Anurag Sawarkar

Physics Applications of AI

University of Geneva

Spring 2023

Application of AI for Galaxy Identification

Contents

1 Introduction and Motivation

Galaxies are mesmerising astronomical structures; a collection of *billions* of stars systems, gas, dust and dark matter. Observational astronomers quickly discovered that while each galaxy is unique, it could be classified into four main classes: *elliptical*, *spirals*, *barred spirals*, and *irregular* galaxies. To visualize the range of morphologies, *Edwin Hubble* devised the *tuning-fork* model, seen in Fig.1.

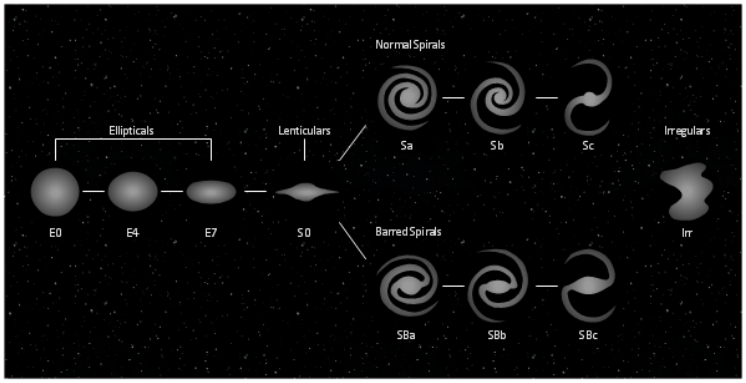


Figure 1: Hubble's tuning-fork model [1]

The study of galaxy morphologies provides us with a much better understanding of the formation and evolution of stars, galaxies, and ultimately the universe. However, to do so, scientists require a *large* data set of a variety of different galaxy morphologies to study. One could, for instance, construct a catalogue of galaxies sorted by their morphologies through individually observing each galaxy and classifying them. However, this process is inefficient and time-consuming for scientists who could've been otherwise have been studying their intrinsic properties. What if instead we could harness the power of *Machine Learning* to automate this process - that is, feed a collection of galaxy images *automatically* captured by a telescope into an algorithm that *automatically* classifies it into a morphology class?

Indeed, that is the focus of this investigation - specifically, to use the citizen-scientist project *The Galaxy Zoo* dataset to *train* a model which would automate this process of classification. The investigation is to then tweak parameters to obtain an efficient and performing model.

2 Context

This section will provide information about what this investigation is based on and its aims.

2.1 The Galaxy Zoo Project

The Galaxy Zoo project is a citizen-scientist project whereby volunteers were tasked with classifying a galaxy by their morphologies. For a given galaxy, volunteers were asked a series of questions about various properties of the galaxy, such as whether it was spirally, elliptical, if it had a bulge, if there was anything peculiar about it, etc. This can be visualized as a flowchart/table seen in Fig.2.

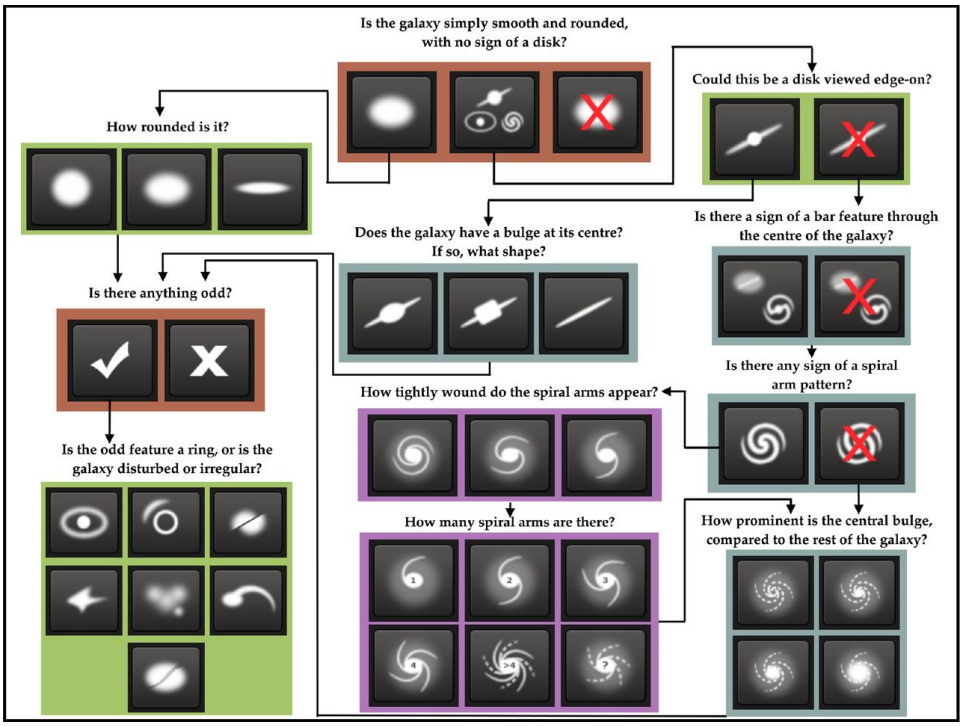


Figure 2: Galaxy Zoo Project Flowchart [2]

We gather our data and inspiration from the Kaggle Galaxy Zoo project; a challenge which uses a collection of 61,572 galaxy images compiled with labels from the Galaxy Zoo project (*the form of the data will be explored in Sec.3*) to create a machine learning model which can classify galaxies.

2.2 Aims

The main aim of this challenge is to firstly create *deep learning models* which address simplified versions of the first two top-level questions seen in Tab.1, and then to employ various "model-strengthening" strategies such as *data augmentation* and *dropout* to see if improves its performance.

Task	Question	Responses	Next
01	*Is the galaxy simply smooth and*	Smooth	07
	rounded with no sign of a disk?	Features or Disk	02
		Star or Artifact	**End**
02	*Could this disk be viewed edge-on?*	Yes	**End**
		No	**End**
⋮	⋮	⋮	⋮
07	*How rounded is it?*	Completely Round	**End**
		In Between	**End**
		Cigar-Shaped	**End**
⋮	⋮	⋮	⋮

Table 1: First two simplified top-level questions adapted from [2]

The first task is as follows:

We only consider Q1 from Tab.1 (3 answers) and samples where at least 80 % of participants gave the same answer. We assumed this to be a certain classification and transformed them to one-hot encoded labels. The task for this part of the exercise is to use the image to predict the corresponding label.

While the second-task is given by:

We consider the second layer of questions from the original Kaggle challenge, i.e. Q2 and Q7. Here, we do not use one-hot encoded labels, but the original floats that range between 0 and 1, thus making the classification problem a regression problem.

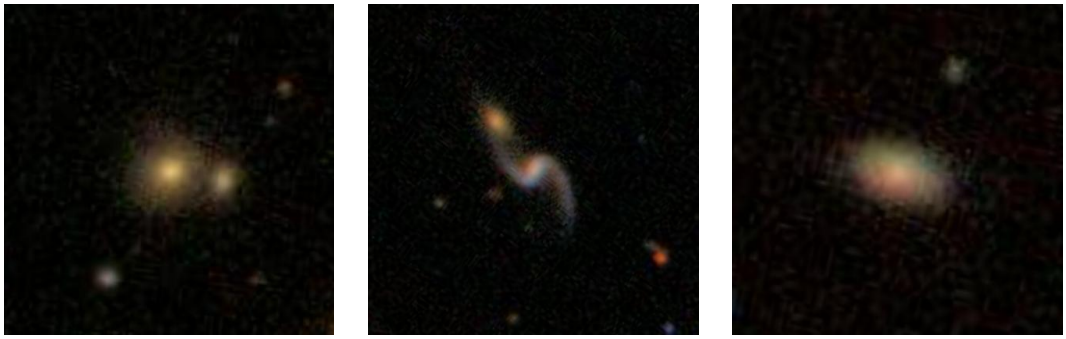
To achieve this, we will use `Python` to create a *Convolution Neural Network* (CNN), a *deep learning model* which operates on the principle of automatically learning *features* from an image in a manner that is highly accurate and efficient, allowing for tasks such as image recognition - perfect for our needs. The first task can be interpreted as a *classification* problem; each image has a definitive label of galaxy class. The second is a *regression* problem; each image has an associated label with a certain probability, as defined by volunteers in the Galaxy Zoo. Although, as we will see with the inherent data structure, it is possible to transform the data such that for tasks as a classification or regression problem.

