Morphological Classification of Galaxies with Machine Learning

PHYS3004 Project Report

by

Malachy Doherty*
Supervised by Dr Adam Moss

Submitted May 2024

*Student ID: 20274950, Email: ppymd5@nottingham.ac.uk

Abstract

This project aims to use Machine Learning techniques such as transfer learning of deep Convolutional Neural Networks, and Vision Transformer architectures to explore the practicality of galaxy morphological classification using crowdsourced labels from the Galaxy Zoo as the ground truth to train a computer vision model. I discovered that the models performed to the same ability as the volunteers when identifying morphological features of galaxies, and even replicated the confusion between features exhibited in the crowdsourced labels.

Table of Contents

Main Content

1 Introduction

The study of galaxy morphologies is significant for our understanding of their formation, history, and evolution, and has important implications for cosmology. To date, sky surveys such as the Sloan Digital Sky Survey (SDSS) and The Dark Energy Survey (DES) have discovered over 400 million galaxies [1][2]. Over the past decade, projects such as Galaxy Zoo have successfully utilized crowdsourcing for the morphological classification of galaxies on sky surveys such as SDSS [3].

However, future projects such as The Legacy Survey of Space and Time (LSST) [4] will produce over 30 terabytes of image data per night, for a total database size of 150 petabytes. That is an estimated 10^{10} galaxies at $z > 6$ with morphological and photometric measurements for 4×10^9 galaxies [5], far more than what can be classified by crowdsourcing.

Computer Vision, a subset of Machine Learning, is capable of solving image classification problems on large datasets, so hence is of interest with regard to morphological classification of galaxies. Creating a model capable of such a task can done using the crowdsourced classifications of galaxies from the Galaxy Zoo, which can be used as an approximation to the true morphological classifications of the galaxies. Along with the images of the galaxies, these classifications can be used to train a Computer Vision model that distinguishes between different morphological features of galaxies.

There are many different approaches to Computer Vision, from Convolutional Neural Networks, which over the past 30 years have been shown to be well suited to the problem, to the more recent use of the Transformer. Despite the vastly different approaches, the shared foundation behind them both is the Neural Network.

2 The Neural Network

2.1 The Artificial Neuron

The fundamental building block of a neural network is the neuron. A neuron within a network takes an input vector $\vec{x} = (1, x)^T$ and maps it to an output scalar $a(x)$. The neuron seeks to imitate the strength of connections between biological neurons by weighting the input $\vec{x}$ by taking its dot product with an equal length vector of weights $\vec{w}$.

$$z = \vec{x}^T \cdot \vec{w} + b \tag{2.1}$$

An additional scalar b, the bias, may also be added. These weights and the bias are learnable parameters by the neuron.

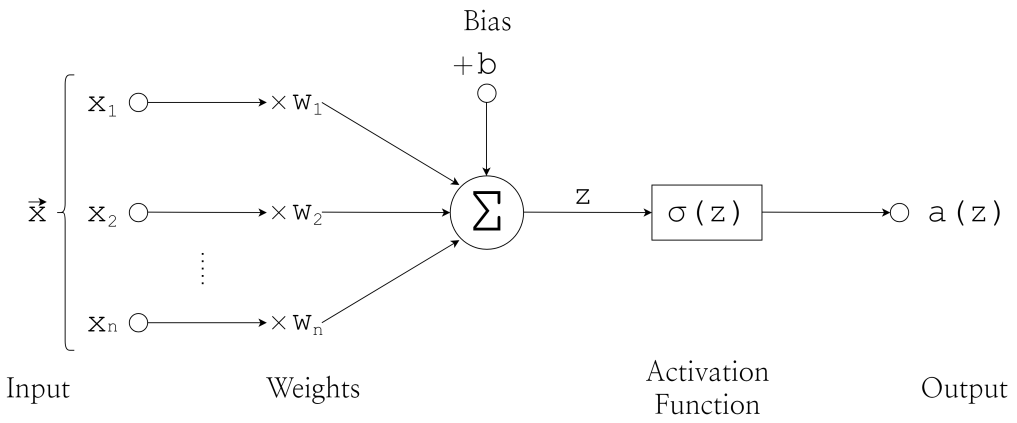


Figure 1: Visualization of an artificial neuron

z is passed through an activation function $a = \sigma(z)$ which maps the inputs to a scalar output $a(x)$, as shown by (Figure 1). The additional bias term allows for the activation function to be shifted towards positive or negative values.

2.2 The Perceptron

One of the most common implementations of the artificial neuron is the Perceptron, a binary classifier, which has the Heaviside step function as its activation function:

$$H(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{2.2}$$

This allows the Perceptron to map its inputs to two possible categories along a planar decision boundary.

2.3 The Multi Layered Perceptron

The problem of classifying non-linearly separable data can be solved by linking together multiple layers of differing numbers of Perceptrons (as shown by Figure 2), again imitating biological neurons. This forms the basis of the artificial neural network, a Multi-Layered Perceptron (MLP). The first layer is the input layer which passes the data into the network, and the last layer is the output layer which provides the final classification of the input by the network. The layers in between are the hidden layers, their activation functions being replaced with the ReLU function:

$$ReLU = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{2.3}$$

Otherwise, the network would remain a linear function of its input. This allows the network to learn abstract and complex relationships between the non-linearly separable input data to achieve a desired output, such as a classification. The exact number of layers, the number of neurons, and their activation functions are known as the architecture of the network.

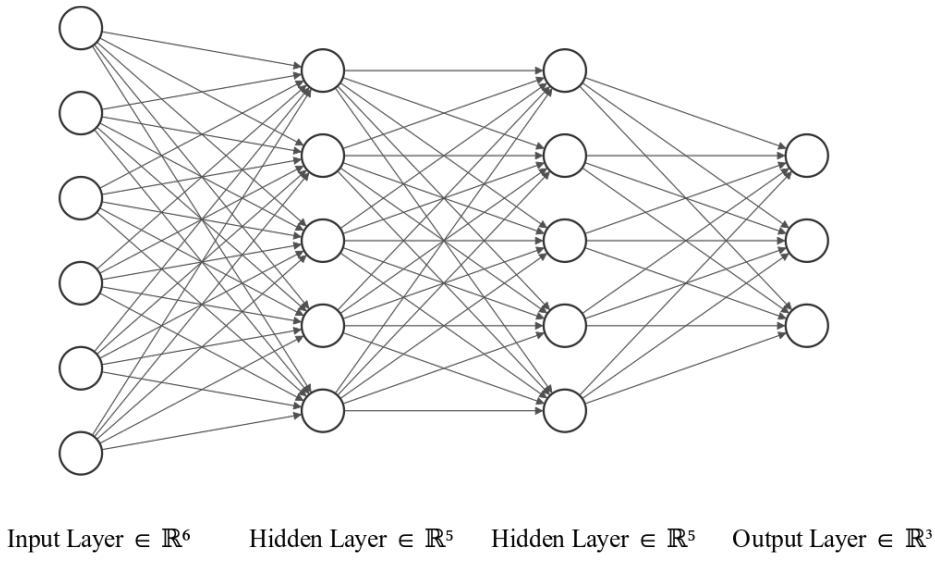


Figure 2: Architecture of a Multi-Layer Perceptron (MLP)

2.4 Training a Neural Network

Neural Networks that are focused on image classification are generally trained using datasets of images with associated labels. This is an example of supervised learning. During training the network receives example images $\vec{x}$ and makes a prediction $\hat{y}$ about the possible label(s) the image could have. The predictions are then compared to the true labels $\vec{y}$. The error between the vectors $\hat{y}$ and $\vec{y}$ is known as the loss. A commonly used loss function for multi-label classification of images is binary cross-entropy, BCE:

$$L_{BCE} = \frac{1}{N}\sum_{i=1}^{N} -(y_i \; log\,(\hat{y}_i) + (1 - y_i)\log\,(1 - \hat{y}_i)) \tag{2.4}$$

2.4.1 Gradient Descent: The Minimisation of Loss

We wish to find the network parameters; weights and biases, that minimise this loss. For a network with parameters θ the loss function w.r.t to these parameters is $L(\theta)$. The loss function around a point θ_t in parameter space can be approximated by a first-order Taylor Expansion:

$$L(\theta) \approx L(\theta_t) + (\theta - \theta_t)^T \cdot \vec{\nabla} L(\theta_t) \tag{2.5}$$

We want our new parameters θ_{t+1} to minimise $L(\theta)$ so we must move in the direction opposite to the gradient:

$$\theta_{t+1} = \theta_t - \alpha \vec{\nabla} L(\theta_t) \tag{2.6}$$

Where α is the learning rate, controlling the step size taken in parameter space. Since the objective of gradient descent is to minimise $L(\theta)$ we require, $L(\theta_{t+1}) < L(\theta_t)$.

$$L(\theta_{t+1}) \approx L(\theta_t) + \left(\theta_t + \alpha \vec{\nabla} L(\theta_t) - \theta_t\right)^T \vec{\nabla} L(\theta_t) \tag{2.7}$$

$$L(\theta_{t+1}) \approx L(\theta_t) - \alpha ||\vec{\nabla} L(\theta_t)||^2 \tag{2.8}$$

Where $||\vec{\nabla} L(\theta_t)||^2$ is the squared norm of the gradient. Since α is a positive constant and $||\vec{\nabla} L(\theta_t)||^2 \geq 0$ for all θ_t, if α is sufficiently small enough then $L(\theta_{t+1}) < L(\theta_t)$.

$$\therefore \theta_{t+1} = \theta_t - \alpha \vec{\nabla} L(\theta_t) \tag{2.9}$$

Therefore, the parameters of the network that minimise the loss can be iteratively found if the gradient of the loss function w.r.t the network parameters is known for each iteration, $\vec{\nabla} L(\theta_t)$ [6].

2.4.2 Gradient Descent: Finding the Gradient

The gradient can be calculated by backpropagating the errors between the predictions and true labels through the network as follows [7]:

The input $\vec{x}$ is passed forward through the network, causing each layer to output a vector of values $\vec{z}^{(l)}$. The output of each layer l with weights $\vec{w}^{(l)}$, biases $\vec{b}^{(l)}$ and activation function σ is:

$$\vec{z}^{(l)} = \vec{a}^{(l-1)} \cdot \vec{w}^{(l)} + \vec{b}^{(l)} \tag{2.10}$$

$$\vec{a}^{(l)} = \sigma(\vec{z}^{(l)}) \tag{2.11}$$

$\vec{a}^{(l)}$ is the output of the layer passed through its activation function σ. The error between $\vec{y}$ and $\widehat{\vec{y}}$ is computed using the loss function. This error is then backpropagated through the network, starting at the output of the final layer.

$$\frac{\partial L}{\partial \vec{a}^{(l)}} = \frac{\partial L}{\partial \vec{z}^{(l)}} \frac{\partial \vec{z}^{(l)}}{\partial \vec{a}^{(l)}} \tag{2.12}$$

$$\therefore \frac{\partial L}{\partial \vec{z}^{(l)}} = \frac{\partial L}{\partial \vec{a}^{(l)}} \frac{\partial \vec{a}^{(l)}}{\partial \vec{z}^{(l)}} \tag{2.13}$$

We now know how a change to the input of a layer will affect the error in its output. However, this input is also a function of the output of the previous layer, and the current layer parameters, so the error in the output will be affected by changing these parameters by:

$$\frac{\partial L}{\partial \theta_t^{(l)}} = \frac{\partial L}{\partial \vec{z}^{(l)}} \frac{\partial \vec{z}^{(l)}}{\partial \theta_t^{(l)}} \tag{2.14}$$

$$\therefore \frac{\partial L}{\partial \theta_t^{(l)}} = \left(\frac{\partial L}{\partial \vec{a}^{(l)}} \frac{\partial \vec{a}^{(l)}}{\partial \vec{z}^{(l)}} \right) \frac{\partial \vec{z}^{(l)}}{\partial \theta_t^{(l)}} \tag{2.15}$$

We then recursively apply the chain rule, computing the gradient of the loss w.r.t the parameters of each layer and accumulate the gradients to obtain the total gradient.

$$\vec{\nabla} L \left(\theta_t^{(l)} \right) = \frac{\partial L}{\partial \vec{a}^{(l)}} \frac{\partial \vec{a}^{(l)}}{\partial \vec{z}^{(l)}} \frac{\partial \vec{z}^{(l)}}{\partial \theta_t^{(l)}} \tag{2.16}$$

$$\therefore \vec{\nabla} L \left(\theta_t \right) = \sum_{l=1}^{L} \vec{\nabla} L \left(\theta_t^{(l)} \right) \tag{2.17}$$

The gradient calculated by backpropagation can then be used to perform gradient descent using Equation (2.9) by the optimisation algorithm of the network, giving us our new network parameters. There are many network optimisation algorithms, but all follow this basic premise of gradient descent through the backpropagation of errors.

2.4.3 Gradient Descent: Consequences and Shortcomings

For a sufficiently small learning rate and enough iterations, gradient descent is guaranteed to converge to a solution, however, it may not be the optimal solution. The network optimiser may converge to a local minima or saddle point in parameter space, where once inside, it cannot escape due to the local gradient being close to zero. This is the vanishing gradients problem. Conversely, if the learning rate is too large, the optimiser may overshoot the global minima and may never reach an optimal solution. This is the exploding gradients problem.

These problems can be avoided by using activation functions that preserve the gradients, initialising the weights of the network to small random values, adjusting the learning rate, using a different network optimisation algorithm, and using normalisation techniques such

as batch normalisation; a method that adjusts the values in each layer of a neural network during training to make learning more stable and efficient.

Another common problem faced by neural networks is overfitting and underfitting. Overfitting is when the network learns correlations in the training data such as noise that does not exist in real-world data causing it to fail to generalise. Underfitting is when the network cannot learn meaningful information from the training data resulting in it being unable to predict what classes data lies within.

