Galaxy morphology classification using artificial neural networks

by
Luis Mario Núñez Beltrán
& María Margarita Reyes Sierra

December 10, 2023

Abstract

The classification of galaxies stands as a challenge in astronomical research, pivotal to our understanding of the universe's structure and evolution. Traditionally, this classification has been predominantly manual, relying on visual inspection by astronomers, a process that is not only time-consuming but also prone to human error and inconsistency. With the advent of vast astronomical datasets, the need for automated, accurate, and efficient classification methods has become increasingly pressing. Recent advancements in machine learning, particularly in the field of Convolutional Neural Networks (CNNs), have opened new avenues for addressing this challenge. This project aims to apply the state-of-the-art CNN architecture, EfficientNet, to the task of galaxy classification. EfficientNet, known for its balance of accuracy and computational efficiency, represents a significant leap forward in image recognition technology. By replicating the results of a leading paper in the field, this study seeks to demonstrate the efficacy of EfficientNet in processing astronomical images.

1 Introduction

Galaxy classification is a complex and evolving field of research whose results contribute to our understanding of galaxy evolution and cosmology. It is a systematic approach used in astrophysics to categorize galaxies based on their apparent morphology and structural features. The classification of galaxies provides a foundation to investigate their intrinsic properties, which can be used to learn more about their formation and evolution. As astronomical instruments evolve and amass an unprecedented volume of data, scientists are faced with huge galaxy samples and visual classification of them is becoming impossible.

In recent years, the field of astrophysics has seen a growing intersection with machine learning, as part of this intersection we have the utilization of convolutional neural networks (CNNs) for various tasks. Prior research on galaxy classification has shown the potential of CNNs in achieving high accuracy rates, outperforming traditional methods, and offering scalability for handling vast datasets. This project will specifically delve into the application of CNNs for galaxy classification. Building on the foundational work in both astrophysics and machine learning, our aim is to harness the power of CNNs to categorize galaxies based on their distinct morphological attributes. Our project employs the data from GZ2, which is the cornerstone for the Kaggle challenge: Galaxy Zoo - The Galaxy Challenge.

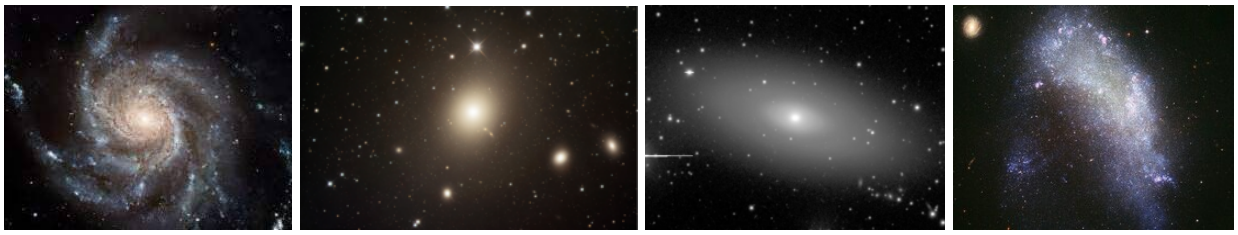
This report is organized as follows. Section 2 describes the problem addressed, and Section 3 discusses the related prior work. Section 4 details the data used in this project, while Section 5 provides insights into the architecture of the CNN employed. Section 6 describes the experiments conducted, and the corresponding results are presented in Section 7. The document concludes with the findings in Section 9.

2 Statement of the problem

Galaxies are most commonly classified according to their shape, brightness, and composition. The most common classification scheme was developed by Edwin Hubble in the 1920s [1]. Hubble divided galaxies into four main types: spirals, ellipticals, lenticulars and irregulars (See Figure 1).

Spiral galaxies have a disk-shaped structure with spiral arms that extend from the center. These galaxies host a range of stellar populations, spanning from young, bluer stars in their spiral arms to older, redder stars in their central bulges. Elliptical galaxies have a smooth, ellipsoidal shape. While they can vary in size and mass, the largest known galaxies are elliptical. They predominantly contain older stars and generally lack the younger, hotter stars characteristic of star-forming galaxies seen in normal spiral galaxies Lenticular galaxies have a transitional form that shares features with both spiral and elliptical galaxies. They have a disk-like structure, similar to spiral galaxies, but lack the prominent spiral arms. In terms of stellar content, they mostly contain older stars and show little to no ongoing star formation. Their appearance is characterized by a smooth, lens-like shape, from which their name is derived. Finally, irregular galaxies can be defined as those whose appearance does not fit into any of the other categories.

The task of manually classifying galaxies comes with a



(a) Spiral galaxy. (b) Elliptical galaxy. (c) Lenticular galaxy. (d) Irregular galaxy.

Figure 1: Main types of galaxies, as categorized by Edwin Hubble and subsequent observations.

set of challenges, among the main ones are:

- The volume of observable galaxies. With the advent of advanced telescopes, the number of galaxies that can be observed and recorded has grown exponentially. Manually classifying these large galaxy samples is becoming an untenable endeavor.
- The subjective nature of galaxy classification by observers can lead to inconsistencies; two different observers might classify the same galaxy differently based on their interpretations of its features.
- The quality of the observational data. Distant galaxies can be challenging to classify accurately, as their full morphology might not be discernible.

For theses reasons, it has then become imperative to develop automated and quantitative approaches for galaxy classification. Leveraging algorithms and machine learning, these systems can efficiently process vast datasets to classify galaxies, often outpacing and outperforming manual efforts.

3 Related Work

Initial efforts to apply machine learning in galaxy classification trace back to the mid-1990s.

In 1995, Naim et al. [2] trained a multi-layer perceptron neural network with a single hidden layer, using morphological features (like ellipticity and bulge size) derived from scanned photographic plates in the B-band for a sample of 800 galaxies observed via a 48-inch UK Schmidt telescope in Siding Spring, Australia. We suggest readers compare the complexity of these early machine learning models, the size of their training samples, and their observational methodologies with those used in contemporary research.

In a study conducted in 2001, Bazell & Aha [3] demonstrated how combining simple classifiers, including Naive Bayes, decision trees, and multi-layer perceptrons, could enhance the accuracy of a compound model.

In 2010, Gauci et al. [4] used random forests to classify galaxies within the Galaxy Zoo 1 dataset into categories such as spiral, elliptical, or star/unknown objects. Their research concluded that random forests are effective in galaxy classification, showing promise for managing extensive astronomical datasets and potentially equating the classification precision of human experts.