3 Method

This section will provide information about the method used to process data, create the models, and train it.

3.1 Data Structure and Processing

As previously mentioned, the data was sourced from Kaggle's Galaxy Zoo challenge [3]. This consisted of approximately 60,000 galaxy images, with an associated label file containing the probabilities that a certain galaxy is a certain class (Tab.2)



(a) Galaxy 100123 (b) Galaxy 102243 (c) Galaxy 107527

Figure 3: Examples of Galaxy Images provided by Kaggle [3]

GalaxyID	Class1.1	Class1.2	Class1.3	Class2.1	Class2.2	Class7.1	Class7.2	Class7.3
	Smooth	*Features*	*Artifact*	*Edge-on disk*	*Visible Disk*	*Round*	*Elliptical*	*Lenticular*
100123	0.462	0.456	0.081	0.000	0.456	0.388	0.074	0.000
102243	0.000	0.999	0.001	0.070	0.929	0.000	0.000	0.000
107527	0.528	0.444	0.027	0.000	0.444	0.019	0.482	0.027
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2: Sample Labels data for Galaxies in Fig.3 from Kaggle [3]

One may notice that each level question probability is *normalized* such that their probabilities add up to 1. The first-level question, consisting of Class 1.1 to 1.3, and the second-level, classes 1.3 and 2.2 to 7.3, both add up to 1 respectively. To calculate the probabilities of classes 1.3 to 7.3, the probabilities of the first-level questions are also taken into account following the logic of the Tab.1. For instance, 46.2% of participants believed that Galaxy 100123 had smooth features (class 1.1). According to Tab.1, they follow the flowchart to task 7, where 83.9% of participants believed it was completely round (class 7.1). The score on Class 7.1 is therefore the product of the two percentages: $0.456 \times 0.839 = 0.388$. Information about higher level questions is hence fully *embedded* in the probabilities of lower level questions.

Now that we have this data, it is important that it is appropriately *pre-processed* to ensure it is in the right format to be used for model training.

The images that are provided are 424x424 RGB JPG images of galaxies. To process them for training, they needed to be cropped to 64x64 for computational reasons.

For the labels, the processing has to be done depending on the task at hand. For the first classification task, only Classes 1.1 to 1.3 corresponding to first-level questions are considered. We then filter out all Galaxies with confidence under 80% (i.e. only consider rows that contain a value > 0.8). However, we notice in Tab.3, that class 1.3 only contains 2 compared to the multi-thousand samples of other classes. Even when reducing this confidence to 50%, this number doesn't increase drastically. Therefore, while it may be considered cheating, the third class was *disregarded* as it would be insignificant and the model wouldn't have enough samples to be allowed to classify it as such. This leaves us with a *binary classification* task. To turn the labels into *certain classification*, the labels were turned into 0 (class1.1) or 1 (class1.2) depending on which class had the highest confidence.

Confidence	Count		
	Class 1.1	Class 1.2	Class 1.3
80%	8110	16135	2
50%	25860	34105	44

Table 3: Class Counts for First Task for 80% and 50% confidence

For the regression task, we considered data only from class 1.3 and 2.1 to 7.3. Since classification into either class 2 or class 7 is already has the the decision in class 1 *encoded*, it was unnecessary to consider the latter aside from 1.3. For this task, the confidence was chosen to be at 70%, as it strikes a balance between providing a data set that is as general as possible, while still being sufficiently reliable for the model to train on. Tab.4 shows the counts for these classes.

In the Python program used to create and train models, the processing of images has several user-defined *configurations*, including crop-size, whether to switch to gray-scale, or *normalize* the RGB pixel values (these reduce the pixel range from 0-255 to 0-1 for computational reasons). For labels, the confidence was configurable.

Confidence	Count					
	Class 1.3	Class 2.1	Class 2.2	Class 7.1	Class 7.2	Class 7.3
70%	6	2835	15124	3269	2217	52
50%	44	5042	24821	8434	8069	578

Table 4: Class Counts for Second Task for 70% and 50% confidence

Once again, we observe a very low sample count for class 1.3 and 7.3 in comparison to the other classes. Therefore, once again we disregard it for reasons stated previously. However, since this is *not* going to be turned into a 0 or 1, it was necessary to normalize the probabilities for the remaining four class.

3.2 Model

The generalized CNN model Architecture can be seen in Fig.4, where an Input layer is connected to an Augmentation Layer, Convolution Layer, Flattening, Dense and Output Layer in a *sequential* manner.

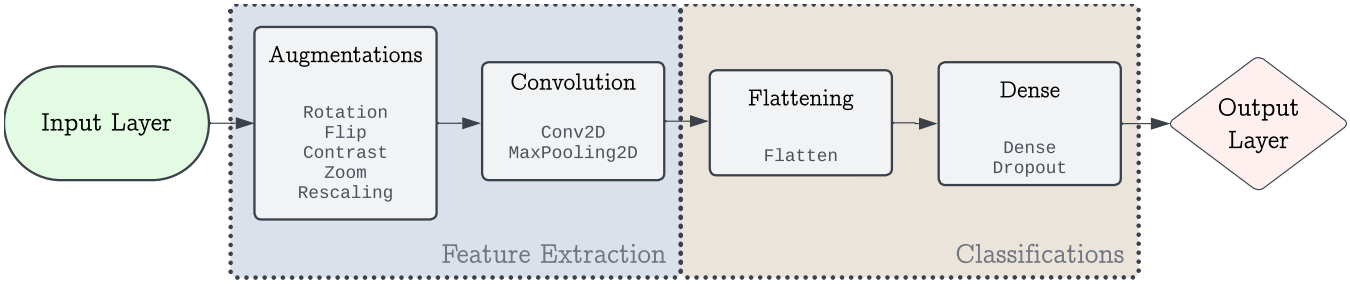


Figure 4: General CNN model used in this project

We start with the augmentation layer. If this block is active, it will apply random rotations, flips, contrasts, zooms and re-scale as specified by a configuration. This process is called *Data Augmentation*. The purpose of this process is to effectively *inflate* the size of the training data for the model by applying various random transformations such as those aforementioned. This increases the diversity of the training thereby enabling the model to be better at generalizing for unseen data as it is provided to opportunity to learn additional features. It is equally important to not include augments which may *change* the meaning of an image. For instance, the stretch application was not applied as it could, for instance, change the definition of a spherical galaxy to an elliptical.

Furthermore, data augmentation can also reduce *over-fitting*. This is when the model is trained *too well* to only the training data because it learns patterns which aren't related to intended subject. This means that if it were to be introduced to unseen data, it would perform poorly, even if it does excellently on training data. Data augmentation's introduction of unseen variations allows for the model to better generalize.

Next are *Convolution* and *Pooling* layers. This is a standard implementation in CNN models, whereby convolution layers apply mathematical *filters* over the image to detect and learn specific features about an image. This may be the spiral of a galaxy, for instance. The output, known as the *feature map* contains information about where the feature is located, and is passed into the pooling layer where its size is reduced by *summarizing* it for the benefit of decreasing computation cost. These layers are where *feature extraction* of the image take place.

