Galaxy Zoo Challenge : Classification of Galaxies in multiple classes

Nikhil Tiwari (38-365-375)
Depatment of Sciences
University of Geneva
Geneva, Switzerland
nikhil.tiwari@etu.unige.ch

***Abstract*—Understanding the origins and evolution of galaxies is crucial for deciphering the broader mysteries of our Universe. Galaxies exhibit a diverse range of shapes, sizes, and colors, with each morphological type providing unique insights into the physical processes that govern their formation and development. As the volume of astronomical data grows exponentially due to advancements in telescope technology, manual classification of galaxy images becomes impractical. To address this challenge, automated classification systems are essential. In this study, we participated in the Galaxy Challenge, leveraging Convolutional Neural Networks (CNNs) to automate the classification of galaxy images into predefined categories.**

Our approach involved tackling three key tasks: a primary classification problem and two subsequent regression problems, all centered around predicting galaxy morphologies. Initially, we experimented with conventional machine learning methods such as Support Vector Machines (SVMs) and decision trees. However, these approaches fell short in performance compared to deep learning techniques. We implemented a CNN for the classification task and further extended its use to regression over 37 distinct galaxy classes, achieving superior results. Additionally, we employed transfer learning using the InceptionV3 model to enhance the regression accuracy.

Our findings demonstrate that CNNs, particularly when combined with transfer learning using InceptionV3 model, significantly outperform traditional methods in classifying and understanding the complex morphologies of galaxies. This work contributes to the growing field of automated astronomical data analysis, paving the way for more efficient and accurate classification systems as data sets continue to expand.

***Index Terms*—Image Super Resolution, Convolutional Neural Networks, Generative Adversarial Networks, Residual Networks**

I. Introduction

Galaxy Zoo is a crowdsourcing project, where users are asked to describe the morphology of galaxies based on images. Galaxies in this problem set have already been classified once through the help of hundreds of thousands of volunteers, who collectively classified the shapes of these images by eye in a successful citizen science crowdsourcing project. However, this approach becomes less feasible as data sets grow to contain of hundreds of millions (or even billions) of galaxies.

We analyze the JPG images of galaxies to find automated metrics that reproduce the probability distributions derived from human classifications. For each galaxy, we determine the probability that it belongs in a particular class using different AI models including Convolutional Neural Networks and other advanced models like InceptionV3 as well.

The problem has two major parts - one where we consider it as a complete classification problem and other where we treat it as regression problem. For the classification problem, we have only 3 classes for galaxy to be classified (which aim to determine if a galaxy is (1) simply smooth and round with no disk, (2) has a disk, or (3) if the image is flawed) and in second part it has been generalised for all 37 classes.

Each galaxy's classification is the result of a specific path down a decision tree. Multiple individuals (typically 40-50) all classified the same galaxy, resulting in multiple paths along the decision tree. These multiple paths generate probabilities for each node. Volunteers begin with general questions (eg, is it smooth?) and move on to more specific ones (eg, how many spiral arms are there?). As a result, at each node or question, the total initial probability of a classification will sum to 1.0. Those initial probabilities are then weighted as follows. The values of the morphology categories in the solution file are computed as follows. For the first set of responses (smooth, features/disk, star/artifact), the values in each category are simply the likelihood of the galaxy falling in each category. These values sum to 1.0. For each subsequent question, the probabilities are first computed (these will sum to 1.0) and then multiplied by the value which led to that new set of responses. Here is a simplified example: a galaxy had 80% of users identify it as smooth, 15% as having features/disk, and 5% as a star/artifact.

Class1.1 = 0.80, Class1.2 = 0.15, Class1.3 = 0.05

For the 80% of users that identified the galaxy as "smooth", they also recorded responses for the galaxy's relative roundness. These votes were for 50% completely round, 25% in-between, and 25% cigar-shaped. The values in the solution file are thus: Class 7.1 = 0.80 * 0.50 = 0.40, Class 7.2 = 0.80 * 0.25 = 0.20, Class 7.3 = 0.80 * 0.25 = 0.20

