LMU Munich, Faculty of Physics

26 May 2023

# Convolutional Neural Network for galaxy cluster mass prediction

Manik Dawar, Nader Mostaan, Pranav Kulkarni

Keywords: CNN, Galaxy clusters, Hyperparameter tuning, Redshift

Summary

This lab report presents the results and discussion of an experiment conducted to determine galaxy cluster mass using Convolutional Neural Networks (CNNs). The date from the simulations of eROSITA observations, specifically from the Final Equatorial Depth Survey (eFEDS). The report covers key concepts of galaxy clusters and provides an overview of CNNs and hyperparameters. The results and discussion section explores the types of networks examined, the approach to building the CNN, and regression improvements such as incorporating redshift, data augmentation, retraining on outliers, and using probabilistic loss functions.

# Contents

# 1 Introduction

Galaxy clusters are fundamental structures in the universe, composed of numerous galaxies bound together by gravitational forces. Understanding their properties, such as their mass, is crucial for advancing our knowledge of cosmology and the formation of large-scale structures. Estimating the mass of galaxy clusters is a challenging task due to their complex nature and the limited observability of their constituents. However, recent advancements in deep learning techniques, particularly Convolutional Neural Networks (CNNs), offer promising avenues for improving mass predictions based on observable features [1, 3].

This report focuses on exploring the application of CNNs for predicting the mass of galaxy clusters using their X-ray images. CNNs have demonstrated exceptional performance in image analysis and pattern recognition tasks, making them well-suited for analyzing large-scale astronomical data. By leveraging the spatial brightness distributions observed in X-ray images, combined with additional information such as the redshift of each cluster, we aimed to develop a predictive model capable of accurately estimating the mass of galaxy clusters with the provided dataset.

To achieve this goal, several approaches are investigated to enhance the accuracy of mass predictions. These include optimizing the hyperparameters of the CNN architecture, incorporating redshift information to account for the distance and age of the clusters, retraining the model on outlier data points, and employing data augmentation techniques to augment the diversity and robustness of the training data. Furthermore, we explore the use of probabilistic loss functions to estimate the uncertainty associated with the mass predictions [2].

This report aims to provide valuable insights into the effectiveness of CNNs and regression techniques in accurately estimating the mass of galaxy clusters.

The report is organized as follows: Section 2 provides a conceptual background on galaxy clusters and CNNs, explaining key concepts and techniques relevant to our study. Section 3 details the data collection and splitting process, including the handling of X-ray images and catalogue data. In Section 4, we present the results and discussion, examining the impact of different approaches on the accuracy of mass predictions. Finally, Section 5 offers a summary of our findings, outlines future directions, and concludes the report.

# 2 Conceptual Background

## 2.1 Key concepts of Galaxy clusters (Nader)

Galaxy clusters are among the largest cosmological structures in the universe, composed of $10^2 - 10^3$ galaxies spread over a size of $1 - 5$ Mpc which are bound together by gravitational forces, with masses ranging from $10^{13} - 10^{15}$ solar masses. Among the most studied galaxy clusters, the Virgo cluster is the nearest rich cluster to our galaxy, and the Coma cluster is the nearest regular cluster. Other notable galaxy clusters in the distant, high-redshift universe are SPT-CL J0546-5345 and SPT-CL J2106-5844, which are the most massive galaxy clusters found in the early Universe.

Galaxy clusters are formed out of galaxies (1%), intracluster medium (9%), and dark matter (90%). The intracluster medium (ICM) consists of low density (approximately $10 - 3\,\text{atoms}/cm^3$) plasma between galaxies with a peak temperature between $2 - 15\,\text{KeV}$,

depending on the total mass of the cluster. Dark matter is by far the dominant constituent of the galaxy clusters, and is not detectable optically and is inferred only through its gravitational effects.

ICM contains strong sources of astronomical X-ray emission with luminosities typically in the range of $1043 - 45\, erg/s$, whose main X-ray emission mechanism is thermal Bremsstrahlung. These X-ray sources are extended (ranging from 200 to 3000 kpc) and are the most common bright extragalactic X-ray sources. Their X-ray spectra are not time-varying and show no strong evidence of low energy photoabsorption, in contrast to discrete sources like in the nuclei of galaxies or stellar sources in our galaxy. These observations show that the X-ray emission from galaxy clusters are diffuse, and do not come from an aggregate of compact sources such as galactic nuclei or binary stellar X-ray sources.