The *Flattening* and *Dense* form the *Fully-Connected* segment of the model which actually performs the *classification* or *regression* depending on the task and structure. It starts by passing the output of the feature extraction segment into the flatten layer, where it is turned into a *flat* format compatible with the dense layers. The dense layers is where relations between different features is established, which is where the model *learns*. Furthermore, a *dropout* layer is added, which drops feature nodes from the dense layer

network. While this may seem counter-intuitive to learning, it ensures that over-fitting doesn't occur by simplifying the model. This is finally passed into the output layer, where we get our intended output.

3.2.1 Set parameters

For the *Binary Classification* CNN model, the architecture with parameters that worked well were:

1. **Augmentation Layer**: A `RandomRotation` with a full 360 degree rotation, `RandomFlip` horizontally or vertically, `RandomContrast` with a contrast parameter 0.5, `RandomZoom` range of $\pm$ 25%, and a `Rescaling`
2. **Convolution**: Here three `Conv2D` layers with 32, 64, and 128 number of filters of filter size (3,3) pixels with standard `ReLU` activation and `padding` of `valid` since we do not care about the borders. In between each `Conv2D` layer, we have `MaxPooling2D` layers with size (2,2)
3. **Flattening and Dense**: Here the output of the convolution layer is passed to the `Flatten` layer, followed by a single `Dense` layer of 128 units, once again with `ReLU` activation. A `Dropout` layer with parameter 0.3 (30%) was also added
4. **Output**: Lastly, we have a `Dense` output layer of 1 unit representing the fact that we want 1 output that is the classification given by the model (0 or 1 for 1.1 or 1.2), with a `Sigmoid` activation
5. **Compile Parameters**: While not explained in this report, the compile fit parameters were the standard optimizer `adam` with `learning rate` 0.001, and the `loss` (*weight penalty*) was `Binary Cross-Entropy`

For the *Regression* CNN model, the architecture is similar, but extra Dense layers were added for the regression task:

1. **Augmentation Layer**: *Same as above*
2. **Convolution**: *Same as above*
3. **Flattening and Dense**: Here the output of the convolution layer is passed to the `Flatten` layer, followed by two `Dense` layers of 128 and 64 units, with `ReLU` activation and with a `Dropout` layer with parameter 0.3 added between each
4. **Output**: Lastly, we have a `Dense` output layer of 4 units representing the fact that we want 4 outputs that is the probabilities for each class with a `linear` activation
5. **Compile Parameters**: the compile fit parameters were the standard optimizer `adam` with `learning rate` 0.001, and the `loss` was `mean_squared_error`

The program that was created for this task features configuration files which allow a user to tune these hyper-parameters if they require.

3.3 Evaluation Metrics

To evaluate the performance of the model, we can use several metrics, subject to the type of problem. For binary classification, the metrics that were used were *accuracy*, *F1-score*, *ROC-AUC*, and a confusion matrix. For Regression problems, there are no *discrete* answers, hence instead we use the *mean-squared-error* (MSE) and the R^2 metric. A confusion matrix will also be used to see how it copes under classification.

- **Accuracy and F1-Score**: Accuracy measures the proportion of *correctly* classified samples across all classes, while the F1-score quantifies the false classifications, considering the influence of the bias of the size of each class. For instance, a model's accuracy may be extremely high, but if its F1-score is low, then that implies that the model works well to classify one class really well, but not the other. An accuracy and f1 score with similar magnitudes implies a balanced class classification. For our dataset, which contains unbalanced data, as seen in Tab.3 and Tab.4, a high and close to accuracy score for f1 would indicate that the trained model is capable of classifying despite the imbalance. Both range from 0 to 1, with 1 being ideal.

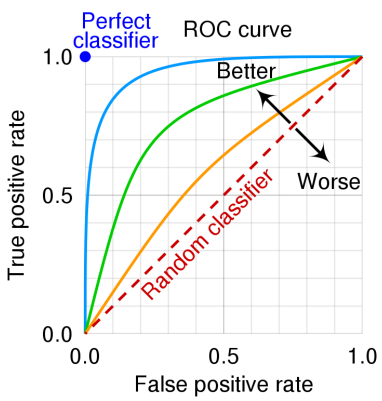
- **ROC-AUC**: The *AUC* or Area Under the *Receiver Operating Characteristic* (ROC) curve is a metric typically used in binary classification. The ROC is a curve which plots a model's correct classification against incorrect classifications as a function of *classification thresholds* (Fig.5.a). The AUC is the area under this ROC, ranging from 0 to 1. If 0, it means all predictions are wrong, if 0.5, it is as good as randomly guessing. The ideal is to get to 1, where all predictions are right.

- **Confusion Matrix**: The confusion matrix is an additional graphic which provides an overview of the prediction vs. actual classifications. It is a square diagram with cells representing the number of instances a classification was made for each class. Diagonal elements mean the prediction class and the actual class are the same, while off-diagonal represent mis-predictions. An example of such a confusion matrix for sample data can be seen in Fig.5.b.

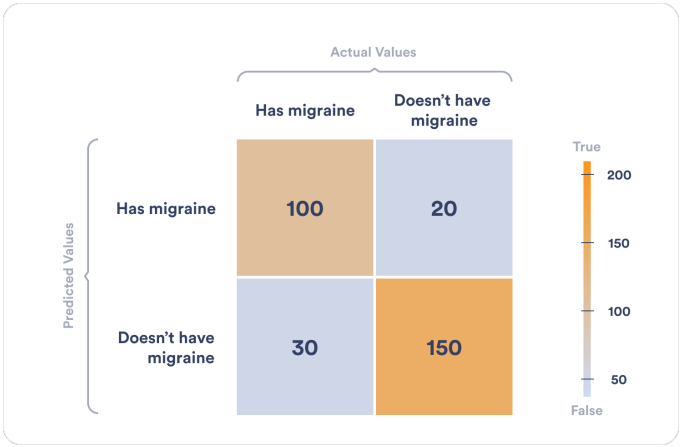
- **MSE**: MSE is a simple metric that takes the average of the square of the differences between predicted and actual values. Ranging from 0 to 1, a lower MSE means a smaller squared deviation from actual values, and hence better performance.

- **R^2**: The R^2 quantifies how well a model's prediction fits the data compared to taking a simple average. A simple example is seen in Fig.5.c whereby on the left we have a fit that is simply the average, while the right is a regression with a lower MSE and a higher R^2. In essence, it *describes* the variance in the data better.

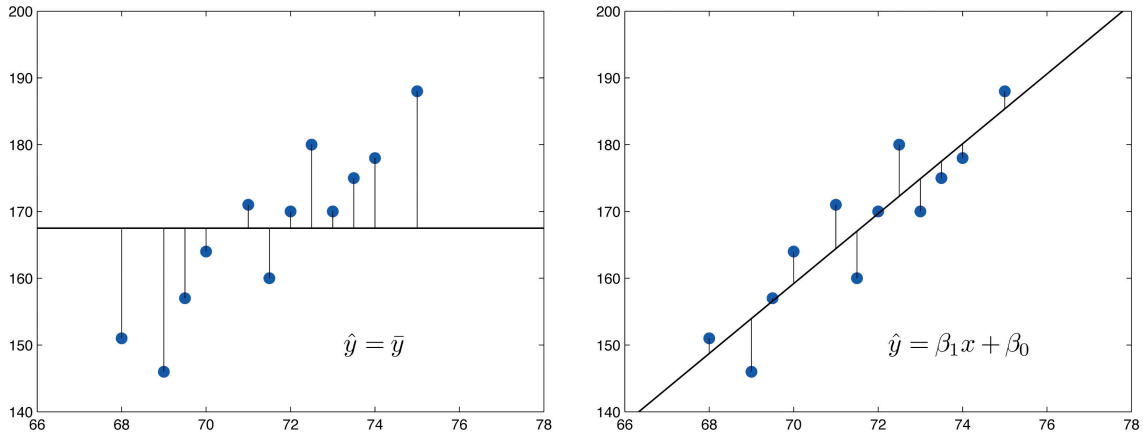
To evaluate training, we will track the accuracy and loss over training epochs for binary classifications, and the MSE over epochs for the regression task.