The below flowchart explains the classification workflow

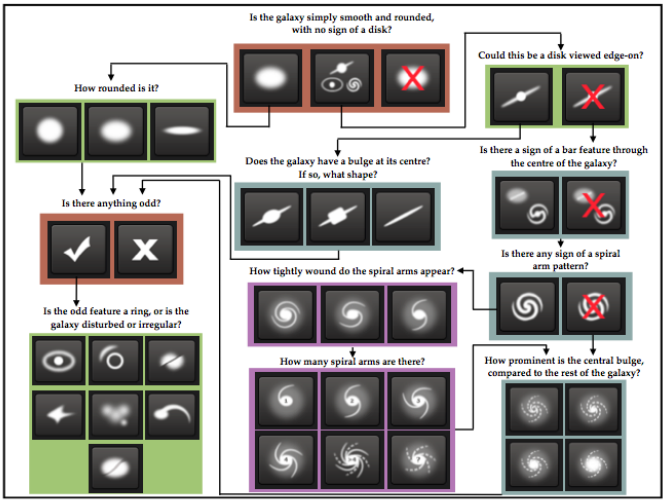

Figure 1. Flowchart of the classification tasks for GZ2, beginning at the top centre. Tasks are colour-coded by their relative depths in the decision tree. Tasks outlined in brown are asked of every galaxy. Tasks outlined in green, blue, and purple are (respectively) one, two or three steps below branching points in the decision tree. Table 2 describes the responses that correspond to the icons in this diagram.

Fig. 1. Flowchart of classification task

II. Data Provided

This dataset contains about 61,578 RGB images of galaxies together with a label vector of 37 dimensions. These labels correspond to answers given to 11 Questions. Each label is a number between 0 and 1 and shows the amount of participants who gave this answer relative to the number of participants that saw the given image. Below is statistics of the top 3 classes 1.1, 1.2, 1.3

	Class1.1	Class1.2	Class1.3
count	61578	61578	61578
mean	0.43	0.54	0.03
std	0.28	0.3	0.04
min	0	0	0
25%	0.18	0.28	0
50%	0.42	0.56	0.01
75%	0.68	0.81	0.03
max	1	1	0.94

Fig. 2. Statistics of data

To understand the data better we look at image of random 5 galaxies to get a better understanding of problem and get idea of original shapes and pre-processing.

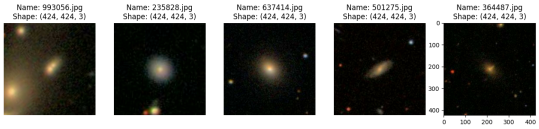

Fig. 3. Plot of 5 random galaxies and their shapes

We also generate correlation heatmap among the 37 classes, look at distributions and pairwise relationship of classes to take better pre-processing decisions.

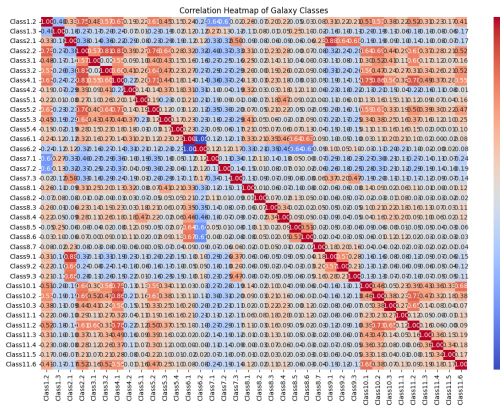

Fig. 4. Correlation heatmap among 37 classes

What: Displays a matrix of correlation coefficients between probabilistic scores of different galaxy classes. Why: Helps to identify pairs of classes that are linearly correlated. This can indicate whether certain galaxy types tend to co-occur.

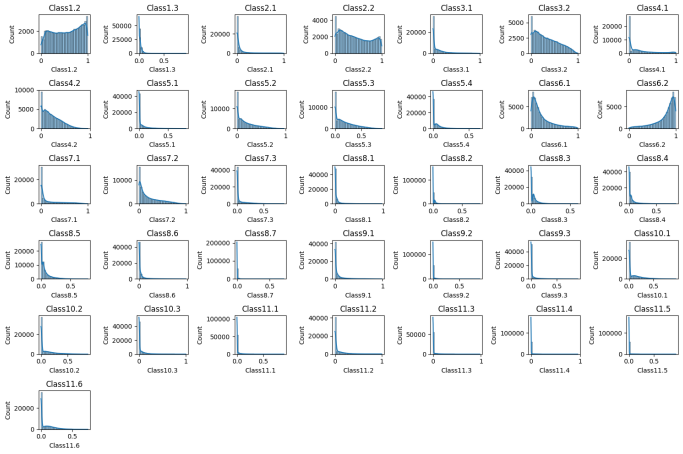

Fig. 5. Distribution of data among each class

What: Histograms with Kernel Density Estimation (KDE) show the distribution of each probabilistic score. Why: Visualizes the spread and skewness of each class's probabilistic score. This helps in understanding how frequently each probability score occurs and detecting any unusual distributions.