The field of galaxy classification via machine learning saw a significant boost in late 2013 with the inception of the Kaggle Galaxy Zoo challenge. This competition tasked participants with creating the most accurate machine learning model to predict the classifications assigned to galaxy images from the Sloan Digital Sky Survey (SDSS) by Galaxy Zoo users. This challenge, more a regression than a classification task, concluded in mid-2014 with a prize of 16,000 USD awarded to the creator(s) of the model with the lowest classification error. The winner of this challenge was UK researcher Sander Dieleman. His winning solution was an ensemble average of 17 convolutional neural networks, each with approximately 40 million trainable parameters.

Kalvankar et al. [5] delved into the capabilities of the state-of-the-art general-purpose image classification network, EfficientNet, in the context of the Kaggle Galaxy Zoo challenge. The objective of this project is to examine and replicate the results presented in their research.

4 Data Description

4.1 Galaxy Zoo: The Project

In July 2007, astronomers from Oxford University had in their possession a data set of ~ 1 million galaxies imaged by the Sloan Digital Sky Survey (SDSS). The galaxies in this data set needed to have their visual morphologies classified. With so many galaxies, it would have taken an individual a thousand lifetimes to classify all of them. Instead, Galaxy Zoo was born.

The Galaxy Zoo project is an ongoing citizen science initiative that enlists the help of the general public to clas-

sify galaxies based on their visual morphology. It's one of the pioneering projects in the field of citizen science, where non-professionals collaborate with scientists on real scientific research. Online participants are presented with galaxy images and are asked to classify them based on their shapes. The basic categories included spiral, elliptical, and irregular.

The project was launched in July 2007. Its initial goal was to classify about one million images of galaxies from the SDSS. The response was overwhelming: It was initially assumed that despite outsourcing the work to thousands in the general public, it would still take years for all of the images to be classified. Within the first 24 hours of launch, Galaxy Zoo founders were stunned to be receiving nearly 70,000 classifications an hour. In the end, more than 50 million classifications were received by the project during its first year, contributed by more than 150,000 people.

Given the success of the first project, Galaxy Zoo 2 (GZ2) was launched in February 2009. GZ2 aimed for more detailed morphological classifications; users were asked to about bars, the shape of the galaxy's core, the presence of spiral arms, their number, and more. The project continued to use the SDSS, but focused on a subset of the most well-resolved images that had already been classified in GZ1 (this often implies that the galaxies in question are at lower redshifts).

This project employs the data from GZ2, which is the cornerstone for the Kaggle challenge: Galaxy Zoo - The Galaxy Challenge. The data we are provided with are:

- **Training images:** Around 62,000 JPG images, which have been extracted from FITS frames showcasing galaxies as observed by the SDSS. Each training image is accompanied by a probability distribution (ground-truth vector of labels), grounded in human visual classifications.
- **Testing images:** Close to 80,000 JPG images, similarly derived from FITS frames of galaxies captured by the SDSS. Our task is to assign probability distributions for each of these images.

4.2 Image acquisition procedure

It is worth repeating that the source of the JPG images in the GZ2 dataset is the SDSS. The primary mirror of the SDSS telescope at Apache Point Observatory (New Mexico, USA) is 2.5 meters in diameter (Figure 3).

The SDSS imaging camera is designed to capture light in five different optical bands (u, g, r, i, and z). The distinct rows of CCDs in the camera are equipped with filters specific to each of these bands, allowing the camera to simultaneously observe the sky in these bands as the telescope scans across it. The gaps or spaces between these CCD blocks are inherent to the camera's design. When the telescope scans the sky, it moves in a way that ensures the gaps are covered in subsequent observations, so no part of the sky is missed. The imaging camera of the SDSS is composed of 30 CCDs, each of these CCDs contains 2048 $\times$ 2048 pixels.

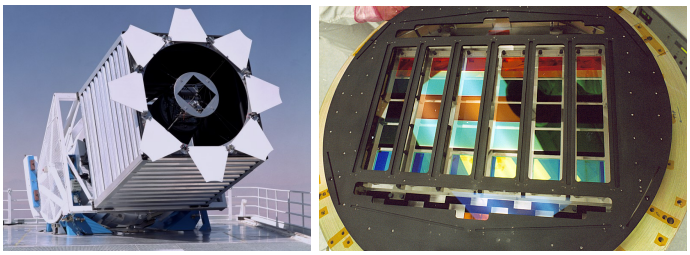


Figure 3: Left: The telescope used to compile the SDSS. Right: The CCD array employed by the SDSS telescope in imaging tasks.

When the telescope observes a given galaxy, the result is five images (one per filter) in FITS format, similar to the ones in Figure 2. Those grayscale images indicate the amount of light the CCD captured in each pixel.

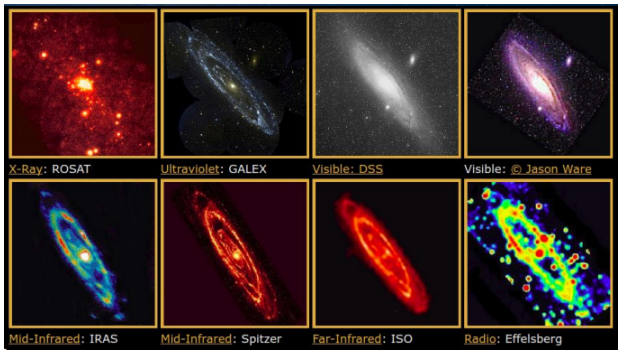


Figure 4: The Andromeda galaxy (M31) observed in different wavelenghts.

One of the benefits of observing a galaxy in different wavelengths is that each one delivers different information. This can be seen from the different appearances of the galaxy in Figure 2 across different filters. A particularly striking example is shown in Figure 4 wherein the Andromeda galaxy (M31) is mapped from X-rays to radio waves.

For each galaxy in the GZ2 sample, three of those FITS images; the g, r, and i bands, were combined by the SDSS to generate 424 $\times$ 424 RGB-color JPG images. Figure 5 exemplifies this.

For this project, we will be utilizing the JPG images provided by the SDSS as our data set, as opposed to the

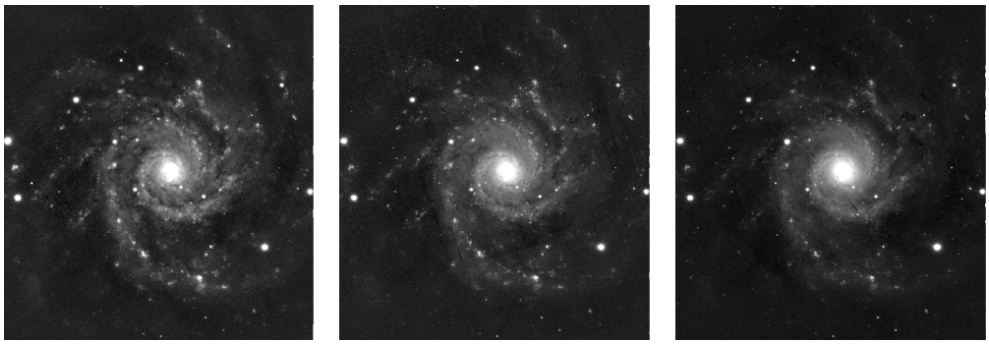


Figure 2: The same spiral galaxy observed with three different wavelength-filters; g (left), r (center), and i (right)

FITS files from which they were composed. The rationale for this decision is multifold, but it is primarily driven by the time-consuming nature of the data reduction process required to work with FITS files, in comparison to a more accessible format like JPG.