Overfitting can be mitigated with regularisation techniques such as dropout, which randomly deactivates neurons during training, and data augmentation, which generates additional training examples for better generalisability. To combat underfitting, one can adjust the network's architecture or reduce the complexity of the problem.

2.5 MLPs and Computer Vision

Despite being effective for classification and regression, MLPs alone are poorly suited to dealing directly with images due to their fully connected structure. With one perceptron for each pixel of the input image, the number of parameters quickly becomes unwieldy leading to computational inefficiency.

3 Convolutional Neural Networks

The limitations of MLPs in image classification spurred the development of Convolutional Neural Networks, CNNs. Emerging in the late 1990s, CNNs were inspired by the visual cortex's structure in animals, where neurons respond to overlapping regions of the visual field at differing spatial scales [8]. With the introduction of LeNet-5 by Yann LeCun et al. [9] in 1998, which successfully recognized handwritten digits, CNNs gained traction. The breakthrough for CNNs came in 2012 when AlexNet, developed by Alex Krizhevsky et al. [10], won the ImageNet Large Scale Visual Recognition Challenge, significantly outperforming traditional methods.

3.1 The Convolutional Layer

In a CNN, the primary layer type is the Convolutional Layer, defined as:

$$\vec{a} = \sigma\left(\sum \underline{I} * \underline{K} + b\right) \tag{3.1}$$

Where b is the bias vector of length O_z. The $*$ operator represents a discrete convolution between $\underline{I}$ the input tensor and $\underline{K}$ the filter kernel. A discrete convolution with a kernel $K(n)$ and an input signal $I(m-n)$ in 1D is defined as:

$$(K * I)(m) = \sum_{n\,=\,-\infty}^{\infty} K(m)\, I(m-n) \tag{3.2}$$

The Convolutional Layer uses a series of discrete convolutions between filter kernels $\underline{K}_i$ of size $K_x \times K_y \times I_z$, and an input feature map $\underline{I}$. To produce output feature maps O_i of size $O_x \times O_y \times O_z$. These convolutional filters are learnable parameters, a tensor of weights and a bias, the goal of which is to create feature maps that encode spatial information at varying spatial scales.

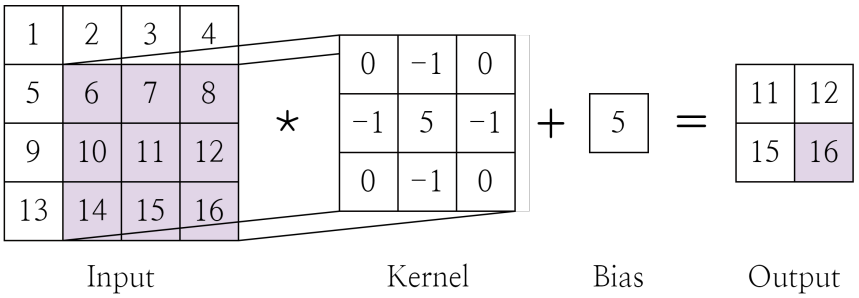


Figure 3: A convolution between an input tensor and a kernel tensor of weights and a bias

During the convolutional process, the kernel filter slides over the input, and the convolution between the input and kernel is computed, as shown in Figure 3. The distance the kernel moves on the input each step is called the stride S. In the case of an image, the input feature map is of shape $I_x \times I_y \times \ I_z$, where I_z is the number of channels, 3 for an RGB image.

$$\underline{O}_i = \underline{K}_i * \ \underline{I} \tag{3.3}$$

These feature maps are stacked along the depth dimension to form the layer's output volume, the size of which is controlled by the layer's hyperparameters K, N, P, and S. Parameter sharing ensures that the same filter is applied across the entire input, reducing the network's parameters. This sharing is achieved by using the same weights at different spatial locations of the input image.

4 Transformers

Transformers, introduced by Vaswani et al. in 2017 [11] have revolutionized machine learning, particularly in natural language processing (NLP). Unlike traditional recurrent neural networks (RNNs) and long short-term memory networks (LSTMs)[12], transformers rely entirely on self-attention mechanisms, enabling them to process input data in parallel rather than sequentially, significantly improving the efficiency and scalability of training on large datasets. Transformers

have since become the foundation for powerful models like GPT (Generative Pre-trained Transformer). Their versatility has since extended beyond text to domains such as images, as shown by Dosovitskiy et al [13] in 2020.

4.1 Attention Mechanisms

The shortcomings of LSTMs and RNNs can be solved by applying a technique called Attention, which was first introduced in 2014 by Bahdanau et al [14].

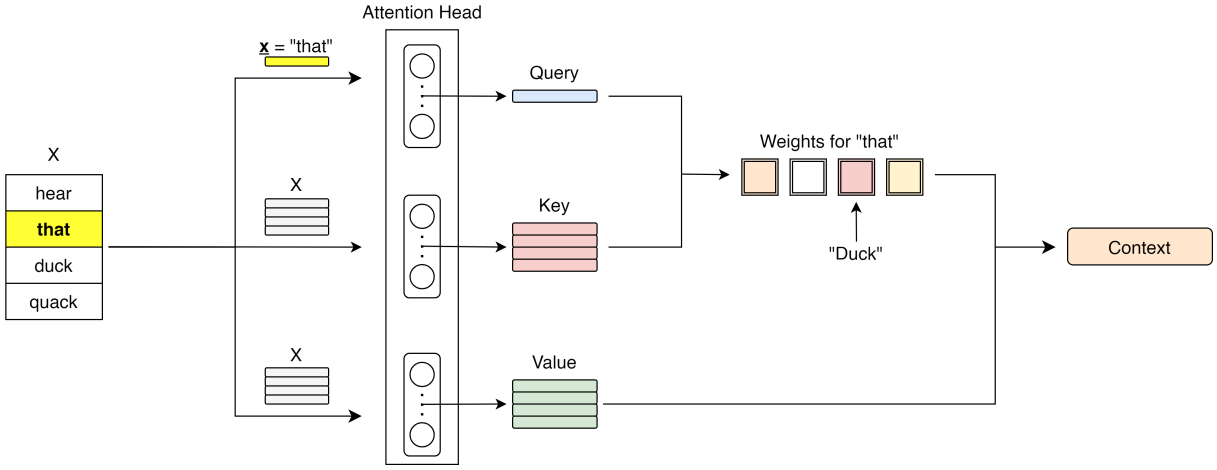


Figure 4: Illustration of how attention works on an input sentence

Attention Mechanisms transform the input sequence into a context vector that highlights the important parts of a sequence. This is done using three main components: queries (Q), keys (K), and values (V). Each word in the sequence is embedded into a high-dimensional space and processed through three sub-networks that form an Attention Head to generate Q, K, and V representations. The Attention Mechanism calculates how much focus each word should have by comparing the query of a word with the keys of all words in the sequence. These comparisons yield weights, which determine the importance of each word. The weighted sum of the values produces the final context vector.

4.2 Attention Is All You Need

The key feature of the Transformer is its ability to perform Self-Attention. That is, the Transformer can identify and weigh parts of an input sequence as important in relation to other parts, adjusting their influence on the output.

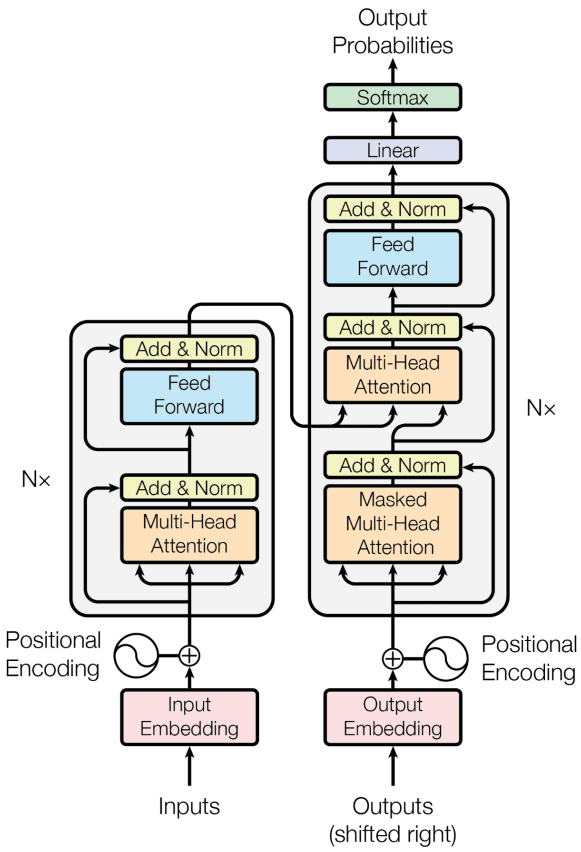


Figure 5: Transformer model architecture from *"Attention is All You Need"*, Vaswani et al. [11]

4.2.1 Self-Attention

1/ Tokenisation:

An input sequence is split into tokens, for a sentence the obvious choice is individual words, vowels, etc.

2/ Input Embedding:

The tokenised sequence is then converted into a set of dense vectors through a learnable embedding layer, which transforms discrete tokens into continuous representations.

3/ Projection:

The embedded dense vectors are projected into three distinct multi-dimensional spaces using different weight matrices to generate the Q, K, and V vectors for each token.

4/ Scaled Dot-Product Attention:

Calculates attention scores by taking the dot product (Figure 6) of each query vector with all

key vectors, scaling these scores by the square root of the key vector's dimension, and applying a softmax function to obtain attention weights. These weights represent the importance of each token relative to others.

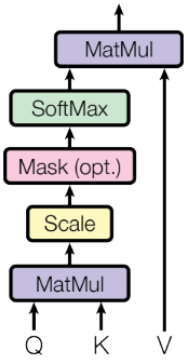


Figure 6: Illustration of Scaled Dot-Product Attention [11]

5/ Weighting:

Each token's attention weights are used to compute a weighted sum of the value vectors. This results in a new representation for each token that incorporates information from the entire sequence, weighted by the attention scores, see Section 4.1.

4.2.2 Multi-Head Attention

Multi-Head Attention uses Attention Heads, where each head performs the self-attention mechanism in Section 4.2.1 in parallel on the input sequence where multiple sets of Q, K, and V vectors are processed independently and their outputs are concatenated together and transformed, allowing the model to capture aspects of the input sequence simultaneously.

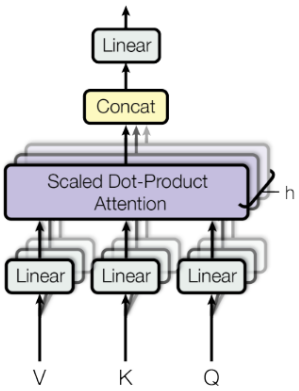


Figure 7: Illustration of Multi-Head Attention [11]

4.3 Vision Transformers

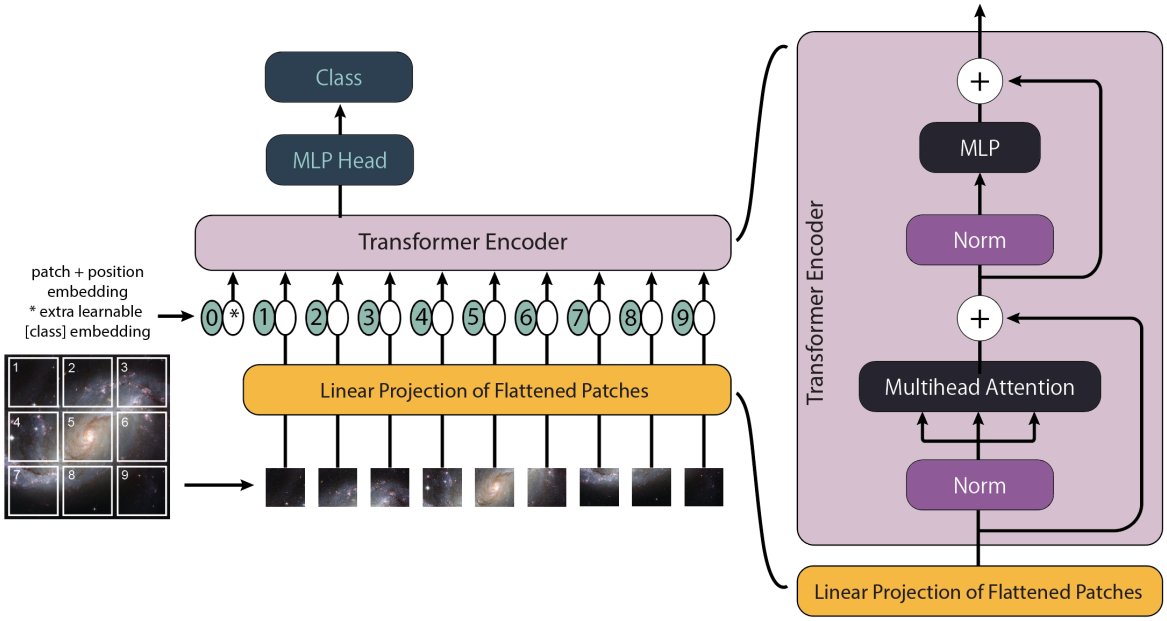


Figure 8: An input image is split into patches of 16 × 16px, the patches are then flattened and projected into a linear embedding layer, where a positional embedding and CLS learnable token are added before being passed into a standard Transformer encoder [13] (image credit, [15]).