(a) ROC curve [7]



(b) Example of a confusion matrix [8]



(c) Average fit (left); Lower R^2 vs. Regression fit (right); higher R^2 [9]

Figure 5: Graphics illustrating several evaluation metrics

3.4 Training Pipeline

The training pipeline which explains the program created for this task, holistically consists of pre-processing data into an appropriate format, splitting the data into *training*, *validation*, and *test* datasets, building the model based on the configuration, actually training the model using training and validation data, and finally evaluating the performance of the model using the test data.

1. **Image and Label Pre-processing**: In general, the images and labels were imported, with the images going through the pre-processing of cropping to 64x64, followed by a grey-scale shift *if* specified by the config. For binary classification, the labels were firstly filtered by if they were above a specified confidence level (80%) and if they even had a corresponding galaxy photo in the images dataset. Then, they were converted to a 1D label of either 0 or 1 representing class 1.1 or 1.2. For the regression task, the image processing stayed the same (aside from reducing the confidence to 70%, but the labels were normalized to account for the classes that were removed (1.3 and 7.3). The images and corresponding labels were packaged together into a single tensorflow dataset.

2. **Dataset Processing**: Before the data can be put into the model, the dataset was *split* into training, validation and training with a ratio of 60-20-20. The training data set is where the model trains, and for each epoch validates with the validation data. The training data is used towards the end when evaluating the trained model's performance on unseen data.

3. **Model Building and Training**: The model is built based on parameters in Sec.3.2.1 via a *functional* framework. The model is trained on 25 epochs, with `shuffle` to get a better generalization and reduce over-fitting by shuffling the test set in every epoch. Furthermore, we use the `callback` `ModelCheckpoint` to save the model's *weights* for each epoch so that we may restore the best-performing parameters for evaluation. In the case of binary classification, we monitor and use the greatest AUC validation parameter, while for regression, we use the least MSE validation parameter. There was also the possibility of using the `EarlyStopping` callback to stop the training when no improvement was detected for some epochs to reduce computational costs. However, to fully study the different parameters, this callback was not used. Instead, during evaluation the best performing parameters were restored so that any damage done by over-training could be reversed.

4. **Model Evaluation**: The test data was run under `model.predict()` to get prediction labels which could then be compared to test data labels. The appropriate metrics discussed in Sec.3.3 were calculated and saved in a metrics file for the appropriate task type. A historical training evolution of accuracy and loss with an ROC curve for illustration purposes / MSE and loss for either model is plotted and saved. A confusion matrix is also generated for classification. For binary classification, the labels are already formatted and used as is. For the regression model, the classification was taken as the class with the highest probability.

The biggest challenge was developing the Python script using configuration files in a concise manner, primarily due to personal inexperience with the language, but also machine learning techniques. Of course, the course aided tremendously, but there was a lot of trial and error with the development of the Python scripts. Countless hours were spent approaching the problem, developing a model for it, testing various parameters (training took the longest), determining which parameters would be appropriate to study and finally tweaking the model so it performs well within the chosen metrics. The results in the subsequent sections are a portion of *presentable* total trials.

4 Results and Analysis

This section will cover the different models tested, their metrics and implications

4.1 Binary Classification

4.1.1 Baseline Gray Scale Model Performance

To set a baseline, we will use the CNN model outlined in Sec.3.2.1 without any sort of additional Augmentation, or Dropout layers. The confidence is set to 80% and the image is loaded in *gray-scale*. Other image parameters are unchanged from what was previously described.