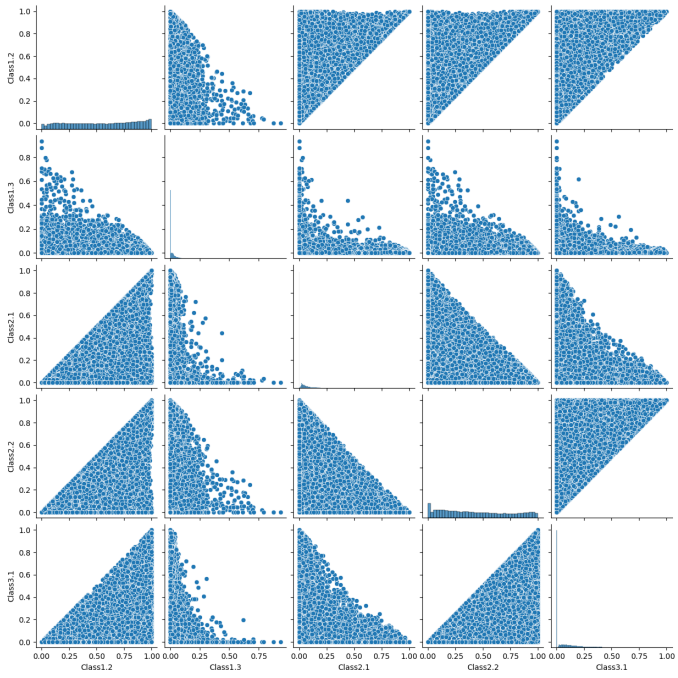

Fig. 6. Pairwise relationship between classes

What: Pairplot for a subset of classes shows the scatter relationships between pairs of classes. Why: Useful for identifying relationships, trends, and potential clusters in the data. Can highlight interesting interactions between probabilistic scores of different galaxy types.

III. Pre-processing of data

Below is the steps taken to pre-process the data for training of AI models:

A. Checking for Missing Values:

Explanation: This step is crucial to identify any data points that are absent from your dataset. Missing values can negatively impact analysis by introducing bias or skewing results. Why it's necessary: Missing data can lead to inaccurate conclusions or even render analysis impossible. Identifying missing values allows us to decide if they are a significant concern, and if so, we choose a suitable imputation method (filling in missing values) or data exclusion strategy. Possible issues: High numbers of missing values might necessitate excluding data points or entire features. The imputation method chosen can introduce bias if not carefully selected based on the data distribution and context.

We did not get any missing values in our dataset.

B. Verifying Probability Distribution and authenticity of data:

Explanation: Since data represents a probability distribution, it's essential to ensure all the values sum to 1 with appropriate weights. This verifies the data integrity and reflects a valid probability distribution. Why it's necessary: A probability distribution defines the likelihood of each possible outcome. If the values don't sum to 1, the probabilities won't be accurate, leading to misleading interpretations. Possible issues: Data collection or processing errors can introduce inconsistencies in the distribution. Result: We did not get any anomalies in probability distributions.

C. Shuffling and Splitting Data:

Explanation: Shuffling randomizes the order of data points before splitting. This helps prevent biases that might arise from the original data order. Splitting the data into training and validation sets (0.2 validation) allows to train model on the training data and evaluate its performance on the unseen validation data. Why it's necessary: Shuffling ensures the model doesn't learn biases based on the order of data presentation. Splitting helps assess the model's generalizability to unseen data. Possible issues: An imbalanced split (unequal training and validation sizes) can lead to overfitting or underfitting the model. Result: We have 48000 samples for training.

D. Part 1: Label Transformation (Classification):

Explanation: This step transforms multi-class probability labels into binary labels for a classification task. We focus on the first three probabilities, classifying data points with at least one of those probabilities exceeding 0.8 as positive (represented by 1) and the rest as negative (represented by 0). This simplifies the problem to a binary classification task. Why it's necessary: If analysis requires binary classification, this transformation is necessary to convert the multi-class data into a suitable format for the model. Possible issues: Choosing an inappropriate threshold (0.8 in this case) might exclude relevant data or introduce bias. Consider exploring different thresholds or alternative classification approaches if needed. Result: We converted regression task as classification problem for part 1.

E. Part 2: Dataloaders for Labels:

Explanation: Dataloaders are a mechanism for efficiently loading and managing the labeled data during training. They handle batching, shuffling (within each epoch), and prefetching data for faster training.

F. Image Pre-processing:

(a) Normalization:

Explanation: Normalization scales image pixel values to a specific range (often between 0 and 1). In our case, dividing by 255 achieves this for each pixel value. This helps improve the training process by ensuring all features are within a similar range and prevents features with larger scales from dominating the model. We can also try finding mean and std from image sample and use it for normalisation but it has been avoided to maintain simplicity.

