Stochastic Gradient Descent (SGD). GeeksForGeeks. September 27 2023. https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/
6. Rectified Linear Units (ReLU) in Deep Learning. Kaggle. September 27 2023. https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning
7. Activation Functions in Neural Networks. Towards Data Science. September 27 2023. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
8. How to Control the Stability of Training Neural Networks With the Batch Size. Machine Learning Mastery. September 27 2023. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/
9. Jason Brownlee. A Gentle Introduction to the Rectified Linear Unit (ReLU). Machine Learning Mastery. September 28 2023. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks
10. René Andrae. Sampled Mean Spectrum generator (SMSgen). Gaia Archive. September 30 2023. https://gea.esac.esa.int/archive/documentation/GDR3/Data_analysis/chap_cu8par/sec_cu8par_a

psis/ssec_cu8par_apsis_smsgen.html
11. Bellas-Velidis & Hatzidimitriou. Unresolved Galaxy Classifier (UGC). September 30 2023. https://gea.esac.esa.int/archive/documentation/GDR3/Data_analysis/chap_cu8par/sec_cu8par_apsis/ssec_cu8par_apsis_ugc.html
12. Overfitting and Underfitting With Machine Learning Algorithms. Machine Learning Mastery. October 6 2023. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/
13. The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks. Analytics Vidhya. October 6 2023. https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/