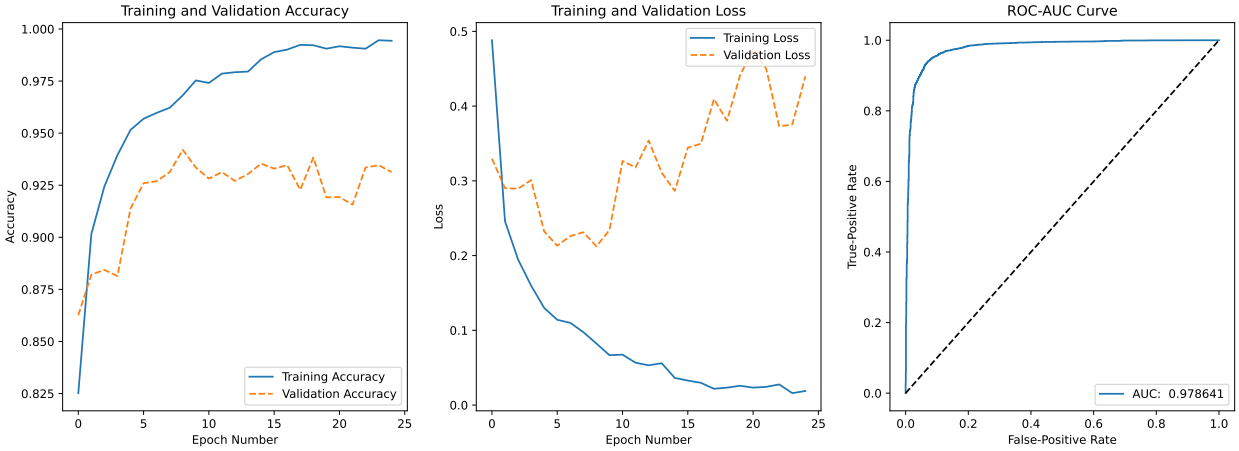


Figure 6: Training and Valid accuracy over epochs with ROC-AUC curve

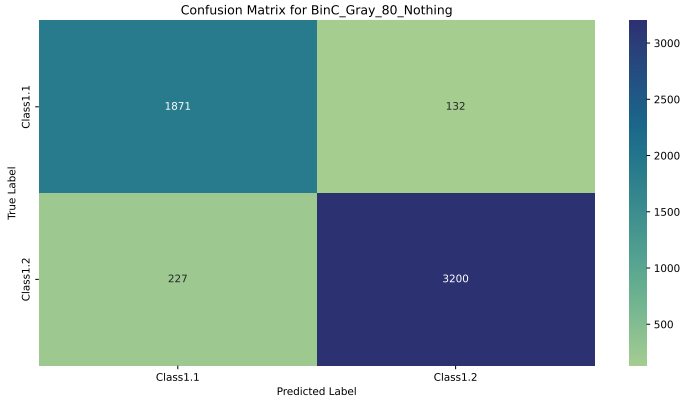


Figure 7: Confusion Matrix for baseline model using training data

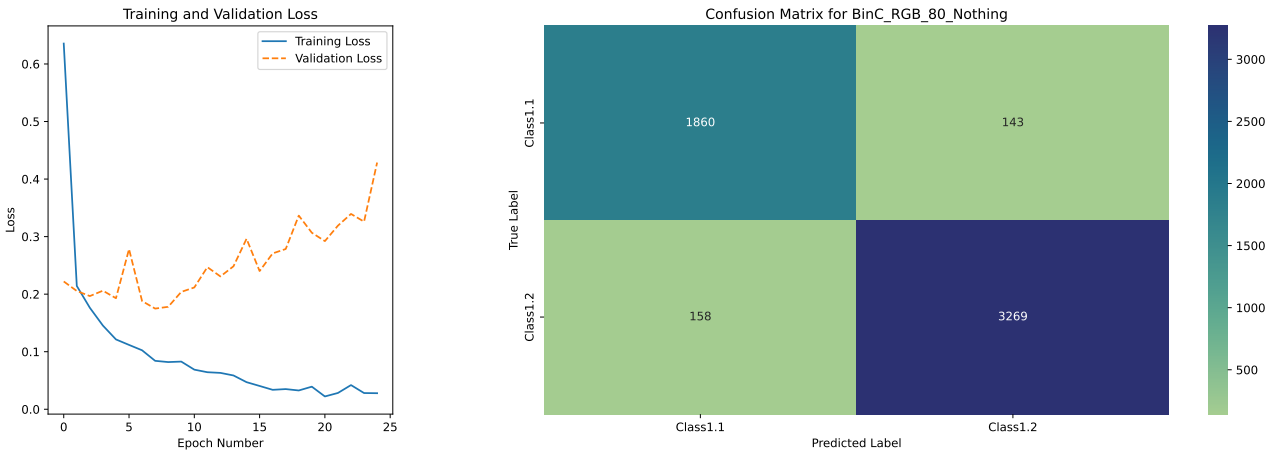
Metric	Accuracy	F1 Score	AUC	Training Time
Score	0.9339	0.9469	0.9339	20 min 47 sec

Table 5: Evaluation Metrics for Binary-80-Gray-Nothing

As seen in Fig.6, we see that the validation data set plateaus, while the training accuracy continues to increase with each epoch. This is a *clear* sign of over-fitting in the model - while it performs excellently on its training data, the unseen validation data suffers. However, its impact is somewhat *deceptive* as the validation nevertheless achieves a respectful accuracy of around 93%. This is reflected in the test data, where in Fig.7 we see that the diagonals well out-number the off-diagonal, implying that the model is doing well at classifying between the two classes, with the high Accuracy, F1, and AUC scores in Tab.5 attesting to this. This is despite the class imbalance in the data set.

4.1.2 Base with RGB

Usually, galaxy morphologies are correlated with color - spiral galaxies tend to possess younger stars and are therefore bluer than spherical galaxies with older, oranger stars. Therefore to improve the model classification, it would make sense to leave the images in RGB color. Repeating the baseline model parameters with RGB we see in Fig.8 that we get a repeat of the over-fitting data, which implies that the model requires regularization for better generalization. In general, the model performs well, as seen with the metrics of Tab.6. Compared to the metrics of the grayscale, this model performs *marginally* better, but with a cost of an additional 10 minutes of training time for 25 epochs. This *could* indicate that setting this RGB parameter may not be worth it, but as we will see, adding regularization flips this assumption.



(a) Training Loss

(b) Confusion Matrix

Figure 8: Loss and Confusion Matrix for model 80 confidence in RGB with no regularization

Metric	Accuracy	F1 Score	AUC	Training Time
Score	0.9446	0.956	0.9413	30 min 34 sec

Table 6: Evaluation Metrics for Binary-80-Gray-Nothing

4.1.3 Model with Regularization

To increase the model's already respectable performance of binary classification, we can add regularization techniques. In the following, we shall study the impact of different techniques by training gray-scale and RGB models with a Dropout layer, Augmentation Layer and one with Both and compare metrics. The confidence will remain at 80% and other configuration parameters will not change. As a side-note, since the accuracy training and confusion matrix are similar in structure for both gray and RGB for each regularization option, only the RGB metrics will be illustrated, while the metrics will be presented for both in Tab.7.

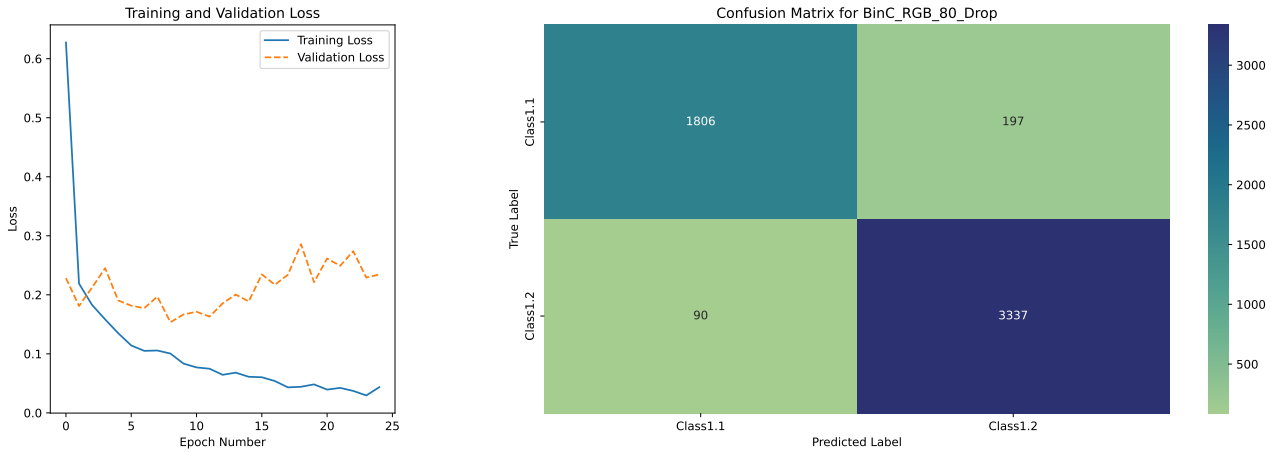


Figure 9: Loss over training and Confusion Matrix for model 80 confidence in RGB with Drop Layer

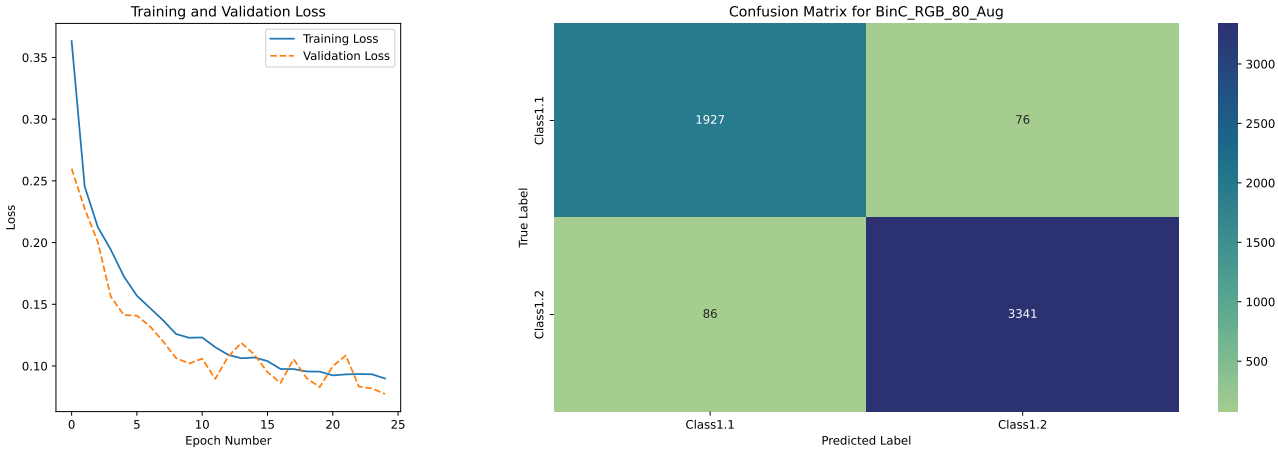


Figure 10: Loss over training and Confusion Matrix for model 80 confidence in RGB with Data Augmentation Layer

As seen in Fig.9, adding the Drop layer only *slightly* reduces over-fitting, while adding training time without providing any benefit to the performance of the model. It is therefore pointless to apply by itself. Using the data augmentation independently (Fig.10) provides a *much* better performance, especially for the RGB model, where the metrics all rise to 97%, while no over-fitting is observed. This is all at a *negligible* cost to the training time. The corresponding confusion matrix demonstrates near-perfect classification.