4.4 Compact Convolutional Transformers

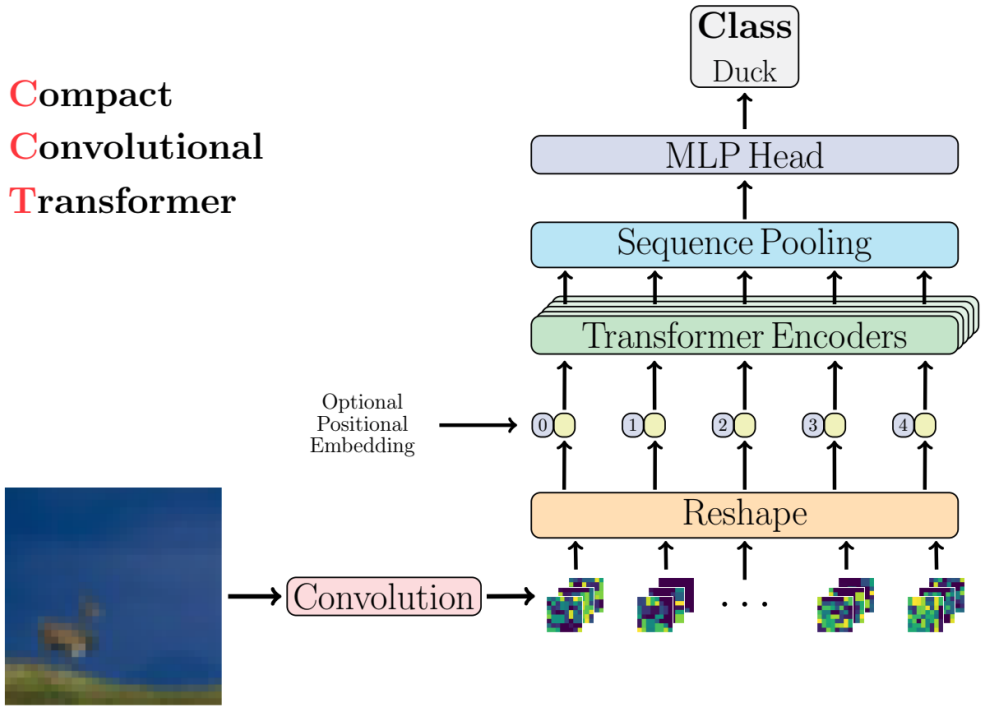


Figure 9: The Compact Convolutional Transformer applies convolutional layers to an input image before passing it into a standard Transformer encoder [16].

4.5 Convolutional Vision Transformers

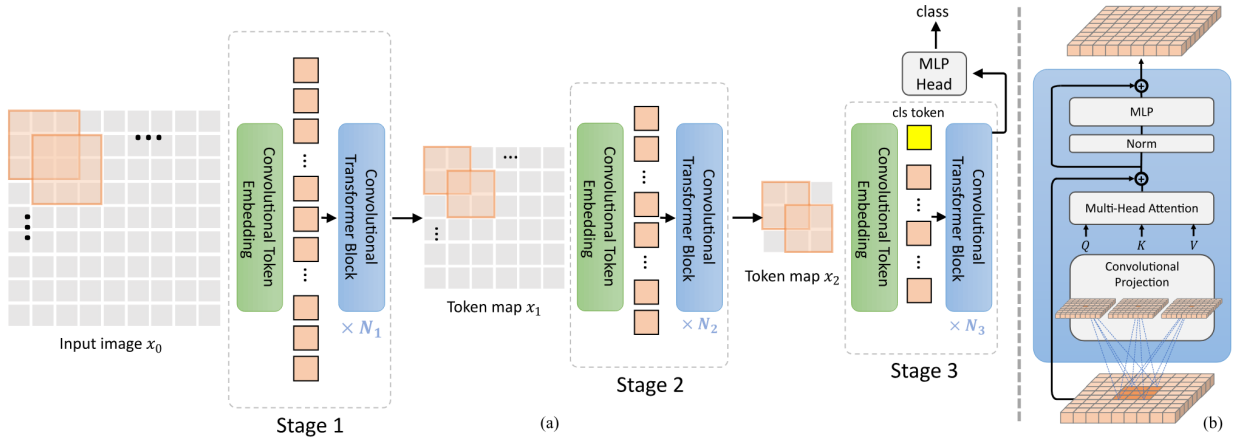


Figure 10: (a) The Convolutional Transformer pipeline, where a Convolutional Embedding replaces the standard embedding layer. (b) The Convolutional Transformer encoder, where a Convolutional Projection replaces the standard projection of tokens into Q, K, V. [17]

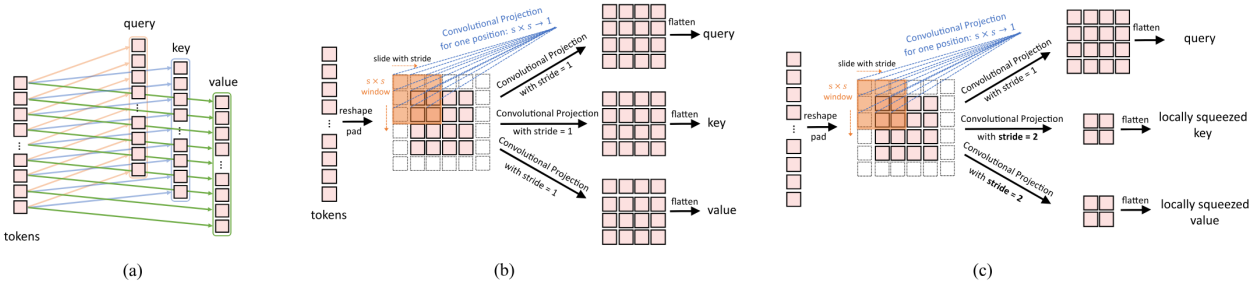


Figure 11: (a) The standard Transformer projection of tokens to queries, keys, and values. (b) The Convolutional Transformers, Convolutional Projection of an input map to output maps of queries, keys, and values. (b) How the stride of the Convolutional Projection Kernels can be used to reduce the size of the key and value output maps (Equation (A-2)), reducing the number of parameters of the layer (Equation (A-1)). [17]

5 The Galaxy Zoo

The Galaxy Zoo is a crowdsourced astronomy project, where volunteers can help assist in the morphological classification of galaxies. The classifications are provided by volunteers

participating in an online survey. Volunteers visit a website [18] where a random image of a galaxy is shown to them, along with a series of questions about the picture of the galaxy, as shown in Figure 12. The series of questions are designed to provide morphological information about the galaxies and follow the Galaxy Zoo 2 Decision Tree, a flow chart of questions. The Galaxy Zoo 2 project classified over 250,000 galaxies and provided over 60 million classifications in total.

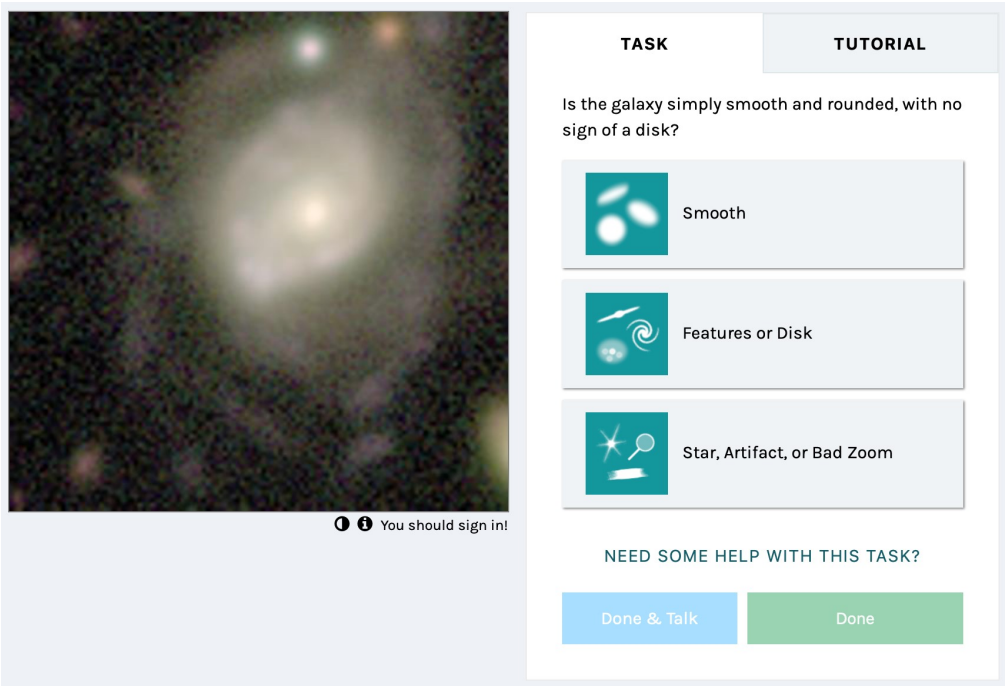


Figure 12: Screenshot of the Galaxy Zoo Questionnaire [18]

5.1 The Decision Tree

Figure 13 shows the Galaxy Zoo decision tree. It consists of 11 questions in total and contains 37 answers that correspond to a morphological feature.

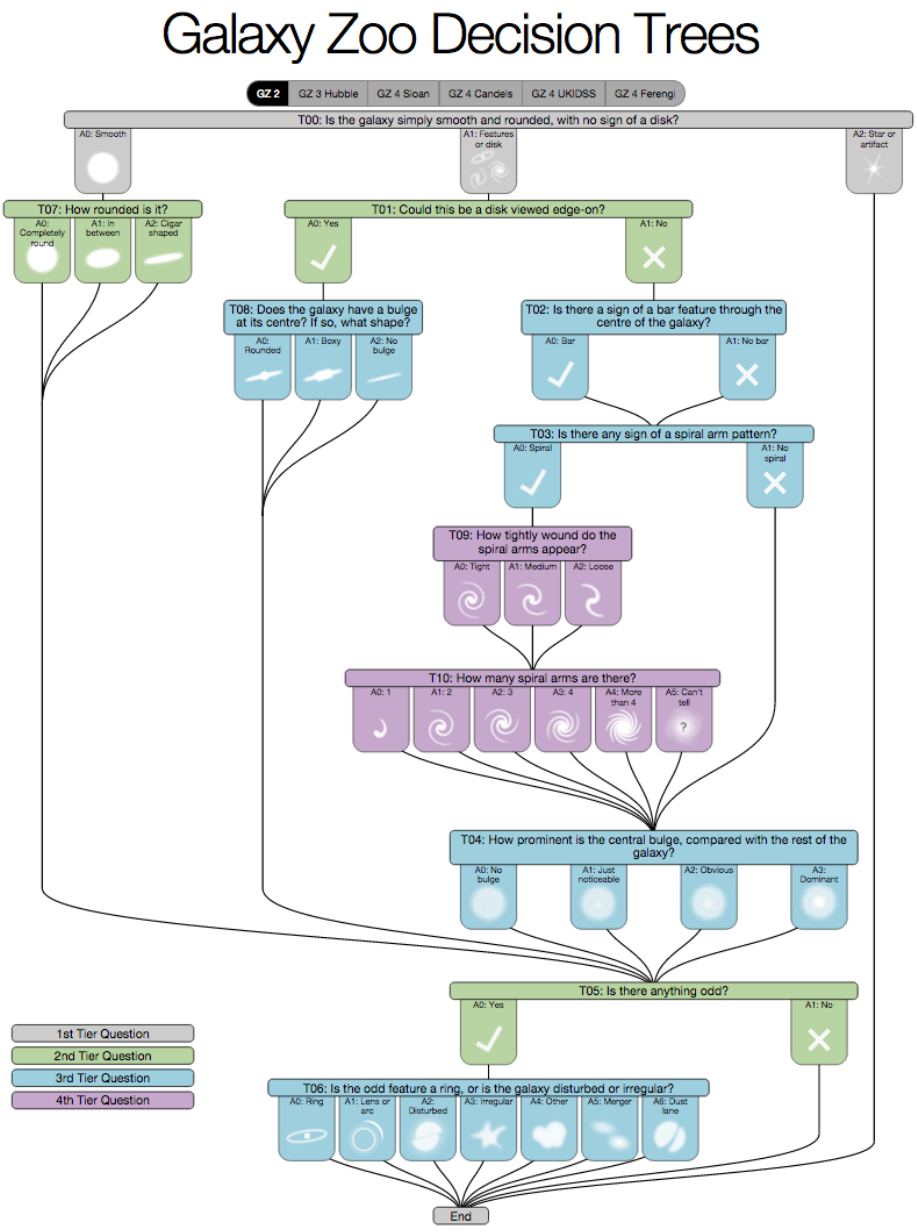


Figure 13: Galaxy Zoo Decision Tree by C. Krawczyk [19]

5.2 The Galaxy Zoo 2 Dataset

The dataset [20] obtained consists of two catalogue files containing the labels and file names of the images, and a folder containing over 200,000 JPG images of Galaxies, of size 424×424 px in 8-bit RGB, see (Figure 14) for an example image. The dataset for this project was obtained through the `"galaxy-datasets"` python module [21].

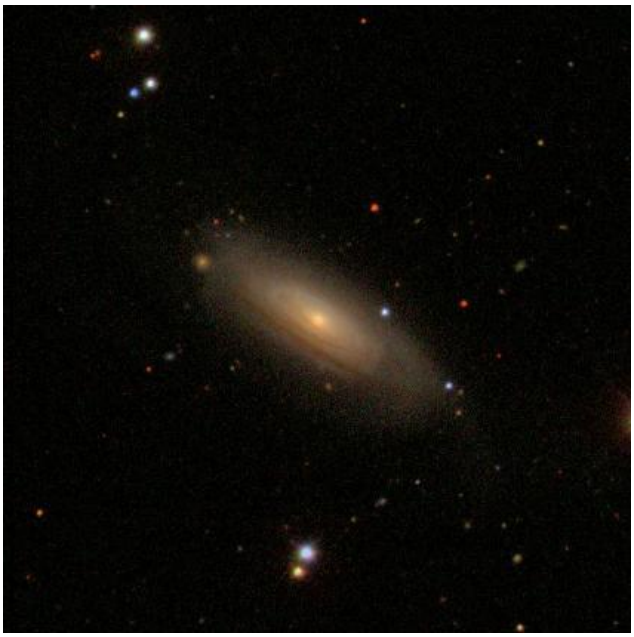


Figure 14: Example Image from the dataset

There are a few differences between this dataset and the decision tree (Figure 13), one being that this version of the dataset simplifies the decision tree slightly by removing the question: `"Is the odd feature a ring, or is the galaxy disturbed or irregular?"`.