(b) Image Cropping and Downsampling:

Explanation:

Cropping: Selecting a specific region of interest (256x256 pixels in our case) from the original image. This can be

helpful if the relevant information is contained within a specific area, or if the background introduces noise. Downsampling: Reducing the image resolution to a smaller size (64x64 pixels). This reduces computational cost during training and can potentially improve model generalization by focusing on essential features. Why it's necessary: Normalization helps training algorithms converge faster and avoid numerical instability. Cropping and downsampling improve computational efficiency and potentially reduce overfitting by focusing on essential features.

(c) Image Augmentation:

Explanation: Image augmentation involves artificially creating variations of your existing images (e.g., translations, rotations, flips). This helps the model learn features that are robust to small variations in the data and improve generalization to unseen images. It is not used for training bigger models but for small models only due to increasing complexity

Why it's necessary: Augmentation can be particularly helpful when the dataset is limited, as it effectively increases the

IV. Models Used

A. Convolutional Neural Networks

Convolutional neural networks (CNN) date back decades and deep CNNs have recently shown an explosive popularity partially due to its success in image classification. They have also been successfully applied to other computer vision fields, such as object detection, face recognition, and pedestrian detection. Several factors are of central importance in this progress: (i) the efficient training implementation on modern powerful GPUs, (ii) the proposal of the Rectified Linear Unit (ReLU) which makes convergence much faster while still presents good quality, and (iii) the easy access to an abundance of data (like ImageNet) for training larger models.

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret image or visual data.

1. Architecture of CNN

A CNN consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

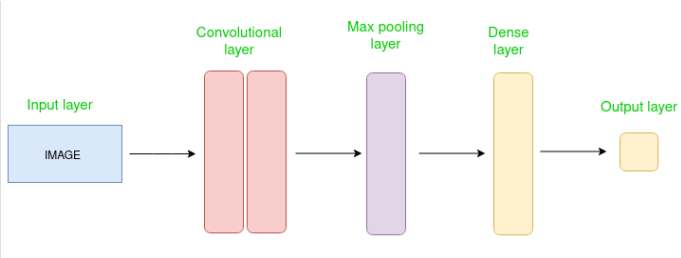

Fig. 7. Basic Architecture of CNN

The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

How Convolutional Layers Work

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e., the channel as images generally have red, green, and blue channels).

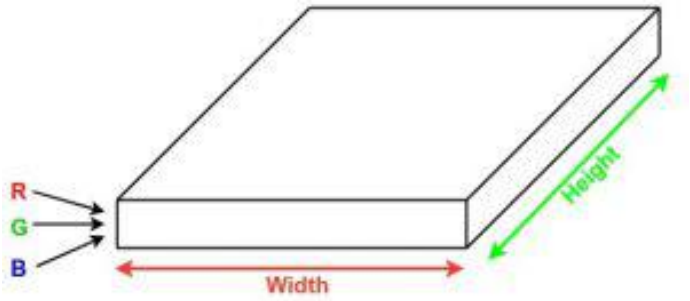

Fig. 8.

Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called Convolution. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

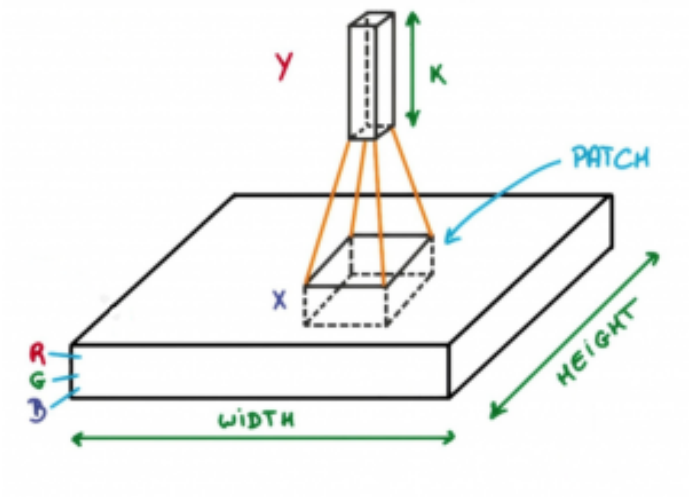

Fig. 9.

Above was logical explanation. Below is little mathematical explanation.

Mathematics Involved in Convolution Process

Convolution layers consist of a set of learnable filters (or kernels) having a small spatial extent (along height and width) but extending through the full depth of the input volume. For example, for an RGB image, the filter would have a size of 3 (for the three color channels).