Figure 5: Three galaxies in the GZ2 sample.

The creation of the JPG images from raw FITS files from an astronomical survey is not a straightforward task; it involves several intricate steps, each of which is time-intensive and computationally demanding. Some examples of the data reduction processes include:

- Calibration: This involves correcting the raw FITS images for any distortions or biases introduced by the imaging equipment. It typically requires the subtraction of dark frames to remove dark current noise, and flat-fielding to correct for variations in the sensitivity of the detector pixels.
- Background estimation and subtraction: FITS images often contain background noise from the sky or instrumental sources, which must be carefully subtracted to enhance the visibility of the galaxy itself.
- Intensity Scaling and Color Mapping: The dynamic range of FITS images is typically much greater than can be displayed in a standard JPEG image. Converting the high dynamic range data into a visual format involves scaling the intensity levels and mapping them to color channels, which is a complex process that often requires manual tuning to produce aesthetically pleasing and scientifically informative images.

Given the extensive nature of these tasks, and considering the constraints of time and resources inherent to a class project, we have decided to use the pre-processed JPG images. That decision allows us to focus on the deep learning aspect of our project rather than the preprocessing of the images.

The galaxy features that are pertinent to our project's objectives — such as spiral arm structure, bar classification, and the presence of galactic bulges — remain discernible and analyzable within the JPG format.

Nevertheless, to foster a better understanding of the procedures involved in data reduction, we have included an appendix detailing some of the most common data reduction steps for raw FITS files.

4.3 Ground-truth labels

Figure 6 contains the classification procedure that resulted in the ground-truth labels for the training data. That is, for each image, the volunteer was asked to go through these 11 "tasks", each task consists of assigning a response to the corresponding questions.

The weighted voting system used in the Galaxy Zoo project to classify galaxies:

- **Initial Classification:**

  Users classify galaxies into primary categories (smooth, features/disk, star/artifact). The results give initial probabilities for each category. These probabilities sum to 1.0 for each galaxy since each galaxy is classified into one of these primary categories by every user.

  For example, if for a given galaxy:

- 80% of users think the galaxy is smooth, then Class1.1 = 0.80.

- 15% think it has features/disk, then Class1.2 = 0.15.

- 5% think it's a star or artifact, then Class1.3 = 0.05.

therefore, the first three entries of the ground-truth vector for that galaxy will be (0.80, 0.15, 0.05,....).

Task	Question	Responses	Next
01	*Is the galaxy simply smooth and rounded, with no sign of a disk?*	smooth features or disk star or artifact	07 02 **end**
02	*Could this be a disk viewed edge-on?*	yes no	09 03
03	*Is there a sign of a bar feature through the centre of the galaxy?*	yes no	04 04
04	*Is there any sign of a spiral arm pattern?*	yes no	10 05
05	*How prominent is the central bulge, compared with the rest of the galaxy?*	no bulge just noticeable obvious dominant	06 06 06 06
06	*Is there anything odd?*	yes no	08 **end**
07	*How rounded is it?*	completely round in between cigar-shaped	06 06 06
08	*Is the odd feature a ring, or is the galaxy disturbed or irregular?*	ring lens or arc disturbed irregular other merger dust lane	**end** **end** **end** **end** **end** **end** **end**
09	*Does the galaxy have a bulge at its centre? If so, what shape?*	rounded boxy no bulge	06 06 06
10	*How tightly wound do the spiral arms appear?*	tight medium loose	11 11 11
11	*How many spiral arms are there?*	1 2 3 4 more than four can't tell	05 05 05 05 05 05

Figure 6: The GZ2 decision tree, comprising 11 tasks and 37 responses. The "task" number is an abbreviation only and does not necessarily represent the order of the task within the decision tree. The text "Question" and "Responses" are displayed to volunteers during classification. "Next" gives the subsequent task for the chosen response.

- **Subsequent Classification:**

  For each primary classification (like "smooth"), additional questions pertain to more specific morphological features. The probabilities derived from these questions are multiplied by the probability of the initial classification to determine the probability for that sub-classification.

  Continuing with the example above:

  - Of the 80% who thought it is smooth: If 50% think it's completely round, then Class7.1 = $0.80 \times 0.50 = 0.40$.

  - If 25% think it's in-between, then Class7.2 = $0.80 \times 0.25 = 0.20$.

  - If 25% think it's cigar-shaped, then Class7.3 = $0.80 \times 0.25 = 0.20$.

  Thus, the ground-truth vector for this galaxy will also contain those entries in their corresponding indices: (..., 0.40, 0.20, 0.20,...).

  For each galaxy, the result is a vector of ground-truth labels with 37 entries. The goal of the Kaggle challenge is to build a deep-learning classifier capable of predicting the ground-truth vector for the galaxies in the test set.

5 EfficientNet family

In this project, we exploit the success of the EfficientNet family of CNNs to tackle the Kaggle Galaxy Zoo challenge. We start with a brief introduction to the EfficientNet-B0 architecture.

5.1 EfficientNet-B0 Architecture

Inspired by Tan et al. [6], Tan & Le [7] developed a baseline network, called EfficientNet-B0, using a Neural Architecture Search (NAS) that optimizes both Accuracy and FLOPS (FLoating point Operations Per Second), setting:

$$\mathrm{ACC}(m) \times \left(\frac{\mathrm{FLOPS}(m)}{T}\right)^{w}$$

as the optimization goal, where $\mathrm{ACC}(m)$ and $\mathrm{FLOPS}(m)$ denote the accuracy and FLOPS of model m, T is the target FLOPS (maximum allowed) and $w = -0.07$ is a hyperparameter for controlling the trade-off between accuracy and FLOPS. This specific value is chosen based on empirical results. The goal of the NAS in this context is to find a model that achieves high accuracy while also being computationally efficient. The hyperparameter w helps balance these two objectives, which are often in conflict; generally, higher accuracy can be achieved with more complex models that require more FLOPS, which is not desirable for mobile or resource-constrained environments.