To understand the X-ray emission of ICM gas to the cluster mass, several models have been proposed. Here we briefly discuss the model based on the thermal Bremsstrahlung, which is the dominant emission process from a gas composed mainly of hydrogen at temperatures $T_g \simeq 10^8\,\mathrm{K}$ and densities $n \simeq 10^{-3}\,\mathrm{cm}^{-3}$. The emissivity (emitted energy per unit time, volume and frequency) at frequency $\nu$ of an ion of charge $Z$ in a plasma of electrons with temperature $T_g$ is given by

$$\epsilon(\nu) = \frac{2^5\pi e^6}{3m_e c^3}\left(\frac{2\pi}{3m_e k_\mathrm{B}}\right)^{1/2} g_\mathrm{eff}(Z,\nu,T_g)\, Z^2 n_e n_p\, T_g^{-1/2} \exp\Big(-h\nu/k_\mathrm{B}T_g\Big)\,, \tag{1}$$

where $n_e(n_p)$ is the electron (proton) density, $g_\mathrm{eff}(Z,\nu,T_g)$ is the Gaunt factor that takes quantum mechanical effects and distant collisions into account. The surface X-ray brightness $I_X(b)$ at a projected radius (on the visible sky) $b$ is proportional to the emission measure

$$EM = \int dl\, n_e(r) n_p(r)\,, \tag{2}$$

where $l$ is the distance along the line-of-sight through the cluster, at a projected radius $b$. To obtain the mass distribution of gas inside the cluster (which is directly proportional to $n_{e,p}(r)$), a gravitational model of the ICM gas is required. To this end, it is common to make the following assumptions,

- The spatial distribution of galaxies is homogeneous.
- Galaxies are modelled by a collisionless gas. This is justified since the time scale of gravitational interactions is much larger than the time a galaxy traverses across the cluster.
- The velocity distribution of the galaxies is assumed to be a Gaussian with mean velocity $\langle v_r \rangle$ and spread $\sigma_r$, position independent and isotropic.
- The ICM gas is modelled by a hydrostatic, isothermal and spherically symmetric fluid. The hydrostatic assumption is motivated by the fact that the sound propagation time across the galaxy cluster is smaller than the age of the cluster. The isothermal assumption is valid if the thermal conduction is sufficiently rapid.

- The gravitation potential of the galaxy $\phi(r)$ comes not only from the galaxies but also from the ICM gas. Thus, the model has to be self-gravitating.
- The total mass of the cluster is distributed in the same way as the galaxies.

The assumptions about the spatial distributions of the galaxy clusters imply that the phase space density $f(\vec{r},\vec{v})$ takes the form

$$f(\vec{r},\vec{v}) = \rho_{gal}(r)(2\pi\sigma_r^2)^{-3/2}\exp\Big(-v_r^2/2\sigma_r^2\Big)\,. \tag{3}$$

$f(\vec{r},\vec{v})$ is conserved along particle trajectories by the assumption of collisionless ness and Liouville's theorem, so it has to be a function of the constants of motion, that are energy and angular momentum. Furthermore, for an isotropic medium, $f$ cannot depend on angular momentum, so it is only a function of the energy per mass $\epsilon = 1/2v_r^2 + \phi(r)$. Thus, $\rho_{gal}(r) = \rho_{gal,0}\exp(-\phi(r)/\sigma_r^2)$. Since we assumed that the total mass is distributed in the same way as the galaxies, the Poisson equations for $\phi(r)$ take the form

$$\frac{1}{r^2}\frac{d}{dr}\Big(r^2\frac{d}{dr}\phi(r)\Big) = 4\pi m_p G\rho_0\exp\big(-\phi(r)/\sigma_r^2\big)\,. \tag{4}$$

In the above equations, $r_c$ is the core radius of the cluster (that is the radius where the cluster mass density drops to half its central density). Although Eq. 4 does not have analytic solutions, there are analytical approximations known as "King" approximations to the potential as the following,

$$\phi(r) = -4\pi G\rho_0 r_c^2\,\frac{\ln\Big(x+\sqrt{1+x^2}\Big)}{x}\,, \tag{5}$$

where $x = r/r_c$, and $\sigma_r^2 = 4\pi G\rho_0 r_c^2/9$. King approximation will be useful for deriving the ICM gas density. To this end, the isothermal ICM gas follows the hydrostatic equation

$$\nabla P = -\rho_g\nabla\phi\,, \tag{6}$$

where the gas pressure is

$$P = k_{\rm B}T_g/\mu m_p\,\rho_g\,. \tag{7}$$

with $\rho_g$ is the gas density, and $m_p$ the proton mass. The above equations give the following equation for the gas density,

$$\frac{d}{dr}\ln(\rho_g) = -\frac{\mu m_p}{k_{\rm B}T_g}\,\frac{d\phi(r)}{dr}\,. \tag{8}$$