5.2.1 Dataset Analysis

Shown in Figure 15 are the galaxies with the highest vote confidence for a given feature. The dataset was filtered such that galaxies with fewer than the average number of votes plus 2 standard deviations (equalling 21 votes) for a given feature were excluded to prevent galaxies with few total votes, and therefore inaccurate classifications from being considered.

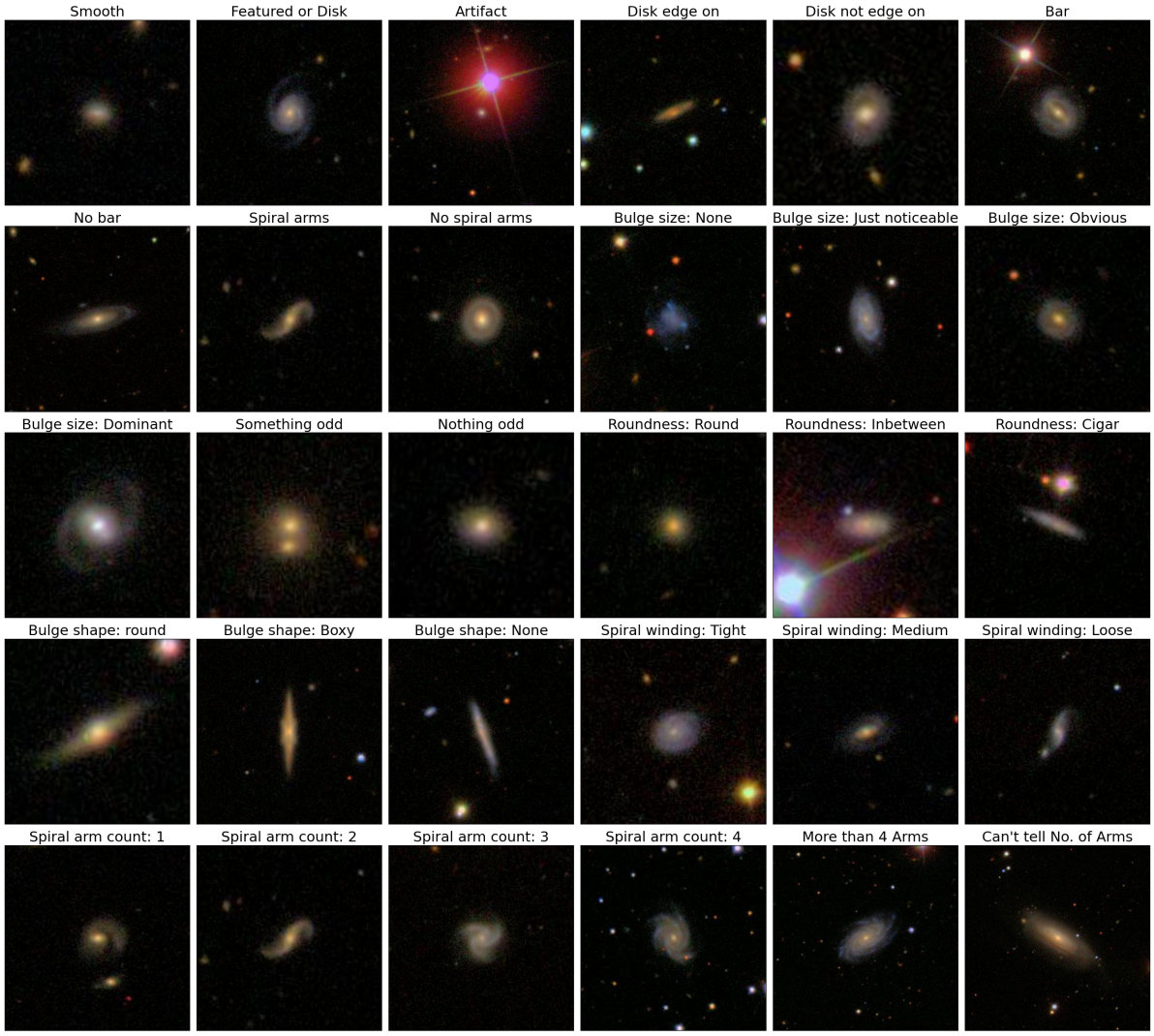


Figure 15: The Galaxies with the highest vote confidence for a given feature

The galaxies shown in Figure 15 visually match the voted features suggesting that the crowdsourced classifications are accurate, and may be a good approximation for the ground truth. However, this may not be the case; Figure 16 shows the mean confidence of each voted feature over the dataset.

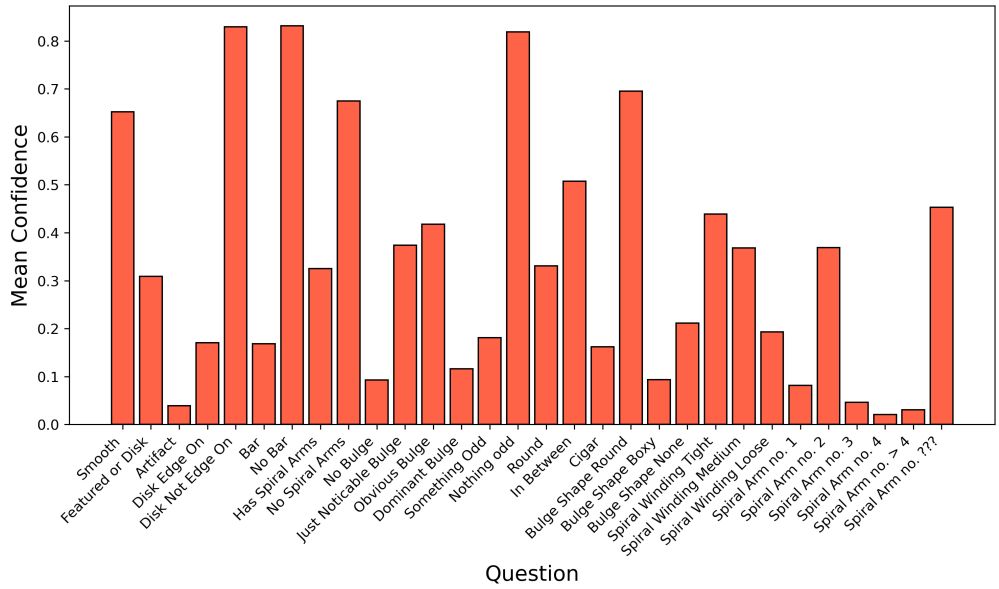
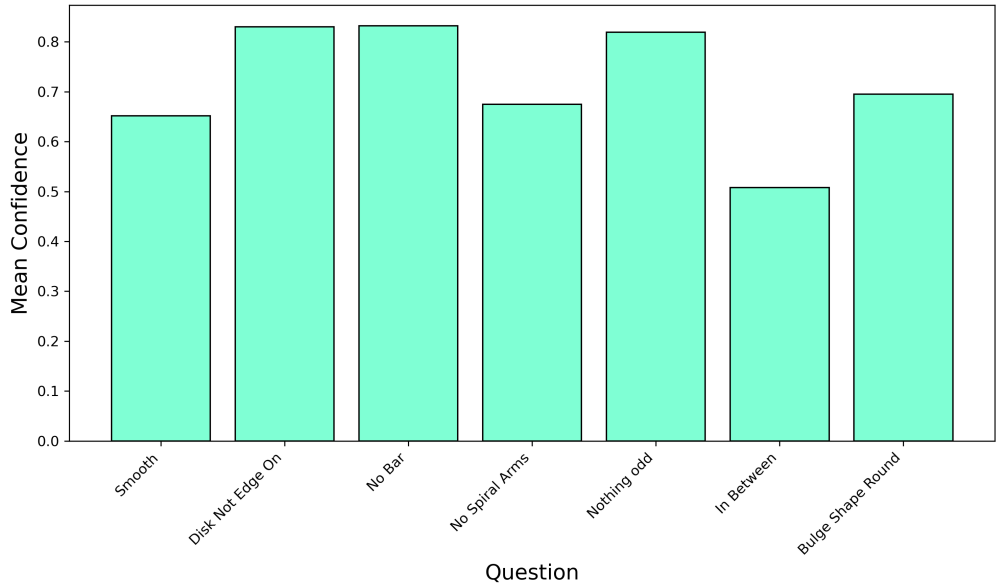
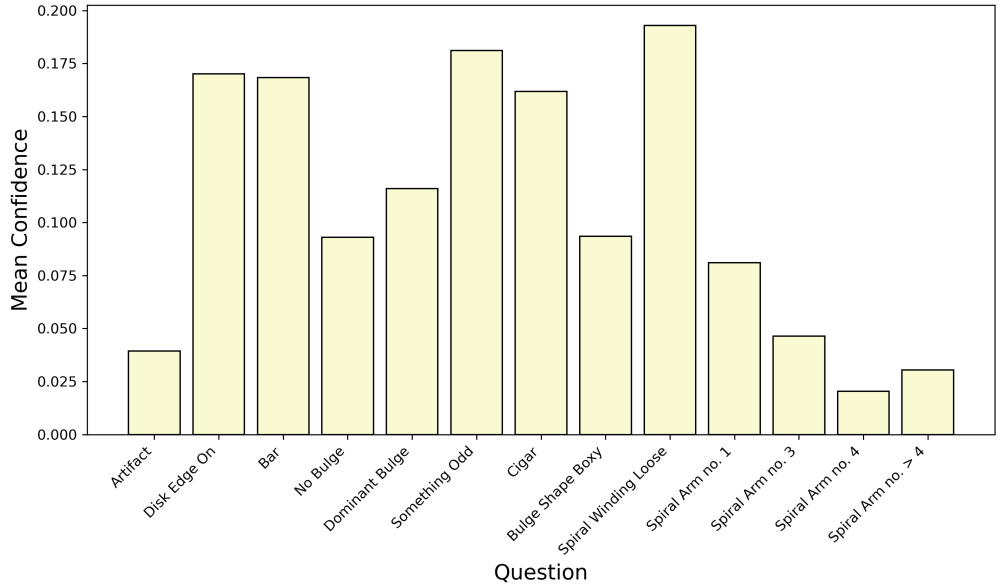


Figure 16: Mean confidence of voted features, showing wild variation in the confidence of votes for features, some such as `"Not Edge On"` have confidences above 0.8, while others such as `"Spiral Arm no.4"` are less than 0.025.

Figure 17a shows the features with a vote confidence of 0.5 or higher, and Figure 17b the features with a vote confidence with 0.2 or lower.



(a) Features with a vote confidence with 0.5 or higher. The features that are associated with smooth elliptical galaxies, seen face-on, with round central bulges are the most confidently answered



(b) Features with a vote confidence with 0.2 or lower. The features that are associated with spiral galaxies, edge-on disks, and bulge shapes or sizes are the least confidently answered. Interestingly there are pairs of features that could be confused with each other that are not confidently answered, such as `"Disk Edge On"` and `"Cigar"` suggesting there is potential confusion between the shape of the disk and if it is seen edge-on.

Figure 17: Most and least confidently answered features

This is better shown in Figure 18 which visually shows the correlations between voted

features. Some correlations between features are expected, such as `"Smooth"` & `"Featured"` since these features are mutually exclusive, and the absence of correlations between others, such as `"Round"` & `"Spiral Arms"` since they are not mutually exclusive, are expected. However, there are unexpected correlations, such as between `"Edge on Disk"` & `"Cigar"` suggesting volunteers could not tell if a galaxy was edge on or a cigar shape.

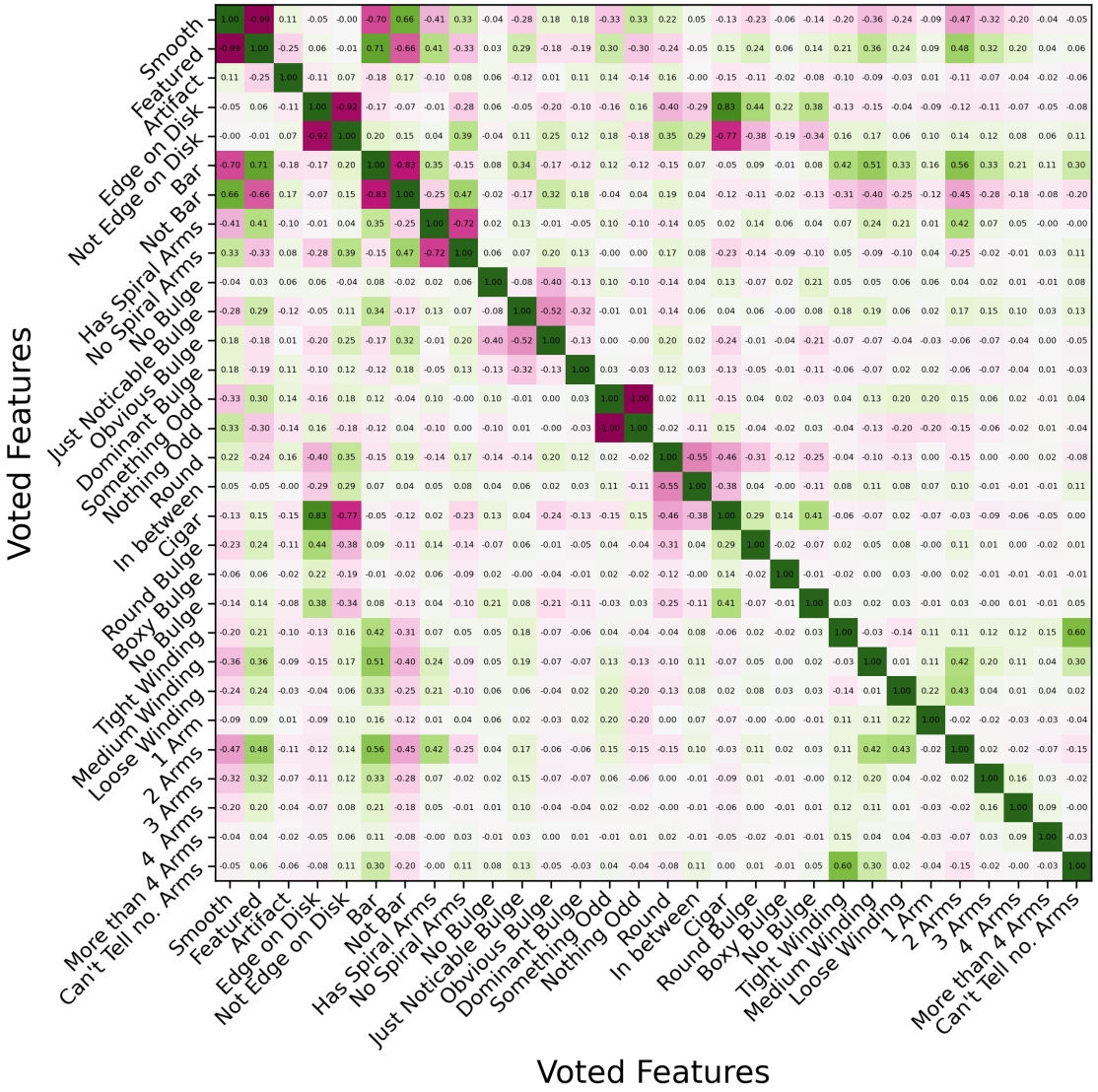


Figure 18: Correlation matrix between voted features for each question from the decision tree and the galaxies with said feature

If we focus on the parts of the correlation matrix that are questions of the decision tree and their answers, as shown in Figure 19, interesting patterns can be seen.