Pooling Layer

The pooling layer downsamples the image to reduce computation. There are two types of pooling layers: Max Pooling and Average Pooling.

Max Pooling

Max Pooling takes the maximum value across each patch of the feature map.

Average Pooling

Average Pooling takes the average value across each patch of the feature map.

Fully Connected Layer

The fully connected layer makes the final prediction. It is a regular neural network layer where every input is connected to every output.

B. InceptionV3 Model

The InceptionV3 model is a variant of the Inception architecture, which is a type of Convolutional Neural Network (CNN) designed for image classification tasks. It was published in 2015 by Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi from Google Inc.

The InceptionV3 model is a deep neural network that consists of several building blocks including.

Stem Block: The stem block is the initial block of the network, which consists of a convolutional layer, a max pooling layer, and another convolutional layer.

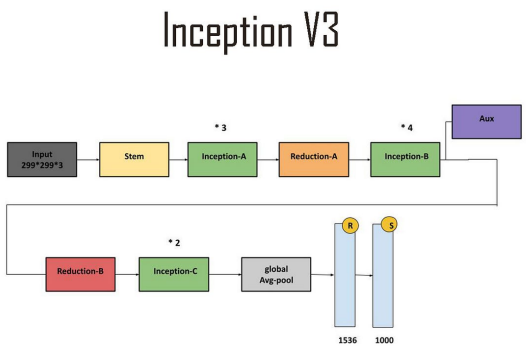

Fig. 10. InceptionV3 model architecture

Inception-A Block: The Inception-A block is a building block that consists of four parallel branches: a convolutional layer with a filter size of 1x1, a convolutional layer with a filter size of 3x3, a convolutional layer with a filter size of 5x5, and a max pooling layer. The outputs of these branches are concatenated along the channel axis.

Inception-B Block: The Inception-B block is similar to the Inception-A block, but with different filter sizes and numbers of filters.

Inception-C Block: The Inception-C block is similar to the Inception-A block, but with even larger filter sizes and more filters.

Reduction-A Block: The Reduction-A block is a building block that reduces the spatial dimensions of the feature maps. It consists of a convolutional layer with a stride of 2, followed by a max pooling layer.

Reduction-B Block: The Reduction-B block is similar to the Reduction-A block, but with different filter sizes and numbers of filters.

Auxiliary Classifier Block: The auxiliary classifier block is an additional classifier that is used to improve the training process. It consists of a convolutional layer, a max pooling layer, and a fully connected layer.

The InceptionV3 model uses a technique called batch normalization, which normalizes the input data for each layer to have a mean of 0 and a standard deviation of 1. This technique helps to improve the stability and speed of the training process.

C. Transfer Learning

Transfer learning is a technique in machine learning that allows a model to leverage pre-trained weights and fine-tune them on a new dataset. This technique is particularly useful when the new dataset is small or similar to the original dataset used to pre-train the model.

The idea behind transfer learning is that a model trained on a large dataset can learn general features that are applicable to other datasets. For example, a model trained on a dataset of images of animals can learn features such as edges, textures, and shapes that are applicable to other datasets of images.

There are several benefits to using transfer learning:

Faster training: Transfer learning allows the model to start with a set of pre-trained weights, which can reduce the training time and improve the performance of the model. Improved performance: Transfer learning can improve the performance of the model by leveraging the knowledge learned from the pre-trained dataset. Less data required: Transfer learning can be used with smaller datasets, which can be particularly useful when the dataset is limited.

The process of transfer learning involves the following steps:

Pre-training: The model is pre-trained on a large dataset, such as ImageNet. Fine-tuning: The pre-trained model is fine-tuned on a new dataset, such as a dataset of images of animals. Freezing: Some of the layers of the pre-trained model are frozen, while others are fine-tuned on the new dataset. Transfer learning is widely used in deep learning, particularly in computer vision tasks such as image classification, object detection, and segmentation.

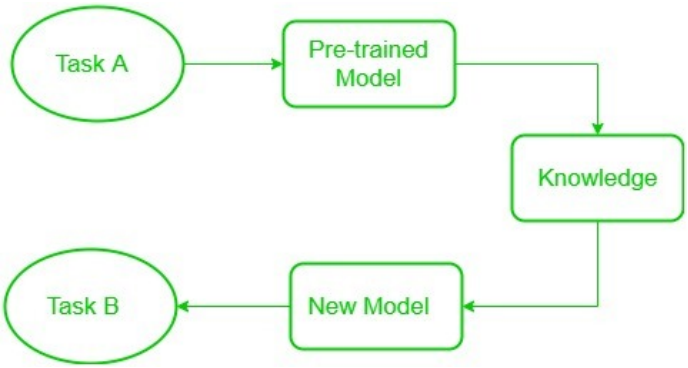

Fig. 11. Transfer Learning concept

Transfer Learning has become immensely popular because it considerably reduces training time, and requires a lot less data to train on to increase performance. In our task we use transfer learning with InceptionV3 model as well.