A negative value for w means that as the FLOPS for a model m increase, the term $\left(\frac{\mathrm{FLOPS}(m)}{T}\right)^{w}$ decreases, assuming $\frac{\mathrm{FLOPS}(m)}{T} > 1$. This effectively penalizes models with higher FLOPS, steering the search towards more efficient

architectures; keep in mind that optimal models are those that minimize the product: $\text{ACC}(m) \times \left(\frac{\text{FLOPS}(m)}{T}\right)^{w}$.

Table 1 and Figure 7 show the architecture of EfficientNet-B0. As can be seen, the MBConv block (Mobile Inverted Bottleneck Convolution) is a cornerstone of the EfficientNet architecture.

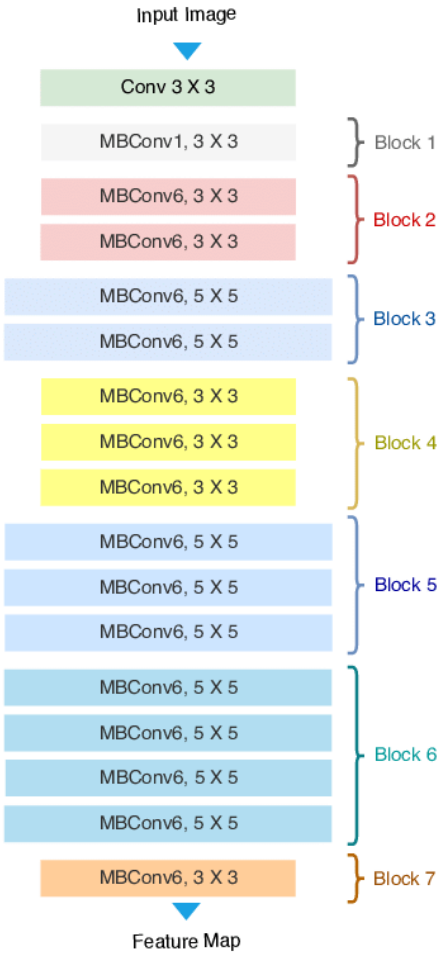


Figure 7: EfficientNet-B0 Architecture.

Table 1: EfficientNet-B0 baseline network. Each row describes a stage with layers, input resolution, and output channels.

Stage	Operator	Resolution	Channels	Layers
1	Conv3x3	224 x 224	32	1
2	MBConv1, 3x3	112 x 112	16	1
3	MBConv6, 3x3	112 x 112	24	2
4	MBConv6, 5x5	56 x 56	40	2
5	MBConv6, 5x5	28 x 28	80	3
6	MBConv6, 5x5	14 x 14	112	3
7	MBConv6, 5x5	14 x 14	192	4
8	MBConv6, 3x3	7 x 7	320	1
9	Conv1x1 & Pooling & FC	7 x 7	1280	1

MBConv block

.

The diagram in Figure 8 shows the structure of an MBConv block.

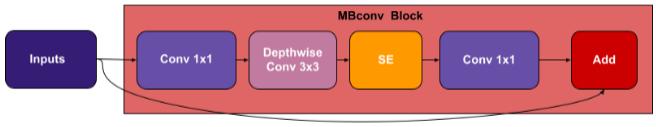


Figure 8: MBConv block.

1. Inputs: This is the data that is fed into the MBConv block, typically a feature map generated from the previous layer of the network.

2. Conv 1×1: The first layer within the MBConv block is a 1×1 convolution, also known as a pointwise convolution. Its main role is to combine the features from the previous layer's feature map into a new set, which can increase or decrease the number of channels without changing the spatial dimensions (Figure 9).

   If a feature map has a shape of $m \times n \times f_1$. A 1×1 convolution is performed with a vector of 1D length f_1 which convolves across the whole image, creating one $m \times n$ output filter. If we had f_2 1×1 convolutions, then the output of all of them is a matrix with size $m \times n \times f_2$. So a 1×1 convolution, assuming $f_2 < f_1$, can be seen as representing f_1 filters via f_2 filters.

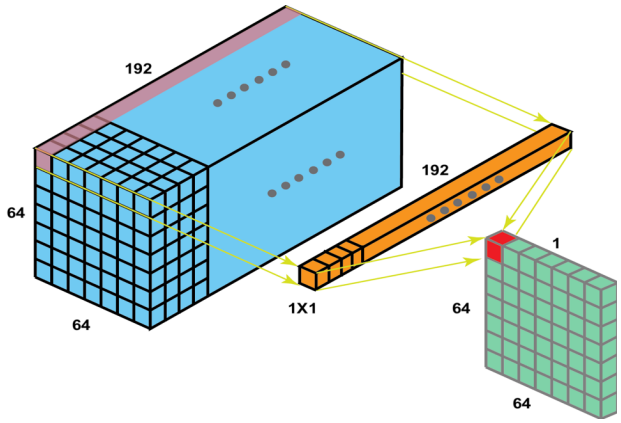


Figure 9: Pointwise convolution.

3. Depthwise Conv 3×3: Unlike standard convolutions that mix inputs from all channels, depthwise convolutions apply a single filter per input channel. In this case, it's a 3×3 kernel. This step is crucial for spatial feature extraction and is more computationally efficient than a full convolution because it separates the

spatial filtering from the channel combination. Compare depth-wise convolutions (Figure 10) with normal convolutions (Figure 11).

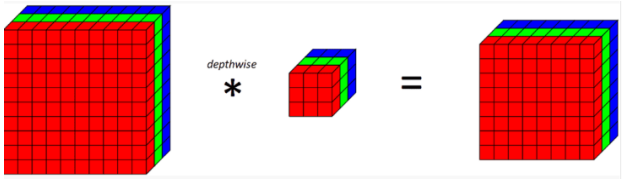


Figure 10: Depthwise convolution

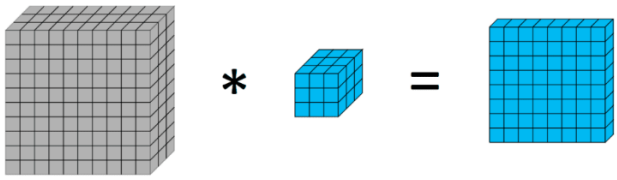


Figure 11: Normal convolution

4. SE (Squeeze-and-Excitation): MBConv blocks include a SE block to recalibrate the channel-wise feature responses by explicitly modeling interdependencies between channels. Within an MBConv block, the SE block is applied after the depthwise convolution and before collapsing the channels back down, providing a mechanism for the network to emphasize informative features and suppress less useful ones. More on this later.

5. Conv 1×1: Another pointwise convolution follows the SE block. This layer typically expands the number of channels again and projects the feature maps back to a higher-dimensional space.