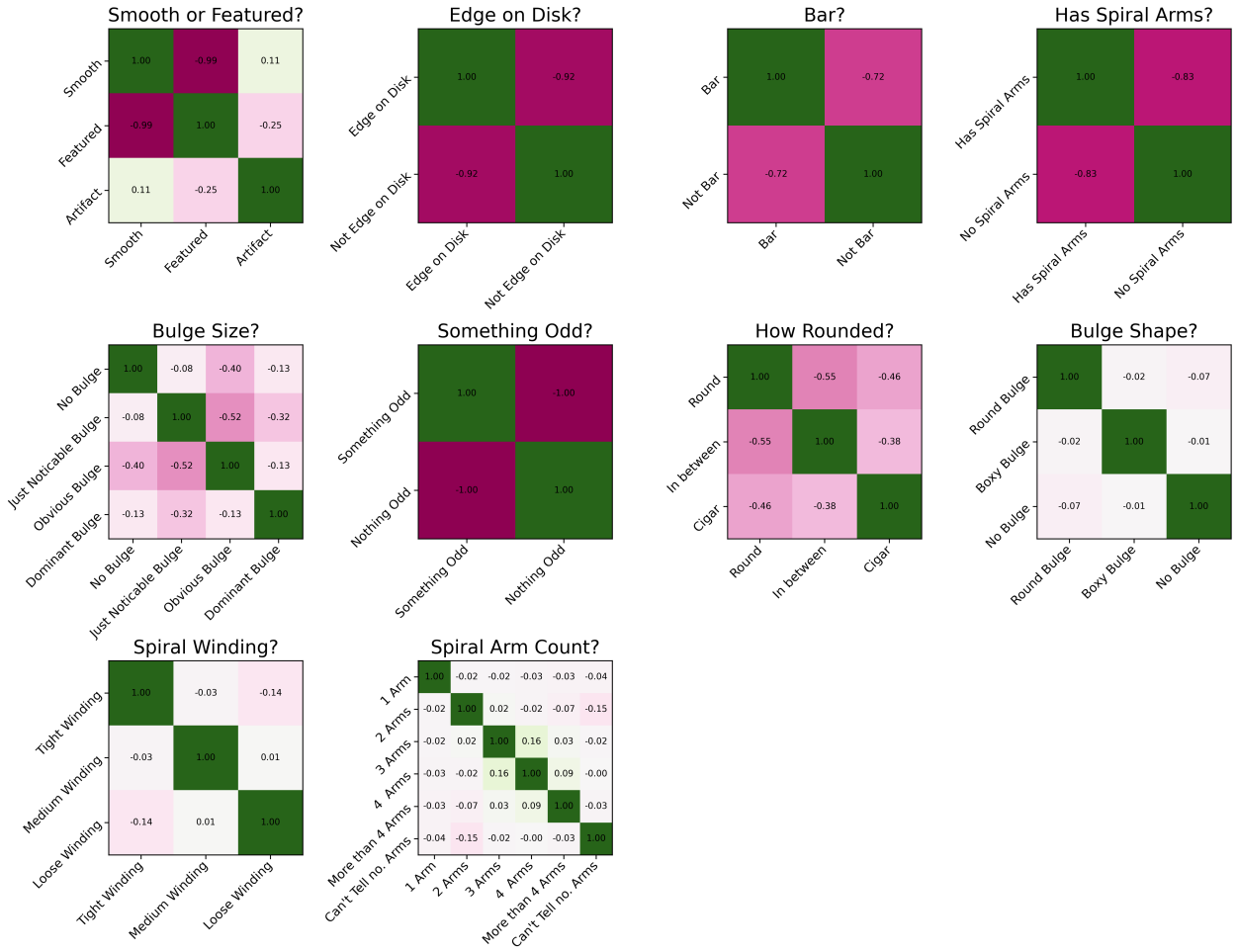


Figure 19: Correlation matrices for each question

The answers to each question are expected to be perfectly negatively correlated with each other, except with themselves, where they will be perfectly positive since they are mutually exclusive with each other (each question can only have one answer).

The matrices for `"Edge on Disk?"`, `"Bar?"`, `"Has Spiral Arms?"`, and `"Something Odd?"` align with expectations, showing strong negative correlations between each different feature. `"How Rounded?"` also matches expectations fairly well but has weaker negative correlations between features, indicating some confusion in identifying the shape of galaxies.

`"Smooth or Featured?"` is interesting, the correlations between `"Smooth"` and `"Featured"` are as expected, perfectly negatively correlated. However `"Artifact"` is not, indicating volunteers are often confused about what images contain artifacts. Volunteers are possibly thinking a very bright background star is the main subject of an image and labelling it as an artifact.

`"Bulge Size?"` is not what is expected, the only somewhat strong negative correlations

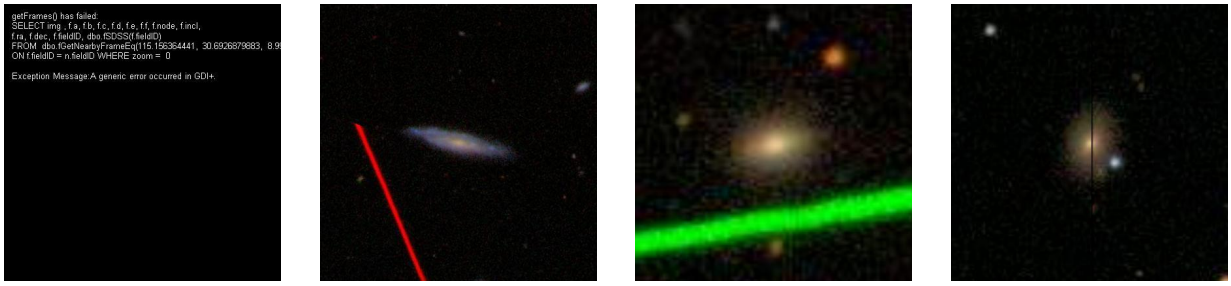
are between "Just Noticeable Bulge" and "Obvious Bulge", there is a fair amount of confusion between the volunteers when it comes to identifying the central bulge size.

"Bulge Shape?", "Spiral Winding", and "Spiral Arm Count?", are dramatically different from the expected correlation matrices, with no correlations between features except for the same feature, showing there is great confusion between volunteers when answering these questions.

See Section 5.2.3, for a discussion on how this may affect any potential models.

5.2.2 Problems With the Dataset

The quality of the PyPi version of the dataset is questionable. There are many image files listed within the catalogue files with names and labels that are missing from the images folder. Another problem is a very small amount of images appear to be an image of an error message Figure 20a. I found these missing images and error message images are the same across multiple machines. Other issues are images with thick coloured lines going partially through or across the entire image (Figure 20b & Figure 20c), and images with thin black lines going through them, Figure 20d.



(a) Error message (b) Part of a coloured line (c) Coloured line (d) Thin black line

Figure 20: Examples of poor quality images found in the dataset

5.2.3 Potential Limitations

Any models built will ultimately be limited by the volunteer classifications. For galaxies with features that are confused often by the volunteers, it is highly likely those crowdsourced labels will be incorrect, leading to to the model getting confused when trying to learn how to identify said feature.

The models will most likely be able to predict between the questions shown in Figure 19 that have strong negative correlations between the other features reasonably well. However, for the other questions, the models will probably struggle to learn anything meaningful and may not perform better than random chance.

6 Methodology

6.1 Dataset Preprocessing

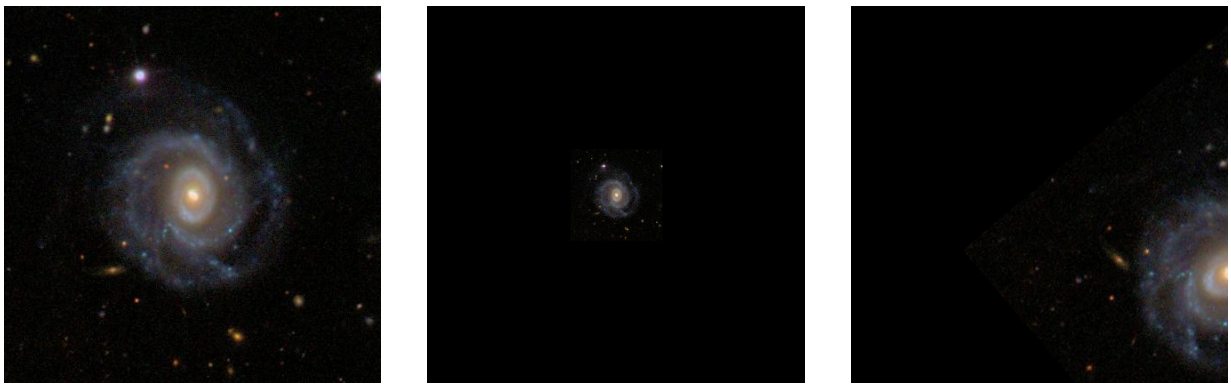
The dataset was preprocessed to construct file paths to the images and the question `"Is there a central bulge?"`.

6.1.1 Dataset Cleaning

The dataset was cleaned to remove missing file paths from the catalogues, and the error message images (Figure 20a) removed from the catalogue by checking if the bottom half of the image only contained pure black pixels.

6.2 Image Augmentation

The augmentations chosen to apply to the images should abuse galaxies' symmetries without increasing the problem's complexity or adding ambiguity. We can flip, rotate, and scale the images to create extra views without worry as galaxies are rotationally symmetric and centrally located in the image. However, we need to be careful with scaling as zooming out too much will create black pixels around the image as padding Figure 21b, and destroys information. Translation is another augmentation to use cautiously. Excessive translation will shift the main galaxy off-centre Figure 21c (even potentially out of frame), making a background galaxy appear central. This could confuse the model and misidentify the galaxy type. Excessive translations also increase the complexity of the problem as the main galaxy could be anywhere in the image. Therefore, I limited scaling or translation to tens of pixels.



(a) Original Image (b) Bad Scaling (c) Bad Translation

Figure 21: Examples of the problems scaling and translation can cause

6.2.1 Preprocessed Augmentation

To save time when training I chose to apply augmentations to images as a preprocessing step. Each image of the original dataset had 12 additional views created, totalling 13 unique views, as shown by Figure 22. Each image view had 3 different augmentations applied. The possible augmentations are: do nothing, horizontal or vertical flip, rotation by an arbitrary angle, zoom in or out, blurr, de-noise to remove salt and pepper noise, add gaussian noise or translate by a few pixels by a random amount. The augmentations were chosen randomly without replacement and then applied to create each view.