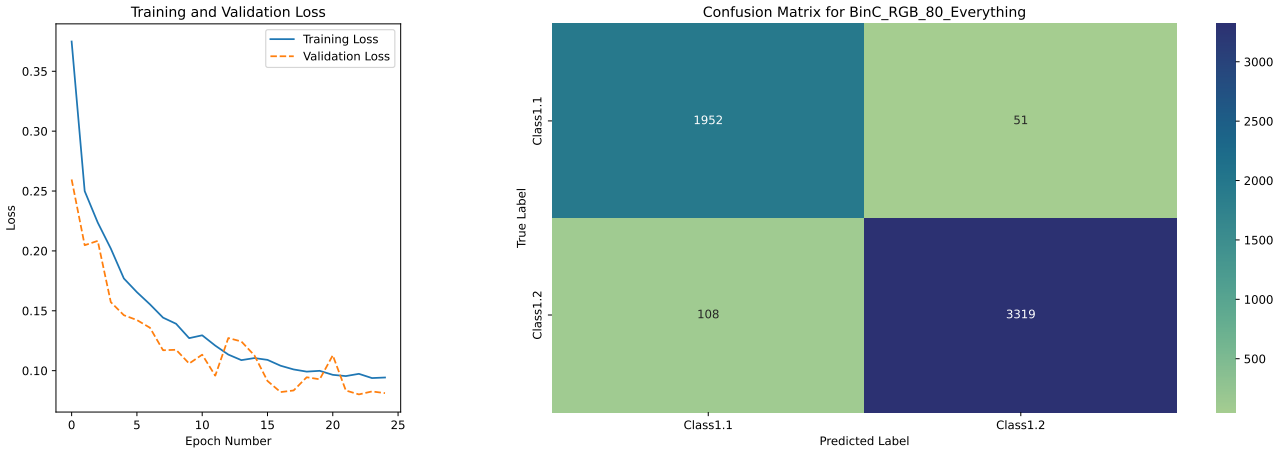


Figure 11: Training Loss and Confusion Matrix for model 80 confidence in RGB with both Drop and Augmentation Layer

Model	Accuracy	F1_Score	ROC_AUC	Training Time
Gray Drop	0.9343	0.9469	0.9363	27 min 6 sec
RGB Drop	0.9471	0.9588	0.9377	32 min 49 sec
Gray Aug	0.9569	0.9655	0.9576	26 min 7 sec
RGB Aug	0.9702	0.9763	0.9685	30 min 25 sec
Gray Everything	0.9606	0.9686	0.9602	34 min 29 sec
RGB Everything	0.9707	0.9766	0.9715	33 min 46 sec

Table 7: Evaluation Metrics for Binary-80 with all combinations of Augment Layers

Finally, applying both Augmentation and Drop appears to provide a very slight boost to performance in comparison to adding just the Augmentation layer, but does so at the cost of a non-negligible training time penalty. Looking at all the models, it is evident that the best-performing model is the RGB-enabled model with *only* an augment layer as regularization. The extra training time compared to its gray-scale counter-part appears to be worth it for the non-negligible difference in performance metrics.

4.1.4 Best-Perfoming Model at 50% Confidence

For the sake of satisfying curiosity, we will train the best performing model, i.e. CNN with Augment layer only, for 80% confidence on the data set reduced to 50% confidence. This will allow us to study how the model architecture performs with a much *larger* data set, and where the classification is objectively harder. We already should expect a reduced accuracy - however problems like over-fitting and a accuracy-f1-score discrepancy should not be present for the model to be somewhat successful. From Tab.3, we know this has around 60,000 samples, so we can also expect a much higher training time.

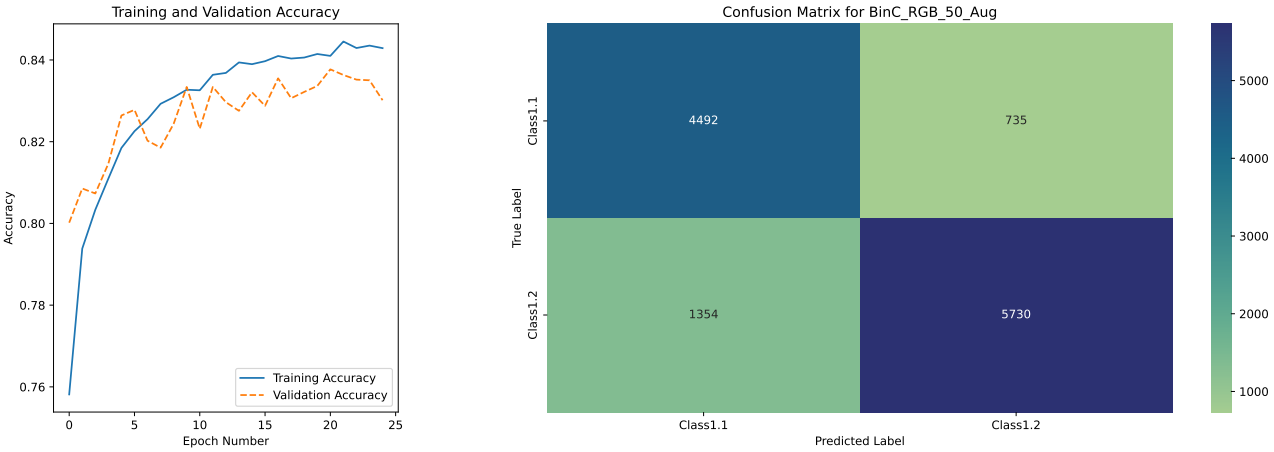


Figure 12: Training Loss and Confusion Matrix for model 50 confidence in RGB with Augmentation Layer

Metric	Accuracy	F1 Score	AUC	Training Time
Score	0.8303	0.8458	0.8341	1hr 9 min 9 sec

Table 8: Evaluation Metrics for Binary-50-RGB-Augmentation

From this visual and metric data, we can confirm what we had predicted - the accuracy has tanked, but at around 83%, the model isn't performing too bad. The close-ness of the accuracy and f1-score suggests that class number discrepancies do *not* affect classification performance, with the confusion matrix showing that it is able to classify the majority of cases in either classes somewhat well. We observe a slight over-fitting, although it is nowhere as severe as if we considered not using the augmentation layer.

4.2 Regression Task

4.2.1 Binary Classification Architecture for Regression

Before using the regression model, it is interesting nevertheless to study how the architecture used for binary classification would perform for the regression task (with modified output parameters and loss functions suited for the regression task and at 70% confidence since there are more classes). For this, we will use RGB images, with only an Augmentation layer and the CNN model structure used previously (best model for binary classification).

Metric	MSE	R^2 Score	Training Time	Accuracy	F1 Score
Score	0.01491	0.8414	37 min 57 sec	0.935	0.9346

Table 9: Evaluation Metrics for Regression-70-RGB-Aug-BinaryStructure

The accuracy and weighted F1 score are metrics used to evaluate the model's performance when the class for each galaxy is chosen through which has the highest probability based on the regression model.