6. Add: Finally, the output of the last 1×1 convolution is combined with the original input through an element-wise addition, which is possible if the dimensions match (if not, the input is usually adjusted with a 1×1 convolution to match the output's size). This step is a form of a skip connection, similar to those used in ResNet architectures [8], and helps mitigate the vanishing gradient problem, allowing for deeper networks.

The "1" in MBConv1 and the "6" in MBConv6 denote the expansion factor used:

- MBConv1: The "1" indicates that there is no expansion in the number of channels before the depthwise convolution. This means that the number of input channels to the block is the same as the number of channels that are expanded internally before the depthwise convolution. Essentially, the block is a "bottleneck" with no expansion, and the internal channel dimension is equal to the input channel dimension.

- MBConv6: The "6" signifies that the number of input channels is expanded by a factor of 6 before the depth-wise convolution. If the block receives an input with 'C' channels, inside the block, the channels will be expanded to '6C' before the depth-wise convolution is applied. This expansion allows the network to create a richer set of features at this point in the architecture.

  The expansion in the number of channels is performed with the point-wise convolution.

Inverted residual structure

The term "inverted" in Mobile Inverted Bottleneck Convolution (MBConv) refers to the structure of the convolutional block, which is the opposite of what was traditionally used in ConvNets.

- Traditional Bottleneck: In traditional bottleneck convolutional blocks, often used in deep learning models, the sequence starts with a high-dimensional input which is first compressed to a lower dimension using a 1×1 convolution (pointwise convolution), then processed through a 3×3 convolution, and finally expanded back to a high dimension with another 1×1 convolution (so, it resembles a bottleneck). This "bottleneck" design reduces the number of input channels (and thus the computational complexity) for the 3×3 convolution (See Figure 12 (a)).

- Inverted Bottleneck: The MBConv block inverts this process. Instead of starting with compression, the block begins by expanding the number of channels with a 1×1 convolution. It then applies depthwise convolution to the expanded channels, which is computationally efficient since it applies a single kernel per input channel. The final 1×1 convolution combines the expressive features back into a lower-dimensional space. (See Figure 12 (b))

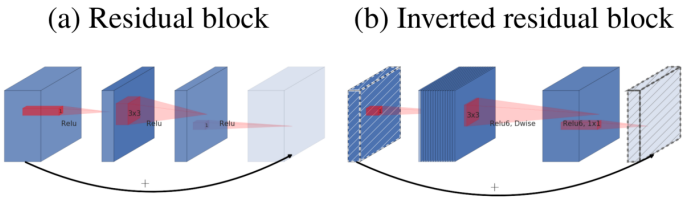


Figure 12: Normal and Inverted residual blocks

This design is a key element of the EfficientNet architecture, as it allows for maintaining the network's efficiency while increasing its capacity to learn complex features. The "inverted" aspect makes the MBConv block particularly well-suited for mobile and other edge devices where computational resources are limited.

SE (*Squeeze-Excitation*) Blocks

A "Squeeze and Excitation (SE) block" is a component used in neural network architectures to improve their performance by explicitly modelling the interdependencies between channels. Figure 13 shows the general architecture of a SE block.

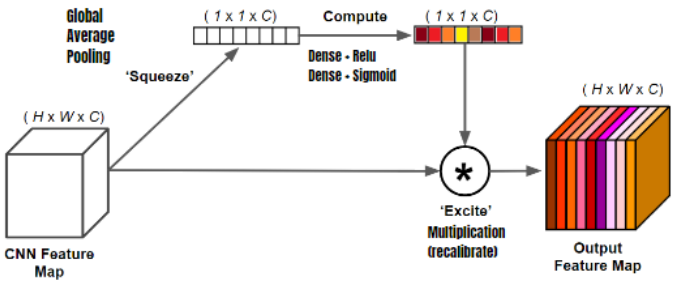


Figure 13: Squeeze-Excitation Block.

1. Global Average Pooling: The global average pooling operation takes the feature map and computes the average of each feature channel, reducing its spatial dimensions to 1x1. This process is meant to "squeeze" the global spatial information into a vector.

2. Dense Layer + ReLU: The vector we got from the previous step goes now through a dense (fully connected) layer which further compresses the former. A ReLU activation function is then applied to this smaller vector.

3. Dense Layer + Sigmoid: Another fully connected layer follows the ReLU activation. This layer serves to scale up the dimensionality of the feature representation to its original size. A sigmoid activation function is then applied after the second dense layer, which outputs values between 0 and 1. This will serve to weight the channels according to their importance.

4. Scale ("Excite"): This step involves the element-wise multiplication of the recalibrated channel weights with the original feature map from the convolutional layer. It allows the network to emphasize informative features and suppress less useful ones.

Each of these elements contributes to the SE block's ability to perform dynamic channel-wise feature recalibration, which can lead to improved performance in deep learning models, especially for tasks involving image recognition.

5.2 Compound Scaling

Scaling up ConvNets is a widely used method to boost their predictive accuracy. Scaling up a ConvNet typically means increasing the "size" of the network to make it more capable of handling complex tasks. This can be done in various ways, such as adding more layers (i.e. increasing the depth of the network), increasing the number of units or kernels in each layer (i.e. increasing the width of the network), or using higher resolution input images (i.e. increasing the resolution of the network).

In most works, it is common to scale only one of the three dimensions – depth, width, and image size for a given ConvNet. The most common way to scale up ConvNets is by increasing their depth. For example, ResNets [8] exist in various sizes, indicated by the number in their names, as such, ResNet-18 has 18 layers, while ResNet-200 has 200 layers. Scaling up ResNet from ResNet-18 to ResNet-200 means increasing the number of layers in the network. Each additional layer allows the network to learn more complex features and patterns in the data, potentially leading to higher accuracy in image recognition tasks.

Tan & Le [7] investigated the following question: is there a systematic and possibly simple method to scale up ConvNets that can achieve better accuracy and efficiency? They were the first ones to show that ConvNets can achieve better performance by changing the three scaling dimensions (width/depth/resolution) simultaneously. We next expose the problem Tan & Le tackled, and their strategy to do so.

Let us remind the reader that a ConvNet Layer i is as a function: $Y_i = F_i(X_i)$, where F_i is the "operator" in the layer (i.e. F_i encapsulates convolution operations, activation, pooling,...), Y_i is the output tensor, and X_i is the input tensor, the latter with shape (H_i, W_i, C_i), where H_i and W_i are spatial dimensions and C_i is the channel dimension. A

ConvNet N can be represented by a composition of layers:

$$N = F_k \circ \cdots \circ F_2 \circ F_1(X_1) = \bigodot_{j=1}^{k} F_j(X_1)$$

In practice, ConvNet layers are often partitioned into multiple stages and all layers in each stage share the same architecture: for example, ResNet [8] has five stages, and all layers in each stage have the same convolutional type with the exception of the first layer which performs down-sampling (Figure 14).