V. Part 1: Classification Problem

VI. Whole Classification Task using Regression

We used CNN model first for our training whose details are as follows:

A. Model Overview

The CNN model is a type of neural network designed to process images. It consists of multiple layers that work together to extract features from images and make predictions. The model takes in images of size 64x64x3 (height, width, and color channels) and outputs a probability distribution over 37 classes.

B. Convolutional Layers

The convolutional layers are responsible for extracting features from the input images. They consist of a series of filters that slide over the image, detecting patterns and features.

Layer 1: Conv2D (64 filters, 3x3 kernel)

The first convolutional layer has 64 filters, each with a size of 3x3. These filters scan the input image, detecting small features such as edges and lines.

Layer 2: Conv2D (64 filters, 3x3 kernel)

The second convolutional layer has 64 filters, each with a size of 3x3. These filters build upon the features detected in the previous layer, extracting more complex features such as shapes and textures.

Activation Function: ReLU

The ReLU (Rectified Linear Unit) activation function is used to introduce non-linearity into the model. It takes the output of the convolutional layer and applies a threshold, setting all negative values to 0 and all positive values to the same value.

C. Max Pooling Layer

The max pooling layer reduces the spatial dimensions of the feature maps, retaining only the most important features. This helps to reduce overfitting and improve the model's ability to generalize.

Layer 3: Conv2D (128 filters, 3x3 kernel)

The third convolutional layer has 128 filters, each with a size of 3x3. These filters extract more complex features from the input image, building upon the features detected in the previous layers.

Layer 4: Conv2D (128 filters, 3x3 kernel)

The fourth convolutional layer has 128 filters, each with a size of 3x3. These filters further refine the features detected in the previous layer.

D. Global Max Pooling Layer

The global max pooling layer reduces the spatial dimensions of the feature maps to a single value, retaining only the most important features.

E. Dense Layers

The dense layers are responsible for making predictions based on the features extracted by the convolutional layers.

Layer 5: Dense (128 units)

The first dense layer has 128 units, which take the output of the convolutional layers and make predictions.

Activation Function: ReLU

The ReLU activation function is used to introduce non-linearity into the model.

Dropout Layer

The dropout layer randomly sets a fraction of the output units to 0 during training, helping to prevent overfitting.

F. Benefits of Dropout

Prevents overfitting by reducing the capacity of the model Improves generalization by forcing the model to learn multiple representations of the data Reduces the risk of memorization by making the model more robust to noise and variations in the data Layer 6: Dense (128 units)

The second dense layer has 128 units, which further refine the predictions made by the previous layer.

Activation Function: ReLU

The ReLU activation function is used to introduce non-linearity into the model.

G. Dropout Layer

The dropout layer randomly sets a fraction of the output units to 0 during training, helping to prevent overfitting.