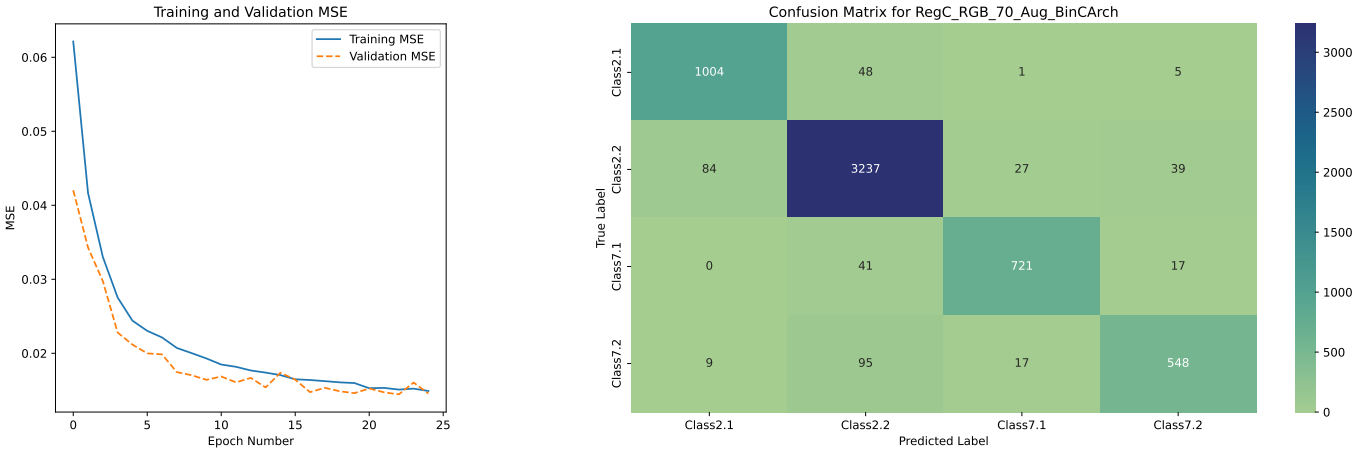


Figure 13: Training MSE and Confusion Matrix for model 70 with best Binary Classification Architecture

With a MSE of 0.0149, and an R^2 of 84%, this isn't such a bad model to begin with, and it fits the data relatively well. Furthermore, we can see in Fig.13 that there is no over-fitting, which implies it performs sufficiently well to unseen data. Turning into classification, we can see that the accuracy and the weighted F1 score across all classes is both consistent and very high - implying that the class imbalance is addressed and class discrimination is very good.

4.2.2 Regression Models at 70% Confidence

For the remainder, we will move to the modified architecture for the regression model as specified in Sec.3.2.1. Then, we will compare the metrics for the base to those with regularization layers applied to study their effects on a *different* type of problem.

Metric	MSE	R^2 Score	Training Time	Accuracy	F1 Score
RGB Nothing	0.02971	0.6808	21 min 25 sec	0.8754	0.8740
RGB Drop	0.02691	0.7099	29 min 43 sec	0.8833	0.8813
RGB Aug	0.01455	0.8476	30 min 8 sec	0.9330	0.9325
RGB Everything	0.01450	0.8485	32 min 9 sec	0.9318	0.9318

Table 10: Evaluation Metrics for Regression-70-RGB-Aug-BinaryStructure

In Tab.10, we observe that the worst-performing model is the one with no regularization applied, which is as expected, and Fig.14 displays this as clear over-fitting. However, the base is still decent at classification, where class discrimination is good based on accuracy and F1 scores at 87%. The best performing model is once again the one with only a data augmentation layer applied, with low MSE, high R^2, no over-fitting (Fig.15) and a 93% accuracy/F1 score. Perhaps surprisingly, this model performs similarly to the one with the binary classification structure, with the exception of a lesser training/computation time.

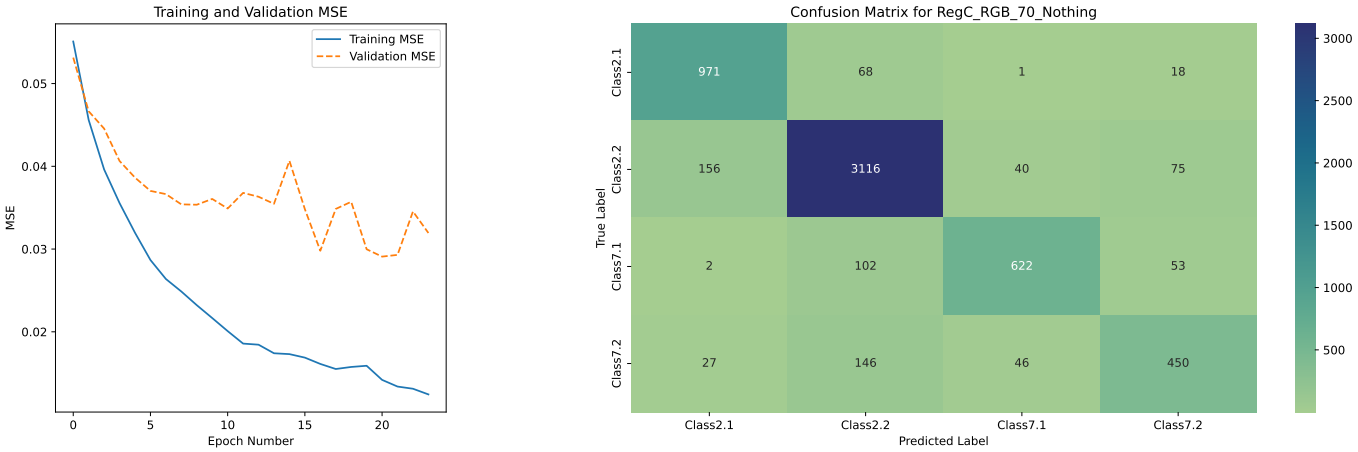


Figure 14: Training MSE and Confusion Matrix for regression model with no regularization

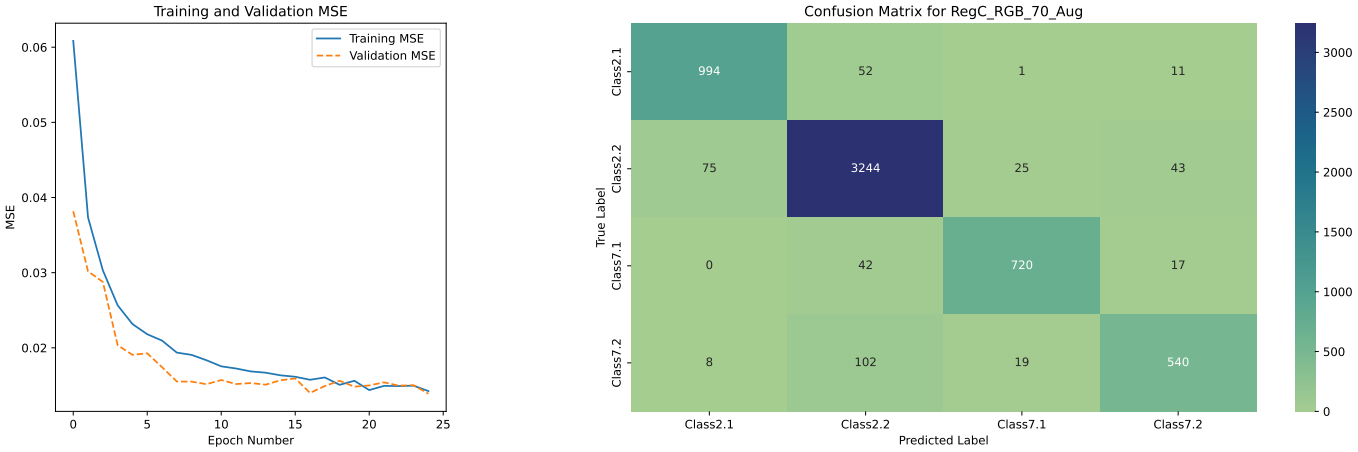


Figure 15: Training MSE and Confusion Matrix for regression model with Augmentation Layer

4.2.3 Best Model at 50% Confidence

This study will include once again reducing the confidence to 50% for the same reasons as outlined when doing the binary classification variation.