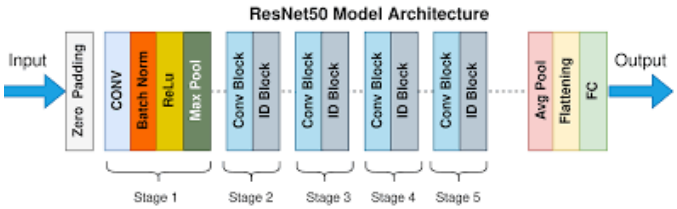


Figure 14: In practice, ConvNet layers are often partitioned into multiple stages and all layers in each stage share the same architecture.

Therefore, we can define a ConvNet as:

$$N = \bigodot_{i=1}^{s} F_i^{L_i}(X_{(H_i, W_i, C_i)})$$

where $F_i^{L_i}$ denotes that layer F_i is repeated L_i times at stage i, (H_i, W_i, C_i) denotes the shape of the input tensor X.

Unlike regular ConvNet designs that mostly focus on finding the best layer architecture F_i, model scaling tries to expand the network length L_i, width C_i, and/or resolution (H_i, W_i) without changing F_i which is predefined by a baseline network. By fixing F_i, the model scaling problem is simplified, but the design space to be explored is still large since it is necessary to test out different L_i, C_i, H_i, W_i values for each stage.

In order to further reduce the design space, Tan & Le restricted all layers to be scaled uniformly so that the target of maximizing the model accuracy for any given resource constraints, can be formulated as the following optimization problem:

$$\max_{d,w,r} \quad \text{Accuracy}(N(d,w,r))$$

$$\text{s.t.} \quad N(d,w,r) = \bigodot_{i=1}^{s} F_i^{d \cdot L_i}\left(X_{(r \cdot H_i, r \cdot W_i, w \cdot C_i)}\right)$$

$$\text{Memory}(N) \leq \text{target\_memory}$$

$$\text{FLOPS}(N) \leq \text{target\_flops}$$

where w, d, r are coefficients for scaling network width, depth, and resolution, respectively (F_i, L_i, H_i, W_i, C_i are predefined parameters in the baseline ConvNet).

- **Depth (d)**
  As mentioned before, scaling network depth (d) is the most common scaling method. The idea behind it is that deeper ConvNet can potentially capture more complex features. However, deeper networks are also more difficult to train due to the vanishing gradient problem. Although several techniques, such as skip (residual) connections [8] and batch normalization [9], ameliorate the training problem, the accuracy gain of very deep networks still gets diminished: for example, ResNet-1000 has similar accuracy as ResNet-101.

- **Width (w)**
  Similarly, extremely wide (w) but shallow networks tend to have difficulties in capturing higher-level features. For example, a network that is wide (having many neurons in each layer) but shallow (having few layers) can be good at capturing a broad array of simple, lower-level features due to its width. For instance, in an image of a face, lower-level features include the lines that form the outline of the eyes, nose, and mouth. However, since the network lacks depth, these lower-level features may not be adequately combined into more complex ones since the depth in a network (more layers) allows for the hierarchical processing of features, where simple features detected in early layers are combined and abstracted in deeper layers, continuing with the face example, higher-level features might include the recognition of a nose or an eye as a whole or even the recognition of facial expressions like smiling or frowning.

- **Resolution (r)**
  Clearly, with higher-resolution input images, ConvNets can potentially capture more fine-grained patterns. However, increasing the size of an input image does not necessarily imply an increase in resolution:

  - Resolution refers to the density of pixels in an image, usually expressed in terms of pixels per inch (PPI). It's a measure of the detail an image holds.
  - A higher-resolution image contains more pixels in the same amount of space, providing more detail and clarity.

  When you resize a smaller image to a larger size without adding new information, you're essentially stretching the existing pixels over a larger area. This does not increase the resolution.

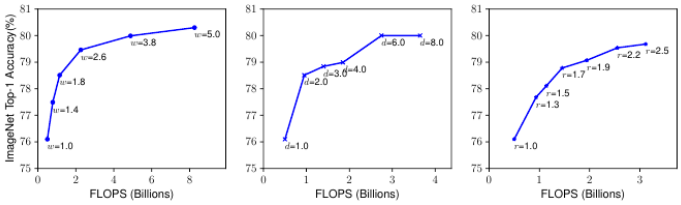


Figure 15: Scaling up a baseline model with different network width (w), depth (d), and resolution (r) coefficients.

Figure 15 shows the scaling up a baseline model with different network width (w), depth (d), and resolution (r) coefficients. Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single-dimension scaling.

In this way, Tan & Le empirically observed that different scaling dimensions are not independent, and made the following observations:

1. Scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models.

2. In order to pursue better accuracy and efficiency, it is necessary to balance all dimensions of network width, depth, and resolution during ConvNet scaling.

As a result of these observations, Tan & Le proposed what they call a **compound scaling method** [7], which uses a coefficient ϕ to uniformly scale network width, depth, and resolution in the following way:

- depth: $d = \alpha^\phi$
- width: $w = \beta^\phi$
- resolution: $r = \gamma^\phi$

subject to the constraints:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

where α, β, γ are constants determined by a small grid search. Intuitively, ϕ is a user-specified coefficient that controls how many more resources are available for the model, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution respectively. According to [7], the FLOPS of a regular convolution operation is proportional to d, w^2, r^2, i.e., doubling network depth will double FLOPS, but doubling network width or resolution will quadruple FLOPS. Since convolution operations usually dominate the computation cost in ConvNets, scaling a ConvNet with equation $w \cdot d \cdot r^2$ will approximately increase total FLOPS by $(w \cdot d \cdot r^2)^\phi$. Hence, the constraint $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ is such that for any new ϕ, the total FLOPS will approximately increase by 2^ϕ.

Starting from the baseline EfficientNet-B0, Tan & Le applied their compound scaling method to scale it up with two steps [7]:

1. They first fix $\phi = 1$, and do a small grid search of α, β and γ. They find the best values for EfficientNet-B0 are $\alpha = 1.2$, $\beta = 1.1$ and $\gamma = 1.15$ under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.

2. They then fix α, β and γ as constants and scale up baseline network with different ϕ, to obtain EfficientNet-B1 to B7.

6 Network architecture and implementation details

In [5], the authors use the EfficientNet architecture and introduce a tail part that helps fine-tune the model to the task at hand. This tail part takes the output of the EfficientNet and introduces a Global Average Pooling which downsamples the output of every dimension and flattens the tensor. A dropout is introduced with a rate of 0.5 to reduce overfitting. The output is then passed onto a fully connected layer with 64 connections and ReLU activation. The output layer consists of 37 units which use a sigmoid activation. Figure 16 shows a brief overview of how the network architecture is modelled for the tail part.