H. Output Layer

The output layer has 37 units, which correspond to the 37 classes in the dataset. The sigmoid activation function is used to output a probability distribution over the classes.

```
Model Summary:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 64)        1792

 conv2d_1 (Conv2D)           (None, 60, 60, 64)        36928

 activation (Activation)     (None, 60, 60, 64)        0

 max_pooling2d (MaxPooling2  (None, 30, 30, 64)        0
 D)

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 conv2d_3 (Conv2D)           (None, 26, 26, 128)       147584

 activation_1 (Activation)   (None, 26, 26, 128)       0

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 128)       0
 g2D)

 conv2d_4 (Conv2D)           (None, 11, 11, 128)       147584

 conv2d_5 (Conv2D)           (None, 9, 9, 128)         147584

 activation_2 (Activation)   (None, 9, 9, 128)         0

 global_max_pooling2d (Glob  (None, 128)               0
 alMaxPooling2D)

 dense (Dense)               (None, 128)               16512

 activation_3 (Activation)   (None, 128)               0

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 activation_4 (Activation)   (None, 128)               0

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 37)                4773

 activation_5 (Activation)   (None, 37)                0

=================================================================
Total params: 593125 (2.26 MB)
Trainable params: 593125 (2.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig. 12. Model Used

I. Benefits of Each Layer

Convolutional layers: Extract features from images, detecting patterns and shapes Max pooling layers: Reduce spatial dimensions, retaining only the most important features Dense layers: Make predictions based on the features extracted by the convolutional layers Dropout layers: Prevent overfitting, improve generalization, and reduce memorization Activation functions: Introduce non-linearity into the model, allowing it to learn more complex relationships between inputs and outputs

J. Results

We got the following results after training given CNN model for 9 epochs with LR = 0.002:

Accuracy: 0.7501
Validation Accuracy: 0.7575
Loss: 0.2464 Validation Loss: 0.2463

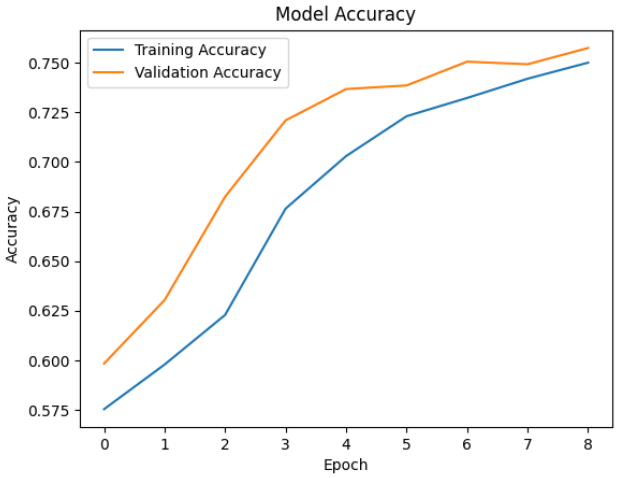

Fig. 13. Accuracy of training model with epochs

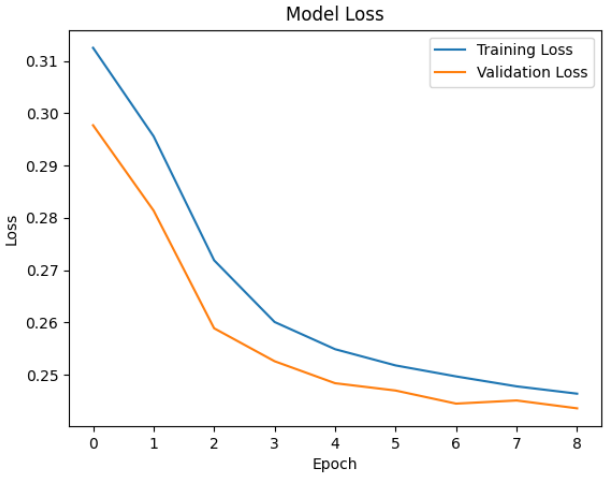

Fig. 14. Loss in model with epochs

As expected, loss decreased with number of epochs and accuracy increased.

VII. Using InceptionV3 model

A. Model Overview

The InceptionV3 transfer learning model is a type of neural network that leverages the pre-trained InceptionV3 model, which has been trained on the ImageNet dataset, to make predictions for a new regression task. The model consists of the pre-trained InceptionV3 model, which is responsible for extracting features from the input images, and a custom top layer that makes predictions based on those features.

B. Pre-trained InceptionV3 Model

The pre-trained InceptionV3 model is a type of convolutional neural network (CNN) that has been trained on the ImageNet dataset, which contains over 1 million images from 1000 different classes. The model has been shown to be highly effective at extracting features from images, and has been used as a starting point for many computer vision tasks.

C. Freezing the Layers in the Base Model

In order to prevent the pre-trained InceptionV3 model from being updated during training, we freeze the layers in the base model. This means that the weights in the base model will not be updated during training, and the model will only learn to make predictions based on the features extracted by the pre-trained model.

D. Custom Top Layers

The custom top layers are responsible for making predictions based on the features extracted by the pre-trained InceptionV3 model.

E. Global Average Pooling Layer

The global average pooling layer reduces the spatial dimensions of the feature maps to a single value, retaining only the most important features.

F. Dense Layers

The dense layers are responsible for making predictions based on the features extracted by the global average pooling layer.

Layer 1: Dense (128 units)

The first dense layer has 128 units, which take the output of the global average pooling layer and make predictions.

Activation Function: ReLU

The ReLU (Rectified Linear Unit) activation function is used to introduce non-linearity into the model. It takes the output of the dense layer and applies a threshold, setting all negative values to 0 and all positive values to the same value.

G. Output Layer

The output layer has 1 unit, which corresponds to the single output node for regression. The linear activation function is used to output a continuous value.

Compiling the Model

The model is compiled with the Adam optimizer and the mean squared error loss function. The mean squared error loss function is commonly used for regression tasks, as it measures the average squared difference between the predicted and actual values.