Given Eq. 5, Eq. 8 has the solution $\rho_g(r) = \rho_{g,0}\big[1+(r/r_c)^2\big]^{-3\beta/2}$, with $\beta = \mu m_p\sigma_r^2/k_{\rm B}T_g$. With the obtained expression of $\rho_g(r)$, $EM$ takes the following analytical form

$$EM = \sqrt{\pi}\big(\frac{n_e}{n_p}\big)\rho_0^2 r_c\Gamma(3\beta-1/2)/\Gamma(3\beta)\,(1+x^2)^{-3\beta+1/2}\,. \tag{9}$$

Since $I_X(b)$ is proportional to $EM$, Eq. 9 yields $I_X(b)\propto\big[1+(b/r_c)^2\big]^{-3\beta+1/2}$. This model naturally relates the X-ray emission to the mass distribution of the clusters.

## 2.2 Convolutional Neural Networks and Hyperparameters (Pranav)

### 2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models that are primarily used for analyzing visual imagery. They are designed to automatically and adaptively learn spatial hierarchies of features from data. A CNN is composed of one or more convolutional layers, often followed by pooling layers, fully connected layers, and normalization layers. The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input, producing a 2-dimensional activation map of that filter.

Pooling layers, also known as subsampling layers, reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. There are several types of pooling, among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value. Fully connected layers connect every neuron in one layer to every neuron in another layer.

### 2.2.2 Hyperparameters

Hyperparameters are variables that define the network structure (e.g., the number of hidden units) and determine how the network is trained (e.g., learning rate). Here are some of the key hyperparameters in CNNs, along with their mathematical details:

- **Number of Filters and Filter Size:** The number of filters in a convolutional layer and their size are crucial hyperparameters. They determine the capacity of the network, with more filters requiring more computational resources. The convolution operation, which is the fundamental operation in a CNN, is defined as:

$$(I * F)(i, j) = \sum_m \sum_n I(m, n)F(i - m, j - n)$$

  where $I$ is the input image, $F$ is the filter, $i$ and $j$ are the spatial dimensions (width and height) of the image, and $m$ and $n$ are the dimensions of the filter. The result of this operation is a feature map that represents the presence of the filter's pattern in the image.

- **Stride Size:** The stride size is another important hyperparameter. It determines how much the filter moves after each convolution operation. A larger stride size reduces the spatial dimension of the output volume and thus reduces computational complexity. Mathematically, if the stride size is $s$, the size of the output volume is given by $(W - F)/s + 1$, where $W$ is the input volume size and $F$ is the filter size.

- **Padding:** Padding is used to preserve the spatial dimensions of the input volume, allowing the size of the input to be the same as the output. This is useful for building

deeper networks, as without padding, the size of the volumes would reduce significantly with each convolutional layer. If the padding is $p$, the size of the output volume is given by $(W - F + 2p)/s + 1$.

- **Pooling Size and Type:** The size of the pooling operation and the type of pooling (max, average, etc.) are also hyperparameters. They determine how much to reduce spatial dimensions and what kind of non-linearities to introduce. The most common type of pooling is max pooling, which is defined as:

$$\text{MaxPooling}(A) = \max_{i,j \in \text{window}} A(i,j)$$

where $A$ is the input feature map, and the window is a small spatial block (e.g., 2x2).

- **Learning Rate, Batch Size, and Number of Epochs:** These are standard hyperparameters for any neural network. The learning rate $\alpha$ determines how much to update weights in the gradient descent process, with the weight update rule given by $w = w - \alpha \cdot \nabla J(w)$, where $w$ is the weight, $\nabla J(w)$ is the gradient of the loss function $J$ with respect to the weight, and $\alpha$ is the learning rate. The batch size determines how many samples to use for each gradient update, and the number of epochs determines how many times the learning algorithm will work through the entire training dataset.