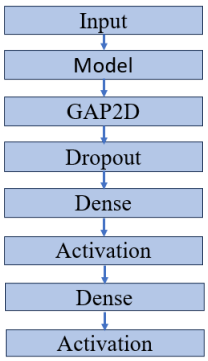


Figure 16: Network architecture of the tail part of the fine-tuned model.

6.1 Data preprocessing: Image cropping

The images provided by Kaggle for their Galaxy Zoo challenge include a rather large portion of the sky around the galaxy the image corresponds to. The following are several reasons why we cropped each image to isolate the target galaxy:

- Enhanced Feature Focus: In a large field view, the galaxy of interest might be surrounded by other celestial objects or background noise. Cropping the image to center on the galaxy allows the model to focus on the relevant features of the galaxy itself.
- Mitigating Telescopic Artifacts: Telescopic images often include artifacts like cosmic ray detections or edge distortions. Cropping the image to focus on the galaxy can minimize the influence of these artifacts on the model's training process.
- Model Efficiency: Training models on large images with lots of irrelevant data can be computationally expensive and inefficient. Cropping reduces the image size, leading to faster training times and lower computational resource requirements.

Kalvankar et al. [5] divided each image into 16 equal parts. The image is then cropped to a window of the central four sections and the remaining 12 sections are discarded as noise, see Figure 17.

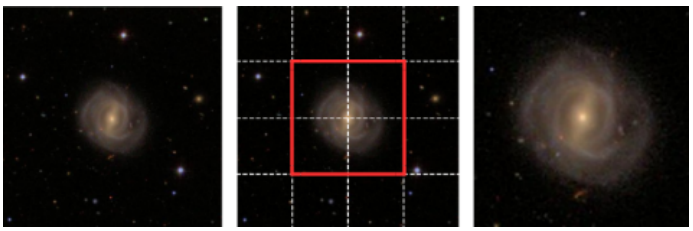


Figure 17: Cropping images

6.2 Data augmentation

Samples of astronomical objects are often biased due to observational and instrumental limitations. The Galaxy Zoo 2 galaxy sample was selected as a subset of the Galaxy Zoo 1 galaxy sample, the subset was chosen so it contains only the nearest, brightest, and largest systems for which fine morphological features can be resolved and classified. This selection criteria introduces, among others, these biases:

- Size Bias: By choosing larger galaxies, the sample may be biased against smaller galaxies that might be just as close. This could lead to an underrepresentation of certain morphological features that are more common in smaller galaxies.
- Luminosity Bias: Brighter galaxies are not necessarily representative of the general galaxy population. This bias could cause the model to perform poorly when classifying galaxies of average or low brightness, which are more numerous in the universe.
- Distance Bias: Focusing on the nearest galaxies could skew the sample towards local galaxy types and away from those that are more common at greater distances. This may impact the model's ability to generalize to galaxies at higher redshifts.
- Morphological Feature Bias: The selection could also prefer galaxies with well-defined morphological features that are easier to classify, potentially overlooking galaxies with subtle or uncommon features.
- Spectral Bias: There might be a bias towards certain types of galaxies that have strong spectral features, which are easier to detect and classify spectroscopically.

These biases can lead to a model that is overfitted to the specific characteristics of the GZ2 sample and that does not perform well on a more varied set of galaxies.

Standard data augmentation techniques are not particularly useful in reducing most of these observational biases. For example, merging galaxies present complex and varied features that are difficult to simulate through standard data augmentation techniques such as rotation, flipping, or scaling (see Figure 18). The same can said of starburst, or active galactic nuclei.

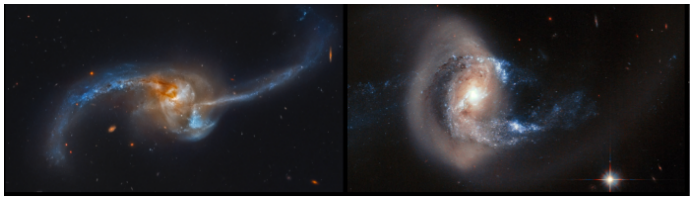


Figure 18: Merging galaxies

Nonetheless, data augmentation was generally recommended to be taken into account by competitors in the Kaggle Galaxy Zoo challenge. According to Sander Dieleman, the winner of the competition: *"My main objective during the competition was avoiding overfitting. My models were significantly overfitting and most of the progress I attained came from finding new ways to mitigate that problem."*

Kalvankar et al. [5], proposed the data augmentation pipeline outlined in Figure 19. The main goal for the data

augmentation procedure is, of course, to increase the robustness to variations in orientation, size, and brightness of the input galaxies.

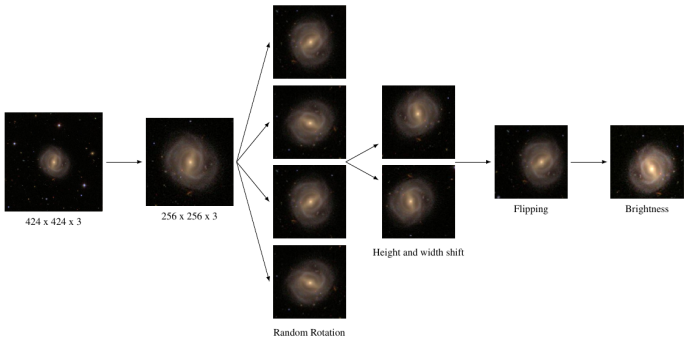


Figure 19: Data augmentation pipeline

6.3 ImageNet

ImageNet is one of the largest and most diverse datasets available for image recognition. It contains over 14 million images categorized into more than 20,000 classes[1]. This extensive range provides a robust platform for training image recognition models.

ImageNet has become a benchmark in the field of computer vision. High performance on ImageNet is often correlated with high performance in other visual recognition tasks. Many successful models pre-trained on ImageNet are available. These models can be fine-tuned for specific tasks, saving time and hardware resources. Models trained on ImageNet have shown great ability in transfer learning. This means that the knowledge gained from ImageNet can be transferred to other domains or tasks with minimal additional training. This makes models trained on ImageNet versatile and adaptable.

Figure 20: The ImageNet database plays a crucial role in the field of computer vision and image recognition

Figure 22 shows that the EfficientNet family of CNNs achieves state-of-the-art accuracy on the ImageNet database [7]. In particular, the most complex member of the EfficientNet family: EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet, while being 8.4x smaller and 6.1x faster on inference than heavier ConvNets like GPipe (Huang et al., 2018) which pushed the state-of-the-art ImageNet top-1 validation accuracy to 84.3% using 557M parameters: training such a large model presents significant challenges in terms of hardware requirements; the network it is so big that it can only be trained with a specialized pipeline parallelism library by partitioning the network.