Metric	MSE	R^2 Score	Training Time	Accuracy	F1 Score
Score	0.01726	0.7796	1hr 14 min 16 sec	0.8324	0.8337

Table 11: Evaluation Metrics for Regression-50-RGB-Augmentation

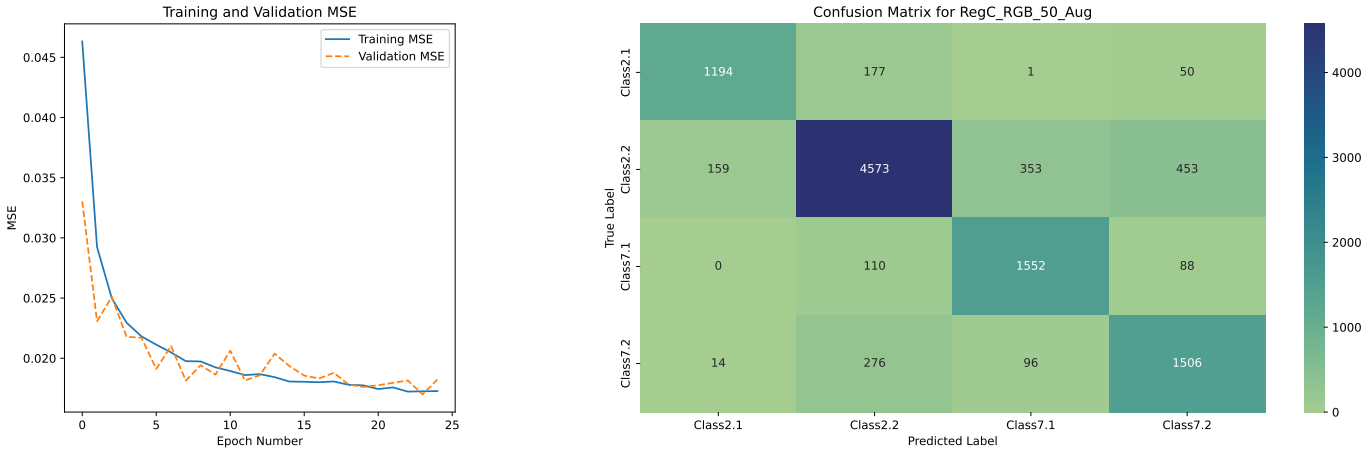


Figure 16: Training MSE and Confusion Matrix for best regression model at 50% confidence

This model performs reasonably well despite the lower confidence threshold. We see no over-fitting, a high R^2 score and accuracy/F1 score. Compared to a higher confidence models, these scores are reduced, understandably due to *more* hard-to-classify data. Class discrimination is good, with diagonal elements on the confusion matrix far outweighing incorrect classifications.

4.2.4 Best Model at 0% Confidence with all Second-level Classes

The last study was to reduce to 0% confidence, while allowing all six classes of the second level question (Class 1.3-7.3) to be in the data-set. This was to study how the best model would perform for a *heavily* imbalanced data-set with no restrictions, and including data with hard classifications.

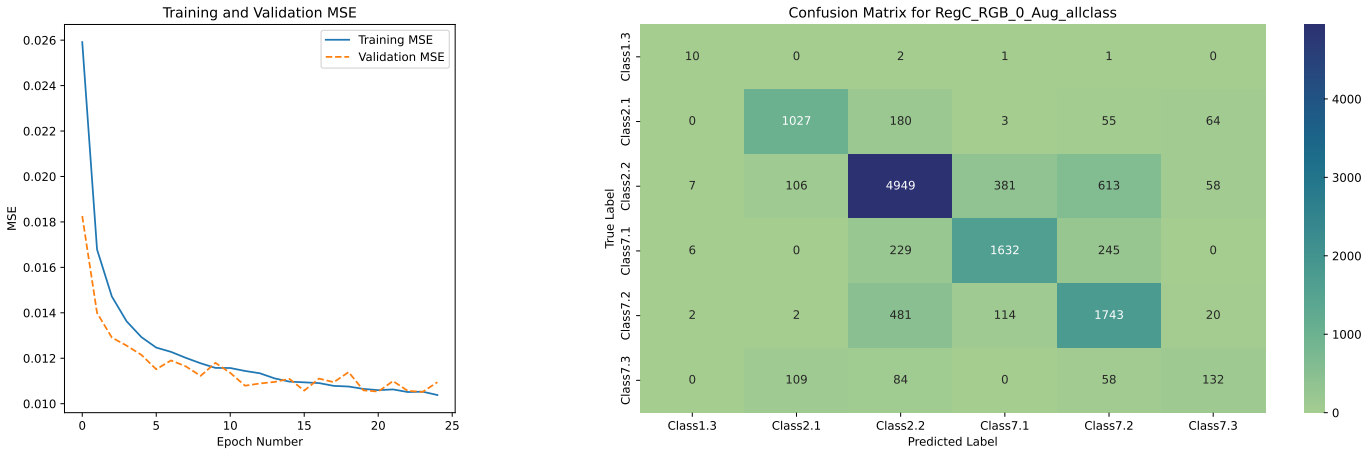


Figure 17: Training MSE and Confusion Matrix for best regression model at 0% confidence

Metric	MSE	R^2 Score	Training Time	Accuracy	F1 Score
Score	0.0107	0.6656	1hr 7 min 49 sec	0.7709	0.7711

Table 12: Evaluation Metrics for Regression-0-RGB-Augmentation

We start by noticing in Fig.17 that there is no over-fitting, suggesting that the augmentation layer has worked. However, while the MSE is *anomalously* low, even below that with higher confidence metrics, the low R^2 score of 66.6% suggests that it does not fit well to a large portion of the data, and is perhaps not the best for this task. For classification, it does decently, with 77% accuracy and a comparable F1 score. The confusion matrix displays what is obvious: Classes 1.3 and 7.3 are not well classified due to their small sample size, but the rest appear to perform well.

5 Conclusion

In conclusion, we have covered the motivations and method of Galaxy Classification using CNN models in Python. We studied variations in models, such as using a direct binary classification model or a regression model to classify. Where the model performed poorly, we studied and applied various regularization techniques such as data augmentation and dropout to increase performance. Through our studies, we have shown that high performing CNN models on a large majority of data is possible, ranging from 80% to 97% accuracy. If the model had been tweaked and trained with more complex architectures and for more epochs, perhaps this would indeed serve great use in this domain.

6 Bibliography

1. https://www.conceptdraw.com/examples/astronomical-diagram - Hubble's Tuning Fork Diagram
2. Willett, Kyle W., et al. "*Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey.*" Monthly Notices of the Royal Astronomical Society 435.4 (2013): 2835-2860. - Galaxy Zoo Project Paper
3. https://www.kaggle.com/competitions/galaxy-zoo-the-galaxy-challenge - Kaggle's Galaxy Zoo Challenge
4. https://www.upgrad.com/blog/basic-cnn-architecture/ - Information about CNN model layers
5. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 - Information about Accuracy and F1 Metrics
6. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc - Information about ROC and AUC metrics
7. https://commons.wikimedia.org/wiki/File:Roc_curve.svg - ROC curve Image
8. https://plat.ai/blog/confusion-matrix-in-machine-learning/ - Confusion matrix graphic
9. https://saylordotorg.github.io/text_introductory-statistics/s14-06-the-coefficient-of-determinati.html - R2 Metric Graphic