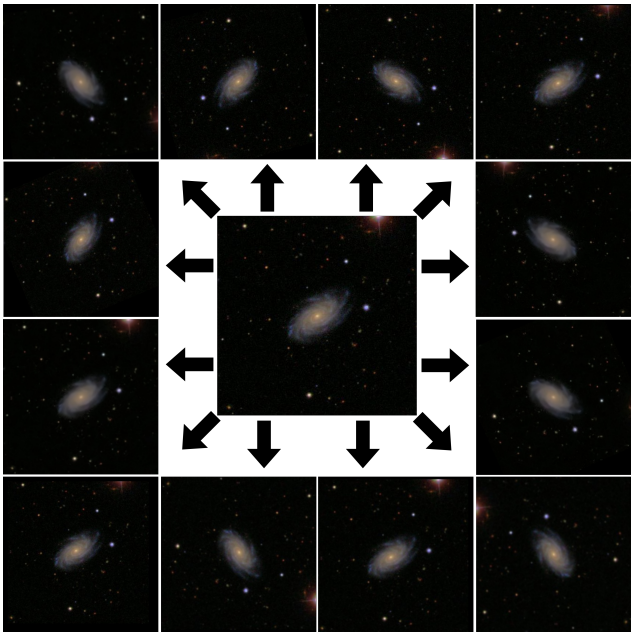
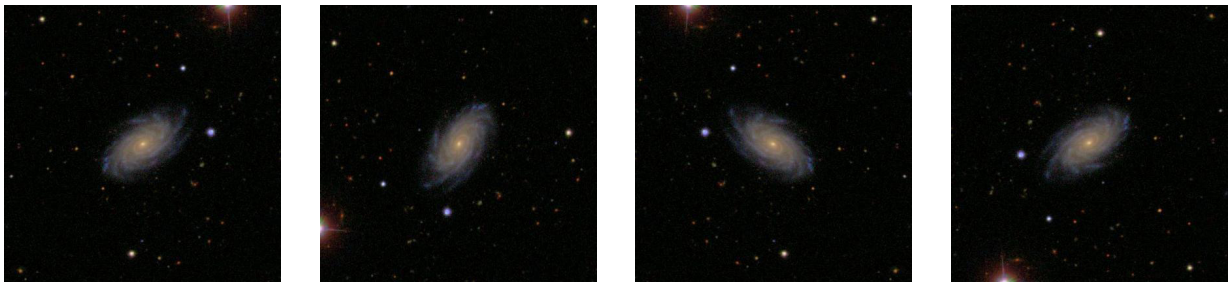


Figure 22: Original image in the centre with 12 unique views, each generated by applying 4 different augmentations at random

6.2.2 On the Fly Augmentation

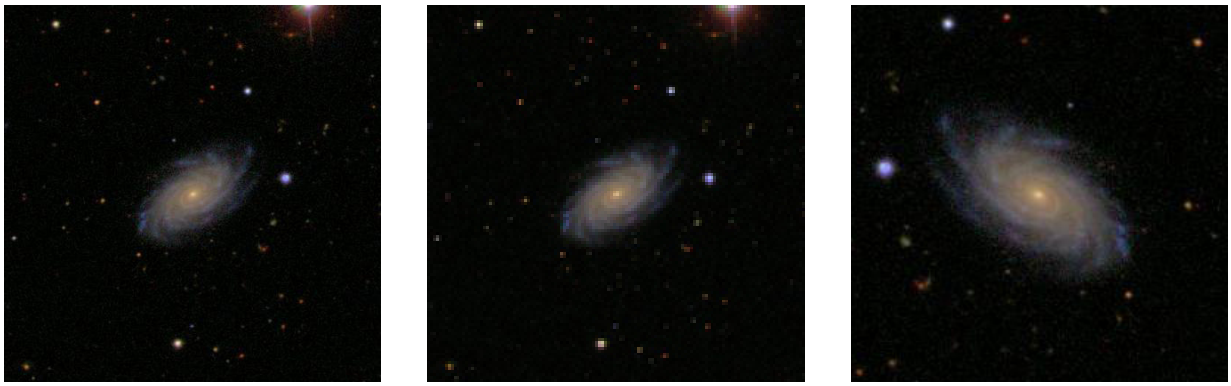
Complex on-the-fly augmentation, such as rotation by arbitrary angles, scaling, and translation with a custom Keras layer was used sparingly due to it significantly slowing the training process. It was only used when training models appeared to be promising to reduce overfitting, and not for initial testing of ideas.



(a) Original image (b) Orthogonal rotation (c) Horizontal flip (d) Vertical flip

Figure 23: Possible augmentations `fetch_image_label_pair` can apply to an image

However, in `dataframe.py` the `fetch_image_label_pair` method of the `DataFrame` class can randomly choose to flip an image around a specific axis, or apply an orthogonal rotation (Figure 25) if enabled, which it was since it did not affect training time significantly. In this method, the images can also be rescaled to a desired size or cropped to a specified bounding box. This is also where the preprocessing of images for pre-trained models is done.

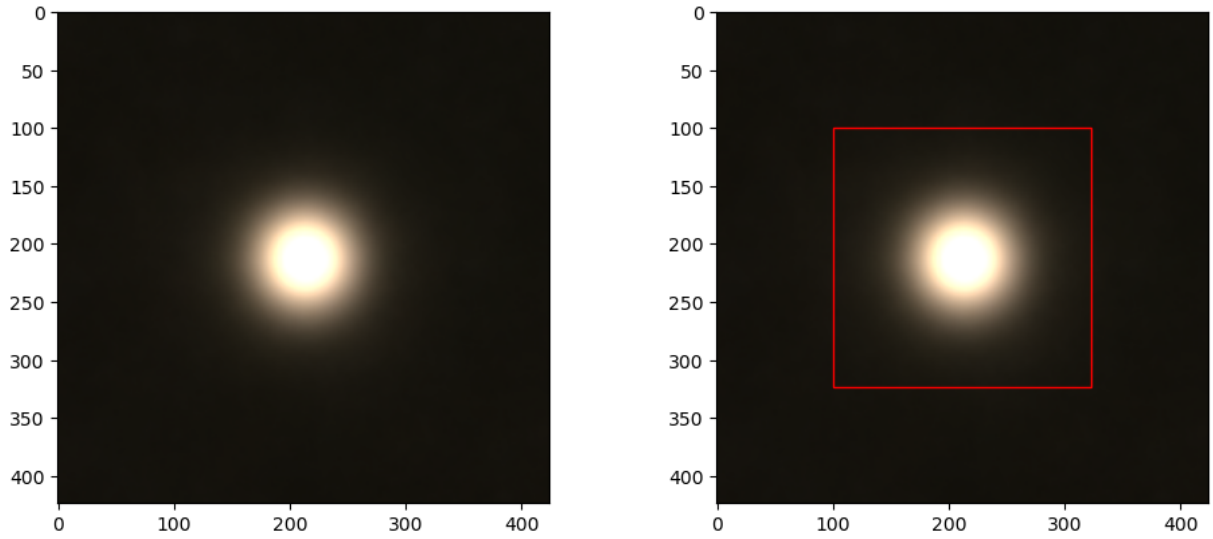


(a) Original Image 424×424px (b) Resized to 128×128px (c) Cropped to 256×256px

Figure 24: Possible augmentations `fetch_image_label_pair` can apply to an image

6.2.3 Image Cropping

I chose to crop images to a size of 224×224px when using CNN models. This was chosen by creating a script that found the average image over 2000 images (Figure 25a) and then finding the centred bounding box that contained all the pixels above 8.5% brightness (Figure 25b).



(a) Average image for 2000 images

(b) Optimal Bounding box

Figure 25: The optimal bounding box found was 227×227px

When using Transformer models, I chose to crop the images to a size of 256×256px, as shown in Figure 26. This is because I was then able to split the images patches that are 16×16px and have a 'nice' number of patches (256).

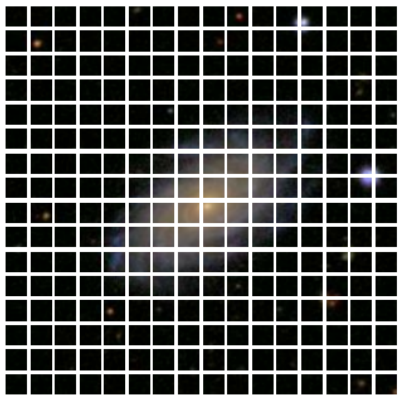


Figure 26: An image split into 256, 16×16px patches to be passed into a transformer model

Cropping to the required dimensions was always done over resizing as downscaling destroys information whereas cropping preserves it. Furthermore, since most of the image outside the found bounding box is black, removing it has minimal affect on the important information about a galaxy in the image.

6.3 Classification Schemes and Labels Used

The morphological scheme defines the classification problem being solved, being the set of morphological feature labels provided to the network for learning.

6.3.1 Morphological Classification Schemes

Table 1 contains the questions used throughout this paper, the question number they were given, and the feature labels associated with each question. Only the labels in "Used" represent the features of a question and were given to the model to predict, see Section 6.3.2 for more information.

Question No.	**Question**	**Labels**	
		Used	**Not used**
Q0	Smooth or Featured?	"Smooth"	"Featured"
Q1	Disk Edge on?	"Yes"	"No"
Q2	Spiral Arms?	"Yes"	"No"
Q3	Has Bar?	"Yes"	"No"
Q4	Bulge Size?	"Dominant", "Obvious", "Just Noticeable"	—
Q5	Something Odd?	"Yes"	"No"
Q6	How Round?	"Round", "In-between", "Cigar"	—
Q7	Bulge Shape?	"Round"	"Boxy"
Q8	Spiral Winding?	"Tight", "Medium", "Loose"	—
Q9	Spiral Arm Count?	"1", "2", "3", "4", "4+", "Can't Tell"	—
Q10	Has Bulge?	"Yes"	"No"

Table 1: The questions used, the feature labels they use, and the numbers given to them

Classification Scheme	Questions Used From Table 1
Question <N>	Q<N>
Hubble	Q0, Q2, Q3, Q10
Reduced Set	Q0, Q1, Q2, Q3, Q6, Q10
All Features	Q0, Q1, Q2, Q3, Q4, Q6, Q7, Q8, Q9, Q10

Table 2: The morphological classification schemes used, and the questions they use with their associated labels.

Table 2 shows the morphological classification schemes used and the associated questions they use, and therefore the labels the schemes use (Table 1). The scheme defines what

morphological features of a galaxy a model will have to identify, and thus controls the complexity of the problem being solved.

6.3.2 Label Binarization

I chose to binarize the labels for each feature as I wanted my models to detect the presence or absence of morphological features, instead of replicating the possible crowdsourced vote fractions an image could have (i.e a galaxy that has a 70% chance of being a spiral galaxy is absurd, it either is a spiral galaxy or isn't). The method of label binarization uses the question structure of the dataset, each vote fraction label of the dataset is given its question as a number. The labels that are answers to the same question are grouped together and converted to binary (Figure 28) by making the label with the largest value in group one and setting the rest to 0. The flow chart below (Figure 27) encapsulates the logic of the `"__to_binary"` method of the `"DataFrame"` class within `"dataframe.py"`.

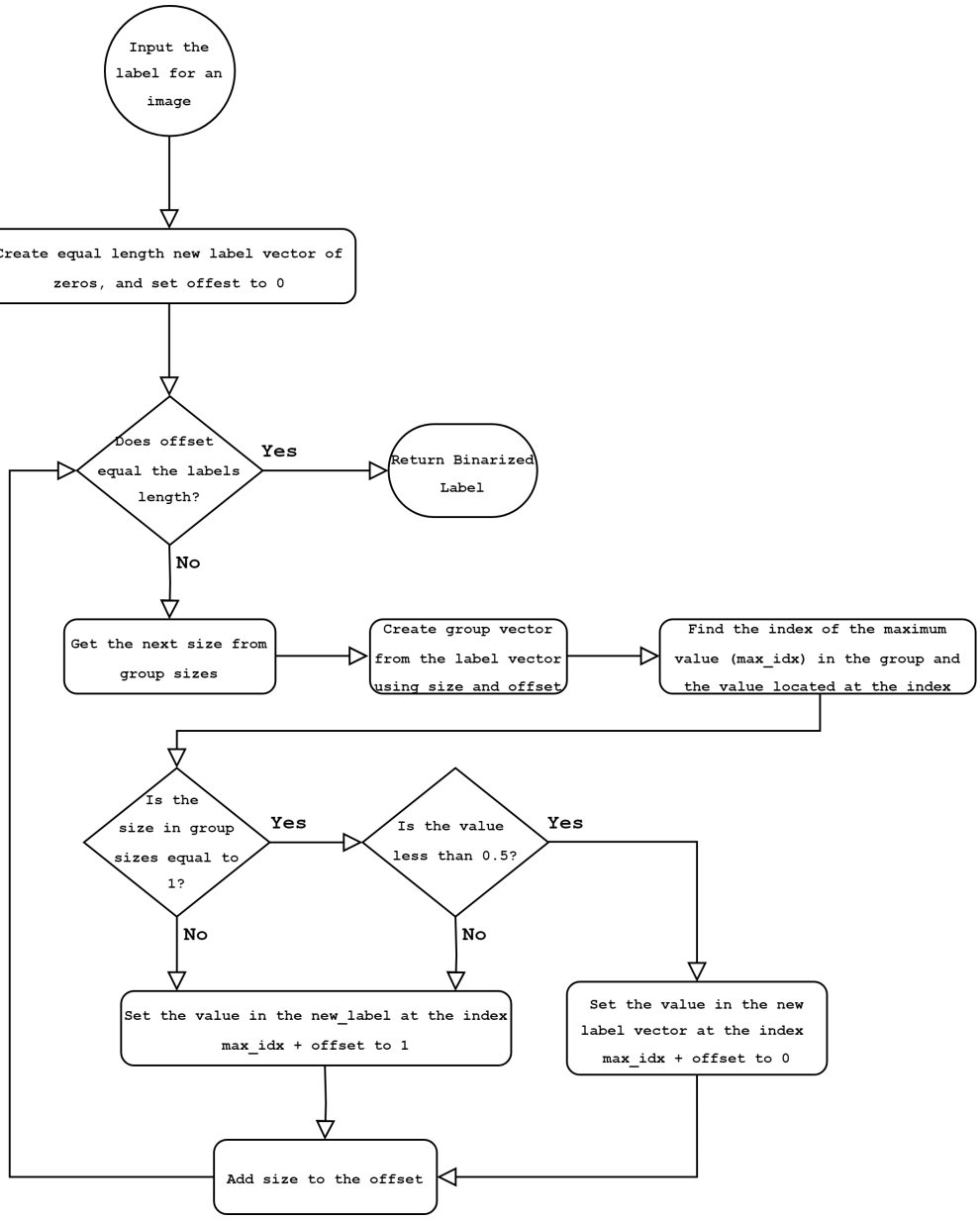


Figure 27: The logic of the `"__to_binary"` method of the `"DataFrame"` class within `"dataframe.py"`

For the questions `Q0: "Is the galaxy smooth and rounded, with no sign of a disk?"`, and `Q1: "Is the disk edge on?"`. The answers to these questions for a galaxy will be represented by a label such as `[0.1, 0.2, 0.7, 0.1, 0.9]`. This method of label binarization will convert this label to `[0, 0, 1, 0, 1]` where the sequences `0, 0, 1` and `0, 1` are the respective answers to the questions indicating the presence and absence of features.

For the questions with only 2 answers that are mutually exclusive, we choose to drop the second label of the pair as it contains redundant information already contained by the first.

i.e. "smooth_fraction" represents the same information as using both "smooth_fraction" and "featured-or-disk_fraction" since they are mutually exclusive. This results in a reduction of the maximum possible labels a model will have to learn from 32 to 21.