- **Regularization Techniques (Dropout, Weight Decay, etc.):** Regularization techniques are used to prevent overfitting in the network. Dropout is a commonly used regularization technique in CNNs, where randomly selected neurons are ignored during training, reducing the chance of overfitting. Mathematically, dropout is implemented by multiplying the output of the dropout layer by a binary mask vector $d$ (with probability $p$ of being zero). Weight decay such as L2 regularization is another technique where a penalty term is added to the loss function, encouraging the network to keep the weights as small as possible. The L2 regularization term is given by $\lambda \sum w^2$, where $\lambda$ is the regularization parameter and $w$ are the weights.

- **Activation Function:** The activation function introduces non-linearity into the network, allowing it to learn more complex patterns. The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU), which is defined as:

$$f(x) = \max(0, x)$$

- **Optimization Algorithm:** The optimization algorithm is used to update the weights of the network. Common choices include Stochastic Gradient Descent (SGD), Adam, RMSprop, etc. Each of these optimizers has its own advantages and disadvantages and they are an active area of research.

- **Fully Connected Layer:** The final layer in a CNN is a fully connected layer, which takes the output of the previous layer and flattens it into a one-dimensional vector. This vector is then passed through a softmax function to produce the final output probabilities for each class. The softmax function is defined as:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

where $x_i$ is the input to the softmax function, and the sum is over all classes.

# 3 Data Organisation(Pranav)

- **Data Collection**: The data consists of an image file and a catalogue file. These were produced from the simulations of eROSITA observations, focusing in this report on the Final Equatorial Depth Survey (eFEDS). The image file contains 3D images of galaxy clusters, while the catalogue file contains information about these clusters, such as their mass and redshift. These files are located in a specified data directory. The `get_data` function is used to load these files and extract the necessary information. For the catalogue file, the function extracts the mass and redshift values, which are used as labels for the model. The mass values are transformed using a logarithmic function (log10) for easier handling in the model. The function returns two arrays: one for the images and one for the labels.

- **Data Splitting**: The data is split into training, validation, and test sets using the `train_test_split` function from the `sklearn.model_selection` module. The split is done in two steps: first, the data is split into a training set (80% of the data) and a test set (20% of the data). Then, the training set is further split into a training set (80% of the original training set) and a validation set (20% of the original training set). This results in a 64%-16%-20% split for the training, validation, and test sets, respectively. The `random_state` parameter is set to 42 in both splits to ensure that the splits are reproducible.

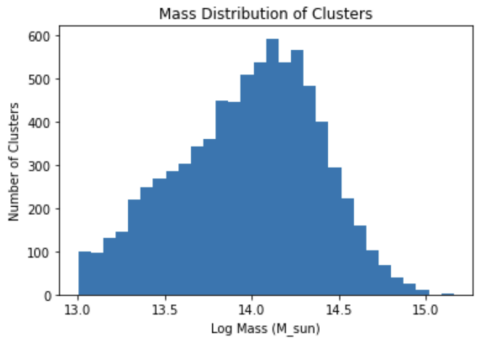
- **Data Formatting**: After the split, the data in each set is converted to a numpy array. This is done to ensure that the data can be properly processed by the TensorFlow model. The shapes of the resulting sets are then printed for verification. In figure1 the histograms of the data for mass, reshifts are plotted.

# 4 Results and Discussion

We first only consider the cluster images and ignore the redshift information corresponding to them. This approach, we shall refer to as the naive approach, where the hyperparameters are not well reasoned, no redshift information is included, or data, augmented. We will start off like this and work our way upward.

## 4.1 Initial approach(Manik)

Starting off with a naive CNN, where we only include the images and not the redshift data, we do not get good results (figure 2). We achieved a minimum validation loss of 0.1471

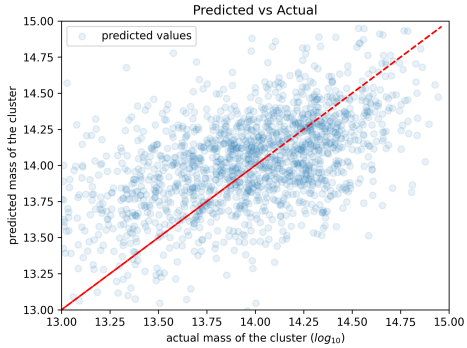



(a) Mass distribution

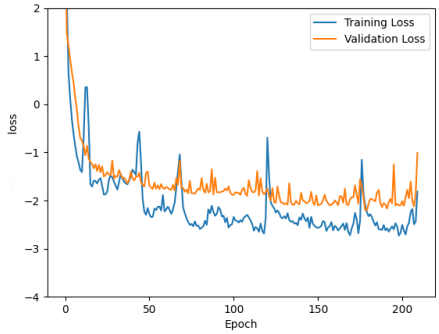

(b) Redshift distribution

Figure 1: Histograms of the mass, redshift distribution of the clusters in the given data

after 99 epochs. In all the comparisons, we train the networks for a max of 300 epochs and patience of 30, where, if the validation loss doesn't improve for 30 epochs, the network with the best validation loss is saved. For more details, please refer to our code.