```
=================================================================
Total params: 22065185 (84.17 MB)
Trainable params: 262401 (1.00 MB)
Non-trainable params: 21802784 (83.17 MB)
```

Fig. 15. Inception V3 model details

H. Benefits of Transfer Learning

Transfer learning is a technique that leverages pre-trained models to make predictions for new tasks. The benefits of transfer learning include:

Improved performance: Pre-trained models have already learned to extract features from large datasets, and can be used as a starting point for new tasks. Reduced training time: Pre-trained models can be fine-tuned for new tasks, reducing the amount of time required to train a model from scratch. Improved generalization: Pre-trained models have already learned to extract features from large datasets, and can be used to improve the generalization of a model.

I. Results

We trained for 19 epochs with LR 0.001 and we got below results (stating the lowest values)

Loss: 0.35 Validation Loss: 0.59 RMSE: 0.36 Validation RMSE: 0.37 Accuracy: 0.5093 Validation Accuracy: 0.4095

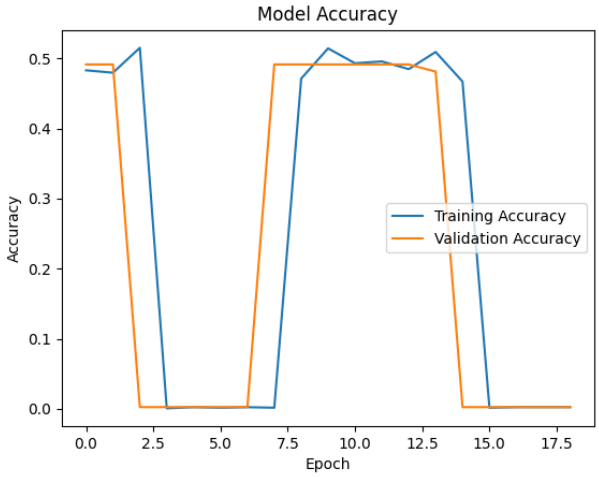

Fig. 16. Accuracy vs epochs

Accuracy increases upto some point and then falls back which shows issues in learning rate or hyperparameters of model since both validation and training accuracy fall down simultaneously.

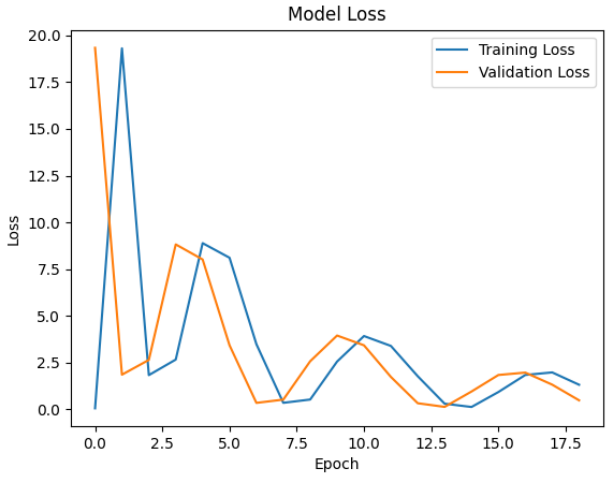

Fig. 17. Loss vs epochs

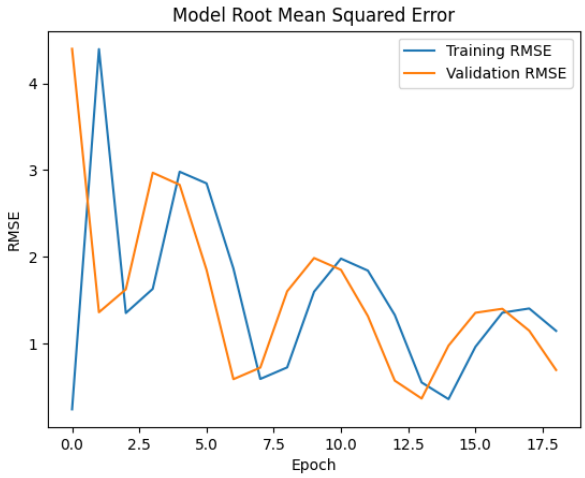

Fig. 18. RMSE vs epochs

VIII. Future Work and Scopes

We can have a combination of models for better results with early stopping callback. Using transfer learning from advanced models like ResNet, AlexNet, XceptionNet and combining their results will produce more accuracy.

References

1) CNN
2) Inception V3
3) Galaxy Zoo challenge