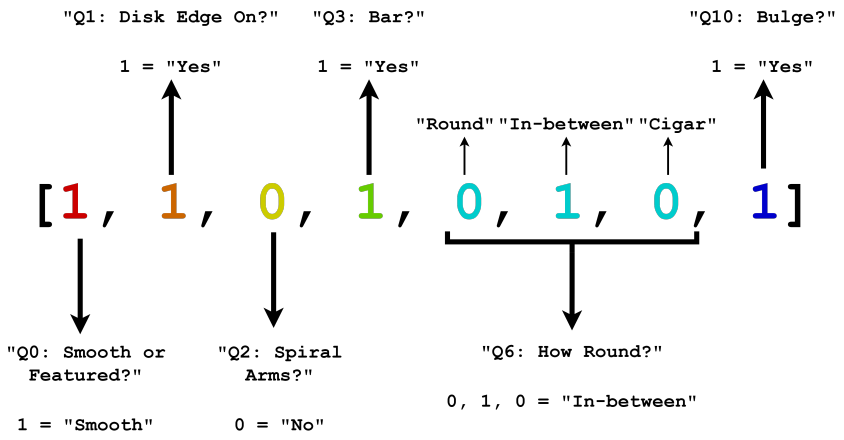


Figure 28: Example of a binarised label using the "Reduced Set" morphological scheme (Table 2), decomposed into it's constituent questions and the associated features.

6.3.3 Removal of Certain Labels

I chose to remove certain labels, be that entire questions or answers to a question. This was done to simplify the complexity of the problem and because the labels and the morphological feature(s) they represented were questionable.

"Artifact" as an answer to Q0:"Smooth or Featured?" was removed, as the class is so rare (less than 1% of the images have as a label) and therefore is difficult to classify; even the volunteers struggled to classify this feature well.

"No Bulge" was removed as an answer to Q4:"Bulge Size?", and Q7:"Bulge Shape?". This is because these features share redundant information, Q4 cannot be yes and Q7 cannot be no for the same galaxy, there either is a central bulge or isn't and there needs to be a label that contains this information (see Sec: 6.3.4). z I decided to remove Q5:"Is there something odd?" entirely, as this question condenses the possible morphological features contained by "T06:Is the odd feature a ring, or is the galaxy disturbed or irregular?", into a single Yes or No answer. Trying to get a model to identify the presence of 7 different morphological features that look completely different with one label is absurd.

6.3.4 Question 10: Is there a Central Bulge?

Since no label indicates the presence or absence of a central bulge outside the context of its size or shape I decided to create a new question Q10:"Is there a Central Bulge?". With labels "bulge-gz2-yes_fraction" and "bulge-gz2-no_fraction" where:

$$\texttt{"bulge-gz2-yes\_fraction"} = 1 - \texttt{"bulge-gz2-no\_fraction"} \tag{6.1}$$

This is done in `"cleandataset.py"` with the function `"construct_bulge_present_question"`. Which takes the labels:

`"bulge-size-gz2_no_fraction"` & `"bulge-shape-gz2_no-bulge_fraction"`

For each image and finds their average, which is `"bulge-gz2-no_fraction"` and uses Equation (6.1) to construct the new question.

6.4 Loss Function Weighting

A method of dealing with imbalanced classes in data during model training is to apply a weighting to the loss function. The weighting is chosen such that uncommon classes in the dataset are weighed more than common ones. This is achieved by multiplying the error in each predicted class by the weight of said class within the loss function, thereby causing a model to pay more attention to the features and patterns of uncommon classes.

Many weighting methods exist such as inverse number of samples, and inverse square root number of samples; the methods of which are self-explanatory. I instead chose to use the weighting method introduced by Cui et al. in "Class-Balanced Loss Based on Effective Number of Samples"

6.4.1 Effective Number of Samples

Effective Number of Samples (ENS) Weighting [22] uses the idea that as the number of examples of a class increases, the additional benefit of a model seeing each example has diminishing returns.

The ENS Weighting for a specific class in a dataset can be calculated as follows:

$$\beta = \frac{N-1}{N} \tag{6.2}$$

Where N is the total number of example images in the dataset and β controls the amount of re-weighting. $\beta = 0$ is no re-weighting and $\beta = 1$ is the inverse number of samples weighting.

For every class in the dataset, find the number of examples for that class n_i. The ENS for a class c is defined as:

$$E_c = \frac{1-\beta^{n_i}}{1-\beta} \tag{6.3}$$

The weighting for that class is then:

$$w_c = -\frac{1}{E_c} \tag{6.4}$$

We then demand that over all the classes C:

$$\sum_{c=1}^{C} w_c = N \tag{6.5}$$

This means we divide the weights for each class by the total of the weights and multiply them by N, giving us our final ENS weights W_c to be given to the loss function.

$$W_c = \frac{w_c}{\sum_{c=1}^{C} w_c} \times N \tag{6.6}$$

6.5 Models Used

6.5.1 Pretrained Convolutional Neural Networks

Model List
ResNet50
ResNet50V2
VGG16
VGG19
MobileNetV3Small
MobileNetV3Large
Xception
InceptionV3

Table 3: List of Pre-Trained CNNs used

6.5.2 Vision Transformer Models

Transformer Model Specifications					
Model	Projection Dimension	Attention Heads	Transformer Units	MLP Units	No. Patches
ViT	64	8	12	2048×1024	256
ViT	64	12	12	2048×1024	256
ViT	64	16	12	2048×1024	256
ViT	64	16	16	2048×1024	256
ViT	64	16	16	2048×1024	256
ViT	128	12	12	2048×1024	256
ViT	64	12	12	4096×2048	256
CCT	64	2	2	2048×1024	256
CvT	32	8	8	2048×1024	256

Table 4: Specifications of Different Vision Transformer (ViT), Compact Convolutional Transformer (CCT), and Convolutional Vision Transformer (CvT) Models experimented with

6.6 Model Training

The Pre-trained CNNs used ImageNet weights and had their layers frozen apart from the MLP head layers during preliminary testing, then during fine-tuning all layers were unfrozen.

For preliminary testing, k-fold cross-validation was conducted with the dataset divided into five folds. Each fold was split, with 80% used for training and 20% for validation. The learning rate was set at 0.001. Training continued until overfitting was observed in the best-performing model, at which point it was reverted to an earlier checkpoint for fine-tuning.

During the fine-tuning phase, the training dataset was split into 80% training and 20% validation. Models were trained for five epochs on the entire dataset, including pre-generated augmented views, with each epoch covering the full dataset. A learning rate of 0.0001 was used for fine-tuning.

7 Results

7.1 Pre-Trained Convolutional Neural Networks

7.1.1 ResNet50

ResNet50					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Q0	0.1555	0.9304	0.0969	0.2208	0.9303
Q1	0.2870	0.8942	0.1800	0.2880	0.8942
Q2	0.4355	0.7899	0.2734	0.3749	0.7900
Q3	0.3464	0.8515	0.2228	0.3255	0.8516
Q4	0.4644	0.6459	0.3102	0.3933	0.7686
Q5	0.2429	0.8904	0.1448	0.2757	0.8904
Q6	0.2039	0.8645	0.1056	0.2517	0.9104
Q7	0.5914	0.6528	0.4126	0.4532	0.6528
Q8	0.6050	0.4054	0.4147	0.4579	0.6847
Q9	0.4031	0.3394	0.2502	0.3513	0.8471
Q10	0.6692	0.6773	0.4866	0.4879	0.6773
Reduced Set	0.2351	0.5758	0.1441	0.4384	0.8982
All Features	0.3617	0.5557	0.2285	0.3403	0.8376

Table 5: General Performance Metrics for Different Classification Schemes

ResNet50								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Q0	0.9851	0.9299	0.9307	0.9078	19510.2695	19490.4082	1445.5914	1425.7292
Q1	0.9503	0.8939	0.8946	0.8172	18726.6113	18710.6289	2225.3696	2209.3884
Q2	0.8803	0.7900	0.7899	0.7692	16536.2285	16537.1328	4398.8672	4399.7700
Q3	0.9272	0.8515	0.8516	0.7346	17870.5215	17868.1973	3067.8015	3065.4766
Q4	0.8237	0.7068	0.5223	0.5883	10953.0859	37331.5938	4540.4053	9982.9131
Q5	0.9642	0.8904	0.8904	0.7560	18627.2969	18627.2969	2308.7017	2308.7017
Q6	0.9736	0.8728	0.8557	0.8655	17917.2656	39272.4805	2599.5181	3018.7327
Q7	0.7325	0.6528	0.6529	0.5761	13640.6270	13637.2627	7298.7363	7295.3726
Q8	0.6143	0.6050	0.1562	0.3961	3255.4526	39729.5000	2142.4973	17680.5469
Q9	0.6807	0.7016	0.1433	0.2442	2996.1726	103394.6172	1285.3809	17939.8262
Q10	0.6773	0.6773	0.6773	0.4038	14147.7656	14147.7656	6788.2329	6788.2329
Reduced Set	0.9594	0.8840	0.8075	0.4482	11004.5254	6081.2373	6081.2373	11004. 5254
All Features	0.8838	0.8119	0.5716	0.1726	72872.3281	295411.1562	16866.7910	54505.7148

Table 6: Classification Metrics for Different Classification Schemes

7.1.2 ResNet50V2

ResNet50V2					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Reduced Set	1.2183	0.7331	0.3266	0.5358	0.6784
All Features	4.8712	0.7419	0.3039	0.5340	0.6972

Table 7: General Performance Metrics for Different Classification Schemes

ResNet50V2								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Reduced Set	0.7073	0.5378	0.3949	0.1144	22546.9629	91102.9375	19325.3301	34512.7617
All Features	0.5863	0.4527	0.2069	0.0419	26411.3145	280147.6562	31942.4453	101154.5625

Table 8: Classification Metrics for Different Classification Schemes

7.1.3 InceptionV3

InceptionV3					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Reduced Set	18.9751	0.6169	0.1720	0.3041	0.8769
All Features	0.4118	0.6259	0.2545	0.3584	0.8185

Table 9: General Performance Metrics for Different Classification Schemes

InceptionV3								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Reduced Set	0.9203	0.8574	0.7664	0.4001	43663.4258	103142.1484	7283.1226	13399.2939
All Features	0.8597	0.8011	0.4974	0.1490	63380.2461	296580.6562	15646.6592	64048.4297

Table 10: Classification Metrics for Different Classification Schemes

7.1.4 Xception

Xception					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Reduced Set	0.2351	0.6690	0.1353	0.2695	0.8992
All Features	0.3559	0.6659	0.2285	0.3380	0.8392

Table 11: General Performance Metrics for Different Classification Schemes

Xception								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Reduced Set	0.9602	0.9018	0.7910	0.4177	45268.6992	105384.7656	4929.7812	11904.7480
All Features	0.8873	0.8225	0.5677	0.1560	72405.5156	296626.7500	15634.1074	54989.6133

Table 12: Classification Metrics for Different Classification Schemes

7.1.5 VGG16

VGG16					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Reduced Set	0.5654	0.7478	0.3949	0.4372	0.7243

Table 13: General Performance Metrics for Different Classification Schemes

VGG16								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Reduced Set	0.7400	0.6296	0.4623	0.1070	26393.3965	94977.0000	15478.6016	30638.9941

Table 14: Classification Metrics for Different Classification Schemes

7.1.6 VGG19

VGG19					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Reduced Set	0.3232	0.7475	0.3232	0.5685	0.6768
All Features	0.5346	0.7473	0.3644	0.4233	0.7352

Table 15: General Performance Metrics for Different Classification Schemes

VGG19								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Reduced Set	0.6524	0.5234	0.5760	0.2508	32870.22	80460.30	29937.77	24219.70
All Features	0.7251	0.6294	0.2071	0.0407	26335.27	296719.03	15536.73	101064.95

Table 16: Classification Metrics for Different Classification Schemes

7.1.7 MobileNetV3Small

MobileNetV3Small					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Q0	0.6761	0.6114	0.3700	0.4919	0.6112

Table 17: General Performance Metrics for Different Classification Schemes

MobileNetV3Small								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Q0	0.7373	0.6141	0.5985	0.6046	12548.40	13083.11	7852.89	8387.60

Table 18: Classification Metrics for Different Classification Schemes

7.1.8 MobileNetV3Large

MobileNetV3Large					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
All Features	236.8960	0.0726	0.3716	0.5092	0.6519

Table 19: General Performance Metrics for Different Classification Schemes

MobileNetV3Large								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
All Features	0.5406	0.3640	0.2646	0.0723	33683.10	252831.31	59227.58	93913.98

Table 20: Classification Metrics for Different Classification Schemes

7.2 Transformer Models

7.2.1 Vision Transformer (ViT)

Vision Transformer (ViT)					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Q0	0.2762	0.8890	0.2031	0.2918	0.8890
Q1	0.6038	0.8594	0.4509	0.4534	0.8594
Q2	0.4914	0.8079	0.3673	0.3960	0.8079
Q3	0.5789	0.8653	0.4357	0.4398	0.8653
Q4	0.6306	0.4369	0.4412	0.4686	0.6667
Q5	0.5736	0.9120	0.4343	0.4367	0.9120
Q6	0.6199	0.5127	0.4361	0.4635	0.6667
Q7	0.6657	0.6810	0.4844	0.4861	0.6810
Q8	0.6375	0.4014	0.4445	0.4718	0.6667
Q9	0.4468	0.3417	0.2817	0.3712	0.8333
Q10	0.6803	0.6691	0.4932	0.4935	0.6691
Reduced Set	0.5687	0.7438	0.3994	0.4384	0.7194
All Features	0.5348	0.7475	0.3632	0.4234	0.7334

Table 21: Performance Metrics of the ViT Model Model for Different Classification Schemes