(a) Fit of predicted vs true mass




(b) Loss curve

Figure 2: Predictions and loss curves from our first model without redshift or data-augmentation)

It should be noted that the plot is done on a log-log scale, so the differences in actual and predicted values of the masses are huge. Next, we identify potential ways to improve upon these predictions and try them one by one.

## 4.2 Regression improvements

We identified four main approaches to try and improve our network's performance:

- **Improve upon the hyperparameters:** number of filters, kernel size, number of convolutional-pooling blocks and number of dense layers added after all the conv-pooling blocks.
- **Include information about the redshift:** There is an inherent ambiguity when we only consider the images. Our CNN essentially only observes the spatial brightness

distributions of the images across the different channels to make predictions about their masses. This alone is not enough, since, depending on the distance of the cluster from us, it would have a different redshift, and consequently, a different brightness profile.

- **Training on outliers:** We could identify all the mass regions where the network is most likely to make bad predictions and retrain it only on those regions.
- **Data-augmentation:** We have access to only a limited amount of data $\approx$ 8000 images. Maybe this is not enough to train a CNN. Since we do not have access to more data, we could augment our existing data and feed that in as extra data. Here we perform rotations to the images as augmentations.

#### 4.2.1 Hyperparameter tuning (Manik)

One natural way to seek improvements in CNN predictions is to simply play around with different hyperparameter combinations (section 2.2) and see which one performs best. The aim is to have an architecture that is complex enough to execute the tasks, but at the same time, not so complex that it is impossible to train with the available computing resources and training data.

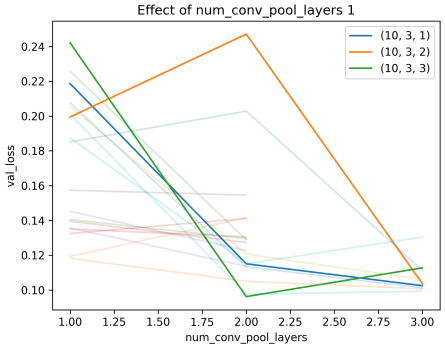
To this end, we identified 4 seemingly crucial hyperparameters that could affect the predictive powers of our CNNs:

- **Number of filters:** There could be several different aspects to the input images. Those should be captured with different filters. If the number of such filters is too low, then CNN will not be able to appreciate the images in their full glory. If the number is too large, then we would have an unnecessarily large number of weights to train and risk overfitting. Since there are ten input channels in our images, 10 filters is a good guess for the minimal number of filters required. There may even be more features that we do not know about, so let's also try larger numbers of filters.

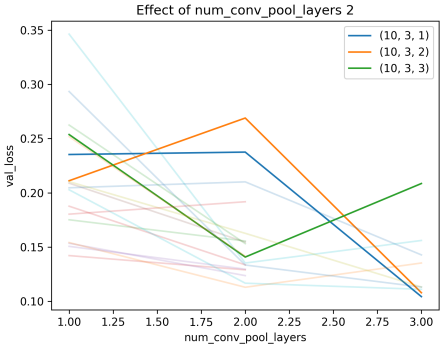
  Therefore we tried filter numbers of {10, 20, 30} for the first convolutional layer. Then, if there were more convolutional layers, we just added 10 to the number of filters in each successive layer. For example, if we started with 10 filters in the first convolutional layer, then in the second one, we used 20 filters, and in the third one, 30, and so on.

- **Kernel size:** We want to strike a balance between the number of weights devoted to each feature and the consequent complexity of training them. So, based on common practices, we tested filter sizes of {3, 5, 7}.
- **Number of convolutional-pooling layers:** The more the number of such layers we have, the more detail our network can potentially extract from the input. But we still want our network to remain trainable and prevent overfitting, so we tested a number of conv-pooling layers {1, 2, 3}.
- **Number of dense layers:** These layers at the end are responsible for interpreting the outputs of the preceding convolutional layers. Here too, striving to find a balance between complexity and trainability, we tried {1, 2, 3} dense layers, with a number of neurons 64, 44 and 24 respectively.