Code

The code for this dissertation along with the markdown source, figures, etc. can also be found in the following GitHub repository: https://github.com/kiraleos/dissertation

```
import matplotlib.pyplot as plt
import matplotlib.colors
import numpy as np
import os
import pandas as pd
import random
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler, StandardScaler, QuantileTransfor
from sklearn.model_selection import train_test_split

HIST_BIN_SCALE = 100
plt.hist(
  redshifts,
  bins=np.linspace(min(redshifts), max(redshifts), HIST_BIN_SCALE),
  edgecolor='black',
  alpha=1,
  color='grey'
)
plt.xlabel('Redshift')
plt.ylabel('Number of Galaxies')
plt.title('Distribution of Redshifts')

plt.savefig('../figures/redshift_distribution.png')

plt.show()

percentile_values = [25, 50, 75, 90, 95, 99]
percentiles = np.percentile(redshifts, percentile_values)

plt.figure(figsize=(10, 5))
```

```
redshifts_sorted = np.sort(redshifts)
cumulative_curve = np.arange(1, len(redshifts_sorted) + 1) / len(redshifts_sorte
plt.plot(redshifts_sorted, cumulative_curve, color='grey', label='Cumulative Cur

plt.xlabel('Redshift')
plt.ylabel('Cumulative Probability')
plt.title('Cumulative Curve of Redshifts')


line_colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
i = 0
x_ticks = []
for percentile in percentiles:
    plt.axvline(percentile, color=line_colors[i], linestyle='--', label=f'{perce
    x_ticks.append(percentile.round(2))
    i += 1

plt.xlabel('Redshift')
plt.ylabel('Cumulative Probability')
plt.title('Cumulative Curve of Redshifts')
plt.legend()

plt.xticks(x_ticks)
plt.show()

print('Mean, median and Standard Deviation:')
print('  Mean:    ', np.mean(redshifts).round(3))
print('  Median:  ', np.median(redshifts).round(3))
print('  Std Dev: ', np.std(redshifts).round(3))

print('Skewness and Kurtosis:')
print('  Skewness: ', pd.Series(redshifts).skew())
print('  Kurtosis: ', pd.Series(redshifts).kurt())

random_samples = data.sample(n=1)

grayscale_colors = ['0', '0.2', '0.4', '0.6', '0.8']

plt.figure(figsize=(10, 5))
i = 0
for index, row in random_samples.iterrows():
    plt.plot(row.index[1:], row.values[1:], label=f'Galaxy #{index}, z = {row.va
    i += 1

plt.xticks([])
```

```
plt.yticks([])
plt.xlabel('Wavelength (366nm to 996nm)')
plt.ylabel('Flux')
plt.title('Galaxy Spectra')
plt.legend()

plt.savefig('../figures/galaxy_spectra.png')
plt.show()

redshift_ranges = [(0, 0.1), (0.1, 0.2), (0.2, 0.3), (0.3, 0.4), (0.4, 0.5), (0.
num_samples_per_range = 1000
average_flux_per_range = []

for start, end in redshift_ranges:
    filtered_indices = np.where((redshifts >= start) & (redshifts < end))[0]

    if len(filtered_indices) >= num_samples_per_range:
        selected_indices = np.random.choice(filtered_indices, num_samples_per_ra
    else:
        selected_indices = filtered_indices

    average_flux = np.mean(spectra[selected_indices], axis=0)
    average_flux_per_range.append(average_flux)

average_flux_per_range = np.array(average_flux_per_range)

wavelengths = np.arange(186)
grayscale_colors = ['0', '0.2', '0.4', '0.5', '0.6', '0.7']

plt.figure(figsize=(10, 6))
for i, (start, end) in enumerate(redshift_ranges):
  plt.plot(
    wavelengths,
    average_flux_per_range[i],
    label=f'Redshifts {start}-{end}',
    color = grayscale_colors[i]
  )
plt.xlabel('Wavelength (366nm to 996nm)')
plt.ylabel('Flux')
plt.xticks([])
plt.yticks([])
plt.title('Average Flux for Different Redshift Ranges')
plt.legend()

plt.savefig('../figures/average_flux.png')
```

```
plt.show()

redshift_ranges = np.arange(0, 0.6, 0.05)
percentages = []
for i in range(len(redshift_ranges)):
    start = redshift_ranges[i]
    if i == len(redshift_ranges) - 1: end = redshift_ranges[i] + 0.05
    else: end = redshift_ranges[i + 1]
    filtered_indices = np.where((redshifts >= start) & (redshifts < end))[0]
    percentages.append(len(filtered_indices) / len(redshifts))

print('Percentage of Data in Each Redshift Range')
print('| Redshift Range | Percentage |')
print('|----------------|------------|')
for i in range(len(percentages)):
    start = redshift_ranges[i]
    if i == len(redshift_ranges) - 1: end = redshift_ranges[i] + 0.05
    else: end = redshift_ranges[i + 1]
    print(f'| {start:.2f}-{end:.2f} | {100 * percentages[i]:.2f} |')

seed_value = 4
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
SAVE_FIGURES = True

data = pd.read_csv('./training_data/TESTclean.csv')
spectra = np.array(data.iloc[:, 1:])
redshifts = np.array(data.iloc[:, 0])

min_max_scaler = MinMaxScaler()
spectra = min_max_scaler.fit_transform(spectra)

spectra_train, spectra_test, redshift_train, redshift_test = train_test_split(sp

model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(186, 1)),
    tf.keras.layers.Conv1D(filters=256, kernel_size=5, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(filters=256, kernel_size=5, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(filters=128, kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(filters=64, kernel_size=2, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
```

```
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='linear')
])

model.compile(optimizer='adamax', loss='huber', metrics=['mean_absolute_error'])

batch_size = 16
epochs = 20
validation_split = 0.3
history = model.fit(spectra_train, redshift_train, epochs=epochs, batch_size=bat

history_df = pd.DataFrame(history.history)
history_df.to_csv('./history.csv')
model.save('./model.keras')
# model = tf.keras.models.load_model('./model.keras')

plt.scatter(history_df.index, history_df['loss'], label='Training Loss', marker=
plt.scatter(history_df.index, history_df['val_loss'], label='Validation Loss', m
plt.xlabel('Epochs')
plt.xticks(np.arange(0, epochs, 1))
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.figure()
plt.scatter(history_df.index, history_df['mean_absolute_error'], label='Training
plt.scatter(history_df.index, history_df['val_mean_absolute_error'], label='Vali
plt.xlabel('Epochs')
plt.xticks(np.arange(0, epochs, 1))
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Mean Absolute Error (MAE)')

plt.show()

spectra_to_predict = spectra_test
redshifts_true = redshift_test

redshift_predicted = model.predict(spectra_to_predict, batch_size=1000)
redshift_predicted = np.reshape(redshift_predicted, (spectra_to_predict.shape[0]
```

```
mae = np.mean(np.abs(redshifts_true - redshift_predicted))
print('MAE:', mae)

BINS = 200
CMAP = 'inferno'

plt.figure()
plt.hist2d(redshifts_true, redshift_predicted, bins=BINS, norm=matplotlib.colors
plt.plot([0, 0.6], [0, 0.6], '-', c='black')
plt.xlabel('redshift (SDSS)')
plt.ylabel('redshift (cnn)')
plt.colorbar()
plt.title("cnn vs SDSS")

HIST_BIN_SCALE = 150

plt.figure()
plt.hist(redshifts_true, bins=np.linspace(min(redshifts_true), max(redshifts_tru
plt.hist(redshift_predicted, bins=np.linspace(min(redshifts_true), max(redshifts
plt.xlabel('redshift')
plt.title("cnn vs SDSS")
plt.legend()

BIN_SIZE = 0.05

absolute_error = redshift_predicted - redshifts_true
bins = np.arange(0, max(redshifts_true) + BIN_SIZE, BIN_SIZE)
bin_indices = np.digitize(redshifts_true, bins)
mean_errors = [np.mean(absolute_error[bin_indices == i]) for i in range(1, len(b
std_errors = [np.std(absolute_error[bin_indices == i]) for i in range(1, len(bin

bin_centers = (bins[:-1] + bins[1:]) / 2

plt.figure()
plt.errorbar(bin_centers, mean_errors, yerr=std_errors, fmt='o', ecolor='black',
plt.plot([0, 0.6], [0, 0])
plt.xlabel('redshift')
plt.ylabel('zDiff = z(CNN) - z(SDSS)')
plt.ylim([-0.59, 0.59])
plt.xlim([0, 0.6])
plt.title(f"Mean error between CNN and SDSS")

plt.figure()
plt.hlines(mean_errors, bins[:-1], bins[1:], colors=['blue'], linewidth=3.0, lab
plt.hlines(std_errors, bins[:-1], bins[1:], colors=['red'], linewidth=3.0, label
```

```
plt.plot([0, 0.6], [0, 0], c='black', linewidth=0.5)
plt.xlabel('redshift')
plt.ylabel('zDiff = z(CNN) - z(SDSS)')
plt.ylim([-0.59, 0.59])
plt.title(f"Mean error and std between CNN and SDSS")
plt.legend()

print("| redshift bin | mean error | std error |")
print("|--------------|------------| --------- |")
for i in range(len(mean_errors)):
    print(f"| {bins[i]:.2f} - {bins[i+1]:.2f} | {mean_errors[i]:.4f} | {std_erro
```