Vision Transformer (ViT)								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Q0	0.9618	0.8890	0.8890	0.8588	18609.7500	18610.1582	2321.8420	2322.2495
Q1	0.8594	0.8594	0.8594	0.4622	18005.1016	18005.1016	2930.8975	2930.8975
Q2	0.8879	0.8079	0.8079	0.7380	16927.0098	16927.0098	4008.9885	4008.9885
Q3	0.8653	0.8653	0.8653	0.4639	18119.4512	18119.4512	2812.5493	2812.5493
Q4	0.6721	0.0000	0.0000	0.2027	0.0000	41872.0000	0.0000	20936.0000
Q5	0.9120	0.9120	0.9120	0.4770	19070.2676	19070.2676	1865.7317	1865.7317
Q6	0.6779	0.0000	0.0000	0.2259	0.0000	41872.0000	0.0000	20936.0000
Q7	0.6810	0.6810	0.6810	0.4051	14241.6895	14241.6895	6694.3091	6694.3091
Q8	0.5190	0.0000	0.0000	0.1910	0.0000	41864.0000	0.0000	20932.0000
Q9	0.6601	0.0000	0.0000	0.0849	0.0000	104660.0000	0.0000	20932.0000
Q10	0.6691	0.6691	0.6691	0.4009	14058.6846	14058.6846	6873.3159	6873.3159
Reduced Set	0.7389	0.7438	0.2722	0.1066	15586.8799	104965.7656	5345.1201	41558.2383
All Features	0.7259	0.7475	0.1227	0.0407	15667.0703	306839.5938	5264.9302	111800.4141

Table 22: Classification Metrics of the ViT Model for Different Classification Schemes

7.2.2 Compact Convolutional Transformer (CCT)

Compact Convolutional Transformer (CCT)					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Reduced Set	0.5660	0.7462	0.3972	0.4372	0.7207

Table 23: Performance Metrics of the CTT Model for the Reduced Set Classification Schemes

Compact Convolutional Transformer (CCT)								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Reduced Set	0.7389	0.7462	0.2737	0.1068	15646.46	105085.14	5289.54	41466.85

Table 24: Classification Metrics of the CTT Model for the Reduced Set Classification Schemes

7.2.3 Convolutional Vision Transformer (CvT)

Convolutional Vision Transformer (CvT)					
Scheme	Loss	Accuracy	Mean ABS Error	RMS Error	Binary Accuracy
Reduced Set	0.5657	0.7458	0.3972	0.4372	0.7201

Table 25: Performance Metrics of the CvT Model for Different Classification Schemes

Convolutional vision Transformer (CvT)								
Scheme	AUC	Precision	Recall	F1	TP	TN	FP	FN
Reduced Set	0.7401	0.7458	0.2731	0.1068	15625.00	105055.30	5311.00	41496.70

Table 26: Classification Metrics of the CvT Model for Different Classification Schemes

8 Discussion

I experimented with changing the projection dimension, number of attention heads, number of transformer units, and MLP units in the MLP head of the ViT model. However, changing these hyperparameters had no significant impact on model performance during preliminary testing and only increased training time. So these models were not fine-tuned and the hyperparameters of a projection dimension of 64, 12 attention heads, 12 transformer units, and 2048×1024 MLP units were chosen. Likewise, changing the number of patches and decreasing the patch size again past 256 patches of size 16×16px did not significantly affect the performance of the ViT model either.

In analyzing the performance of ResNet50 and Vision Transformer (ViT) across various schemes, it is evident that ResNet50 consistently outperforms ViT in terms of accuracy, precision, and recall. This consistent superiority of ResNet50 is particularly notable given the complexities involved in the tasks.

Interestingly, for questions 4, 6, 8, and 9, ViT demonstrates a false positive rate of zero. Upon closer examination, it becomes clear that this is because the Vision Transformer has essentially learned to always predict zero for these features. This behaviour suggests that ViT may be underfitting by never predicting positive outcomes for these particular questions as it cannot learn anything meaingful from the data.

In contrast, ResNet50 shows a different pattern for the same questions. It appears to have learned to predict the average optimal answer, reflecting a more balanced approach even in the face of feature confusion. This is somewhat expected, as the correlation matrices of volunteer responses indicate significant confusion between the features associated with

these questions. ResNet50's ability to navigate this confusion by aiming for average optimal answers might be contributing to its overall better performance compared to ViT.

Thus, while ViT's strategy minimizes false positives to zero for certain features, ResNet50's method of predicting average optimal answers hint at an issue of a local minima of the loss function the optimiser is getting trapped in or the models are simply underfitting and cannot make predictions about these features from the given dataset. However, this result is not completely unexpected and closely matches the volunteer confusion matrices in Figure 19 indicating that the confusion between the volunteer labels for these questions has directly affected the model's ability to learn to identify the associated morphological features.

Given more time, I would experiment more with convolutional transformers, which have shown promise in preliminary tests. The lack of computational resources and time constraints prevented a thorough implementation. Additionally, ensemble methods appear to be a viable strategy, especially considering the insights gained from experimenting with individual question morphological schemes.

Exploring a higher resolution photo dataset could also be an option as there is more information in the images for a model to learn from. Moreover, the application of instance segmentation to isolate only the galaxy being classified could significantly reduce confusion caused by background galaxies, thereby improving a model's predictive capabilities.

Incorporating these changes could address some of the limitations observed in the current models, such as the Vision Transformer's tendency to underfit by predicting zero for certain features to avoid false positives, and the ResNet50's strategy of aiming for average optimal answers. By expanding the scope of experimentation and data quality, there's a potential to overcome the challenges of local minima and underfitting, leading to more robust and accurate models.

Appendices

A CNNs - Additional Information

i Parameters in a Convolutional Layer

The number of parameters in a convolutional layer is given by:

$$N_{param} = (K_x \times K_y \times O_z + 1) \times \ I_z \tag{A-1}$$

Where the addition of 1 is due to the bias

ii Output Shape of Convolutional Layer

For a convolutional layer with N filters the output dimensions are related to the input by:

$$O_x = \frac{I_x - K_x + 2P}{S} + 1, \quad O_y = \frac{I_y - K_y + 2P}{S} + 1, \quad O_z = N \tag{A-2}$$

Where P is the size of the padding, an additional margin added to the edges of the input. "Full padding" is a padding value of $P = K$–1, and "Valid padding" refers to $P = 0$. Full padding ensures that convolutions can be applied to edge pixels as thoroughly as interior pixels, preserving the spatial dimensions during the convolution. The padding can be chosen to increase the output dimension of the convolutional layer, but commonly a stride value and valid padding may be chosen that decreases the output dimension.

iii Pointwise Convolution

Pointwise convolutions (Figure 29) use a $1 \times 1 \times I_z$ kernel. The number of pointwise filters applied will give the same number of output channels in this layer, increasing or decreasing the dimensionality of the input.

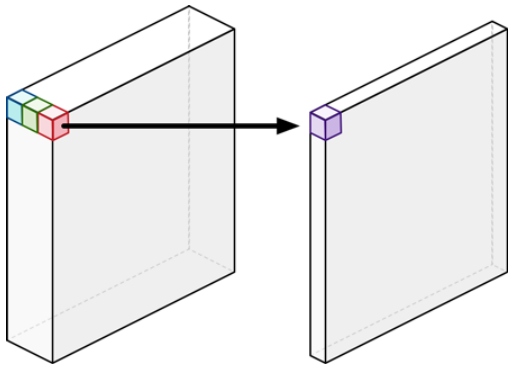


Figure 29: Illustration of Pointwise Convolution [23]

iv Depthwise Convolution

Depthwise convolutions (Figure 30) convolve each channel of the multi-channel input with the respective channel of the filter, concatenating output features rather than summing them, effectively reducing dimensionality.

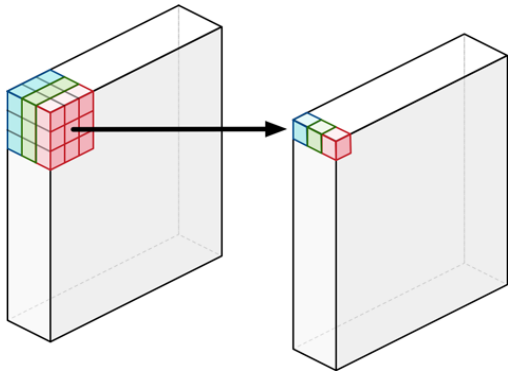


Figure 30: Illustration of Depthwise Convolution [24]

1.4.1 Depthwise Seperable Convolution

The standard convolutional layer can be replaced with 2 layers, a Depthwise convolution followed by a Pointwise convolution with the same depth as the number of filters in a standard convolutional layer, this is the depthwise separable filter (Figure 31). It reduces the number of parameters of the layer and is more computationally efficient.

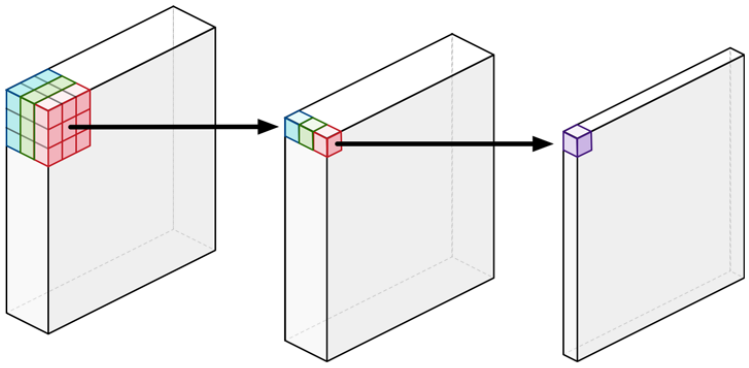


Figure 31: Illustration of Depthwise Separable Convolution [23][24]

v Pooling Layer

The Pooling layer (Figure 32) performs down sampling on the input reducing its dimensionality, the type of pooling is controlled by the kernel. A max pool kernel summarises areas on the input within the kernel to the maximum of those values. The average pool kernel similarly returns the mean value of the input within the kernel.

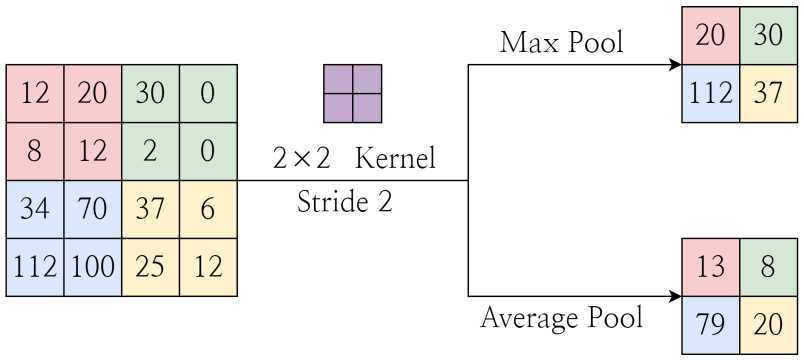


Figure 32: Illustration of Pooling Layers in a Neural Network

B Transfer Learning

Transfer learning repurposes a model developed for one task as the foundation for a model on a related task, using pre-trained models trained on large datasets to capture general features useful across different tasks. This technique reduces the time and computational resources needed for developing new models and improves model performance, especially when data is limited. For instance, a neural network trained on ImageNet, a database of labelled images of over 14 million images, can act as a feature extractor for a new image classification task. The early layers, which detect basic features like edges and textures, remain fixed, while the

later layers are fine-tuned on the new dataset. This allows the model to adapt to the new task's specifics while benefiting from the pre-trained model's general knowledge.

Bibliography

[1] Sloan Digital Sky Survey — classic.sdss.org. `https://classic.sdss.org/home.php`. [Accessed 22-05-2024].

[2] Overview - The Dark Energy Survey — darkenergysurvey.org. `https://www.darkenergysurvey.org/the-des-project/overview/`. [Accessed 22-05-2024].

[3] Zooniverse — zooniverse.org. `https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/`. [Accessed 22-05-2024].

[4] Large Synoptic Survey Telescope. About Rubin Observatory — lsst.org. `https://www.lsst.org/about`. [Accessed 22-05-2024].

[5] Jacek Becla, Andrew Hanushevsky, Sergei Nikolaev, Ghaleb Abdulla, Alexander S. Szalay, María A. Nieto-Santisteban, Ani Thakar, and Jim Gray. Designing a multi-petabyte database for LSST. *CoRR*, abs/cs/0604112, 2006.

[6] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[8] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-dujaili, Ye Duan, Omran Al-Shamma, Jos'e I. Santamar'ia, Mohammed Abdulraheem Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8, 2021.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[15] Joshua Yao-Yu Lin, Song-Mao Liao, Hung-Jin Huang, Wei-Ting Kuo, and Olivia Hsuan-Min Ou. Galaxy morphological classification with efficient vision transformer, 2022.

[16] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers, 2022.

[17] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers, 2021.

[18] Zooniverse — zooniverse.org. `https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/classify`. [Accessed 22-05-2024].

[19] `https://blog.galaxyzoo.org/2015/04/06/visualizing-the-decision-trees-for-galaxy-zoo/` [Accessed 22-05-2024].

[20] Kyle W. Willett, Chris J. Lintott, Steven P. Bamford, Karen L. Masters, Brooke D. Simmons, Kevin R. V. Casteels, Edward M. Edmondson, Lucy F. Fortson, Sugata Kaviraj, William C. Keel, Thomas Melvin, Robert C. Nichol, M. Jordan Raddick, Kevin Schawinski, Robert J. Simpson, Ramin A. Skibba, Arfon M. Smith, and Daniel Thomas. 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):2835–2860, 09 2013.

[21] galaxy-datasets — pypi.org. `https://pypi.org/project/galaxy-datasets/`. [Accessed 22-05-2024].

[22] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *CVPR*, 2019.

[23] Matthijs Hollemans. Regularconvolution@2x.png. `https://machinethink.net/images/mobilenets/RegularConvolution@2x.png`, 2017. [Accessed 22-05-2024].

[24] Matthijs Holleman. Depthwiseconvolution@2x.png. `https://machinethink.net/images/mobilenets/DepthwiseConvolution@2x.png`, 2017. [Accessed 22-05-2024].