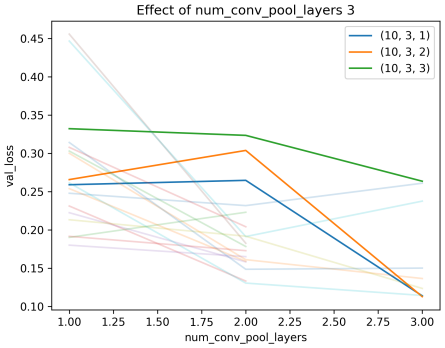
We trained all possible combinations of these hyperparameters for 300 epochs and early-stopping with patience 30. There are $3^4 = 81$ such combinations, but only 53 out of those are possible, because, with each conv-pooling block, the dimensions of the data are reduced. With these tests, the goal was to identify trends in the validation loss. We plotted these trends for each combination, keeping 3 hyperparameters fixed and varying the fourth one. This allowed us to spot the biggest differentiator in performance - a number of convolutional layers. We saw that for almost all combinations of the three complementary fixed hyperparameters (number of features, filter size, dense layers) - increasing the number of convolutional layers lowered the validation loss.




(a) Smallest loss




(b) Medium loss




(c) Largest loss

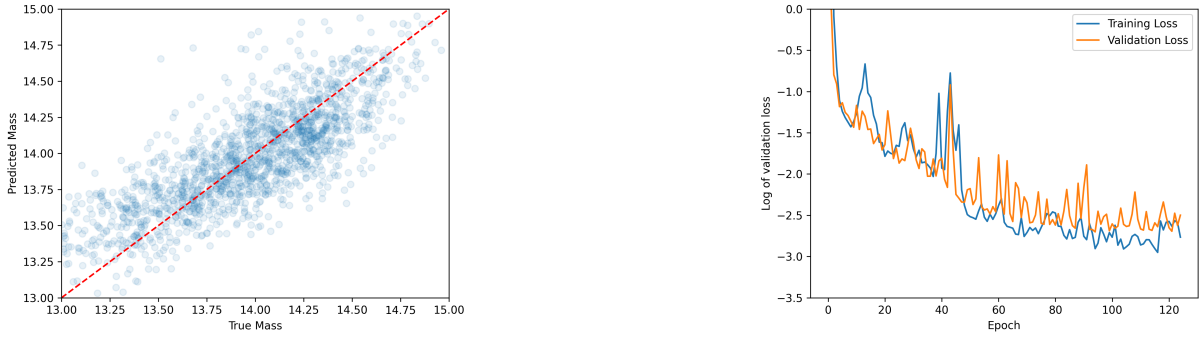
Figure 3: Effect of the number of convolution-pooling blocks on validation loss. Each label tuple denotes the number of filters, filter size and number of dense layers respectively.
Each training was done for three different initializations of the network. That is why there are three figures, ordered according to the performance of the initialization, with the best performing one shown first.

### 4.2.2 Including Redshift (Pranav)

The redshift information is included in the model by concatenating it with the output of the last layer of the CNN. This is done using the Concatenate function from Keras. The concatenated output is then passed through a final dense layer to produce the model's output.

This means that the model is trained to predict the mass of the galaxy cluster based on both the image data and the redshift of the cluster.

Redshift is a measure of how much the light from an object in space is stretched due to the expansion of the universe. In the context of galaxy clusters, a higher redshift means that the cluster is further away and its light is older. By including the redshift in the model, the model is able to take into account the distance and age of the clusters, which can affect their observed properties. This improved the accuracy of our model's predictions.




(a) Fit of predicted vs true mass

(b) Loss curve

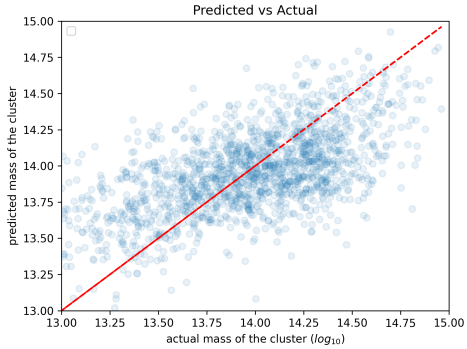
Figure 4: Predictions and loss curves from our model with redshift and without data-augmentation

### 4.2.3 Data augmentation(Pranav)

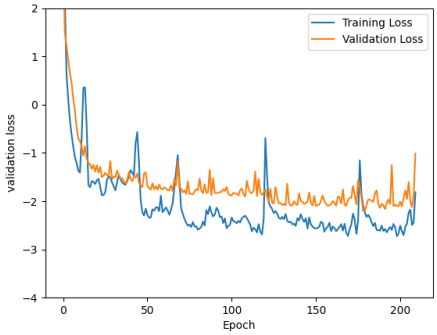
Data augmentation techniques are applied to the training data to artificially increase the size of the dataset and help the model generalize better. Techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks. The techniques used in our model are horizontal flipping and rotation of the images. In our code, data augmentation is applied using the `Sequential` model from Keras. Two data augmentation techniques are used:

- `RandomFlip("horizontal")`: This function randomly flips each image horizontally (i.e., about a vertical axis). This can help the model generalize better to new data by providing it with more diverse examples during training. For example, a galaxy cluster that appears on the left side of an image might appear on the right side in a flipped version of the image.
- `RandomRotation(0.1)`: This function randomly rotates each image by a factor of 0.1. The rotation is a random value in the range [-10%, +10%] multiplied by 180 degrees. This can help the model generalize better by providing it with examples of galaxy clusters at different orientations.

The augmented data is then used to train the model. The logic behind using data augmentation is that it can help the model generalize better to new data. By training the model on more diverse data, the model can learn more robust features and is less likely to overfit to the training data.




(a) Fit of predicted vs true mass



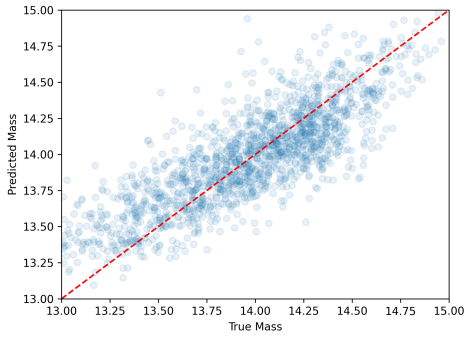

(b) Loss curve

Figure 5: Predictions and loss curves from our model with only data-augmentation)

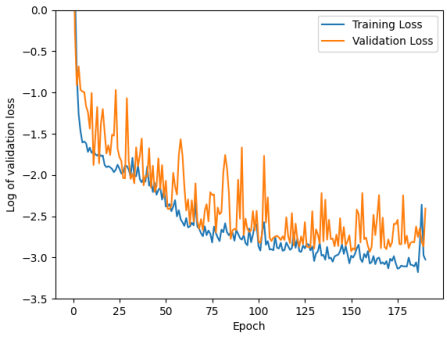
### 4.2.4 Combining the above(Manik)

Based on the attempts for improvement above, and striking a balance between complexity and trainability, we decide to go for the maximal number of conv-pooling blocks that we tested i.e. 3, while keeping the other hyperparameters at their minimal tested values to minimize complexity i.e. the number of dense layers = 1, filter size = $3 \times 3$ and number of features = 10. Additionally, we include the redshift information in the input and augment the available data.

The results improve drastically on combining these modifications, as shown in figure 6. Here we achieved a minimum validation loss of 0.0533 after 161 epochs; a significant improvement over the 0.1471 we got without all these modifications.




(a) Fit of predicted vs true mass




(b) Loss curve

Figure 6: Predictions and loss curves from our model after combining redshift, data-augmentation and hyperparameter search

### 4.2.5 Probabilistic loss (Manik)

We used CNNs to predict the masses of galaxy clusters from their X-ray images from 10 different channels and information about their redshifts. Now we also want to predict the

uncertainty associated with our predictions. This can be done by modifying the output of our CNN and training it on a different cost function - the negative log-likelihood.

The likelihood is given by:

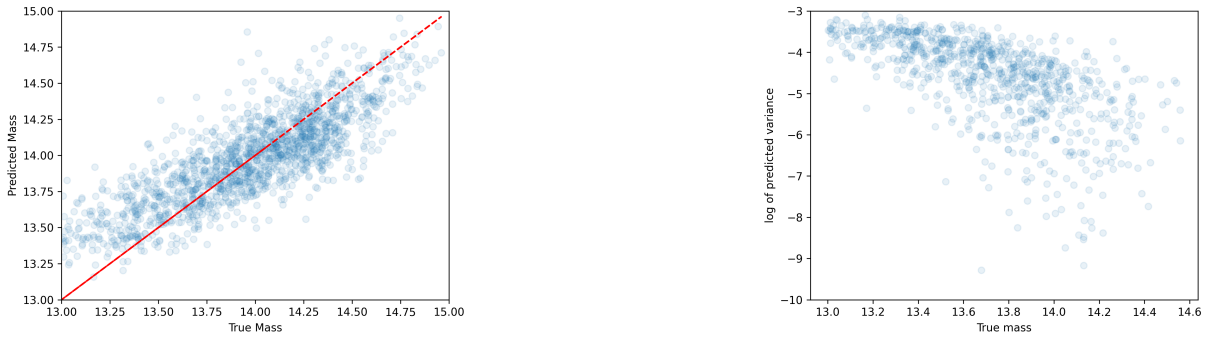
$$p(y|x,\theta) = \frac{1}{\sqrt{2\pi\sigma(x,\theta)^2}} \exp\left(\frac{(y - f(x,\theta))^2}{2\sigma(x,\theta)^2}\right). \tag{10}$$

So the negative log-likelihood is:

$$-\log p(y|x,\theta) = \frac{1}{2}\log(2\pi) + \log(\sigma(x,\theta)) + \frac{(y - f(x,\theta))^2}{2\sigma(x,\theta)^2}. \tag{11}$$

The first term is just a constant shift, so it can be ignored. The third term is proportional to our original cost function - the mean-squared error, but weighted by the inverse of the uncertainty of our prediction - so the smaller the uncertainty, the better our predictions ought to be to offset the small term in the denominator. The second term penalizes large uncertainties; without it, our network might as well be maximally uncertain and earn a good score.

We already achieved decent predictions using MSE, so we kept the weights of our best model trained with it, replaced its output layer, such that now it had two outputs instead of one - one for the mass and the second one for the uncertainty, and trained only the last dense layer before the output layer, to teach it to predict not just masses, but also their uncertainties. The mass predictions, therefore, did not change much, but we now also achieved uncertainty predictions ref. figure 7.




(a) Mass predictions

(b) Variance in predictions

Figure 7: Predictions on retraining with log-likelihood

The loss curves are shown in figure 8.

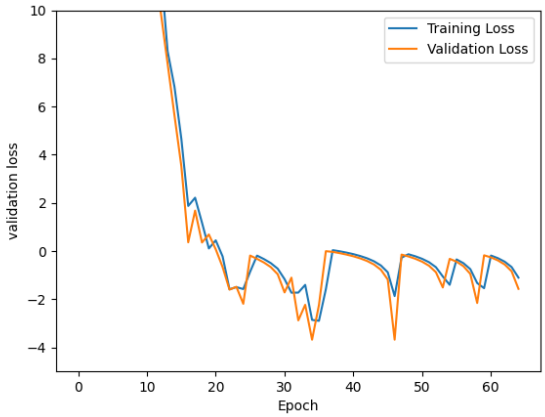



Figure 8: Loss curves from the retraining with the log-likelihood cost function

# 5 Summary

In this lab report, we focused on exploring the application of Convolutional Neural Networks (CNNs) for predicting the mass of galaxy clusters using their X-ray images. We introduced the importance of estimating the mass of galaxy clusters and the potential of CNNs in this task. We investigated various approaches to enhance the accuracy of mass predictions, including optimizing hyperparameters, incorporating redshift information, retraining on outlier data points, and employing data augmentation techniques. Additionally, we explored the use of probabilistic loss functions to estimate the uncertainty associated with mass predictions.

We provided a conceptual background on galaxy clusters and CNNs, explaining key concepts and techniques relevant to our study. We described the data collection and organization methods, including the handling of image and catalogue data. The data was split into training, validation, and test sets, and we applied data formatting techniques to prepare the data for training our CNN model.

Our results and discussions showed the impact of different approaches. We started with an initial naive CNN approach without considering redshift or applying data augmentation techniques, but we identified the limitations and the need for improvements. We then discussed the impact of hyperparameter tuning, including the number of filters, filter size, number of convolutional-pooling layers, and number of dense layers. We also explored the inclusion of redshift information and data augmentation techniques, which led to improved predictions.

Finally, we highlighted the use of probabilistic loss functions to predict the uncertainty associated with mass predictions. By modifying the model's output and training it with the negative log-likelihood, we were able to improve the predictions and estimate the uncertainty of the mass predictions.

# References

[1] Stefano Borgani. Galaxy clusters: challenges for the xxi century. *Journal of Physics: Conference Series*, 280(1):012002, 2011.

[2] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *arXiv preprint arXiv:1802.10501*, 2018.

[3] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.