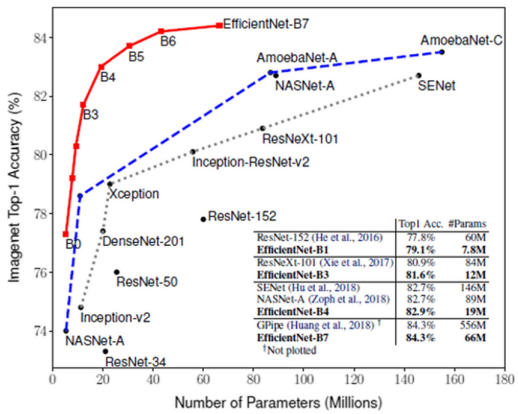


Figure 21: EfficientNet models have achieved state-of-the-art accuracy on ImageNet, outperforming many other architectures. For example, EfficientNet-B7 has reached a top-1 accuracy of 84.3% on ImageNet, which is a remarkable achievement.

6.4 Training procedure and results

The training dataset for the Kaggle Galaxy Zoo challenge consists of 61,578 images. To facilitate supervised learning, we allocated 15% of this dataset (9,236 images) for validation, leaving 52,342 images for training. The model training process involves batch processing (see Section 6.2) 70 images at a time, without shuffling, directly from their storage directory. This setup means that the model's learnable parameters are updated approximately 850 times per epoch.

To evaluate our model's performance on both the training and validation datasets, we employed the Root Mean Square Error (RMSE) metric as defined in Section 6.5. The model's parameters were iteratively refined using an Adam

[1]ImageNet is often associated with just 1,000 classes primarily due to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which popularized the dataset within the machine learning and computer vision communities. This annual competition focused on a subset of ImageNet with 1,000 classes. The challenge was to classify images correctly into these classes, and it quickly became a benchmark in the field. The 1,000-class subset provided a manageable yet challenging benchmark for evaluating and comparing different algorithms, especially deep learning models.

optimizer with an initial learning rate (α) of 0.00015. If there's no improvement in the validation RMSE after 4 epochs, we reduce the learning rate by 20%. We save the model weights to disk after every epoch, provided they lower the validation RMSE. The training process halts if there's no improvement in the validation RMSE for 10 consecutive epochs. Each training run was initially planned for 50 epochs.

Our training methodology comprises two primary phases: fine-tuning and full-tuning.

Fine-tuning: We initially set the base model (EfficientNet-B0) parameters pre-trained on ImageNet-1K (see Section 6.3) as non-trainable, focusing training on the parameters of the additional layers ('tail') that we integrated into the ENB0. After 30 epochs, the RMSE on the validation set plateaued at approximately 0.11807. Dissatisfied with this outcome, we proceeded to the next phase.

Full-tuning: Beginning from the weights obtained in the fine-tuning phase, we switched all parameters in the complete model (ENB0 plus the added 'tail') to trainable. The model underwent 40 additional training epochs. This full-tuning achieved RMSE values of 0.0814 and 0.0810 on the validation and test sets, respectively.

Remarkably, the full-tuned model's performance on the test set ranked within the top 10 scores on the official Kaggle competition leaderboard.

6.5 Performance evaluation

The Kaggle challenge asks competitors to evaluate the performance of the models over the training data with the RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2}$$

Where:

- N is the number of galaxies times the total number of responses. This represents the total number of data points (or predictions) being evaluated.

- p_i is the predicted value for the i-th data point.

- a_i is the actual value for the ith data point.

7 Gallery

The image below displays the first few rows and columns of the CSV file, which stores the predictions made by our model for galaxies within the test set.

GalaxyID	Class1.1	Class1.2	Class1.3	Class2.1	Class2.2	Class3.1	Class3.2	Class4.1
100018	0.407888	0.582523	0.014407	0.029137	0.532876	0.107306	0.396716	0.185159
100037	0.531101	0.437521	0.013075	0.338021	0.096537	0.013105	0.08513	0.023849
100042	0.78581	0.207034	0.010072	0.141288	0.053498	0.007146	0.050583	0.014603
100052	0.634374	0.334699	0.029625	0.00811	0.320185	0.017686	0.305511	0.049302
100056	0.316638	0.674685	0.00692	0.676182	0.050973	0.010971	0.04835	0.021833
100058	0.238444	0.729814	0.030508	0.015315	0.721773	0.052961	0.634807	0.290587
100062	0.441033	0.487294	0.069283	0.275303	0.226579	0.039146	0.211201	0.099828
100065	0.598563	0.353768	0.034324	0.026118	0.334162	0.074819	0.217131	0.051989
100071	0.045705	0.944093	0.012568	0.012842	0.903818	0.682815	0.203796	0.831806
100076	0.266995	0.690293	0.042448	0.084344	0.59689	0.060578	0.557547	0.309636
100084	0.653159	0.337232	0.019585	0.036471	0.308667	0.044546	0.280843	0.080316

Figure 22: Excerpt from the CSV file showing the first rows and columns of the predictions of our model for galaxies in the test set.

The following figures exemplify the values of this dataframe and their predictive power upon the visual morphology of galaxies in the test set.

Figure 23: Galaxies with a high 'Class1.1' value, i.e. galaxies with a high likelihood of being considered "smooth" by GZ volunteers

Figure 24: Galaxies with a predicted high 'Class4.1' value, i.e. galaxies with a high likelihood of having a spiral pattern identified by GZ volunteers.

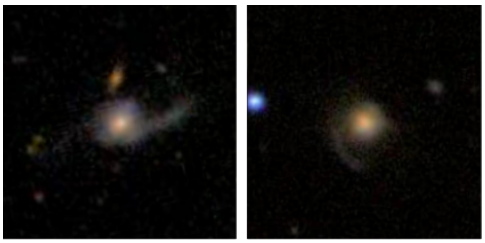


Figure 25: Galaxies with a predicted high 'Class11.1' value, i.e. galaxies with a high likelihood of being considered as having one spiral arm by GZ volunteers.

Figure 26: Galaxies with a predicted high 'Class11.3' value, i.e. galaxies with a high likelihood of being considered as having three spiral arms by GZ volunteers.

Figure 27: Galaxies with a predicted high 'Class3.1' value, i.e. galaxies with a high likelihood of being considered as having a central bar by GZ volunteers.

Figure 28: Galaxies with a predicted high 'Class2.1' value, i.e. galaxies with a high likelihood of being considered as having an edge-on orientation by GZ volunteers.

Figure 29: Galaxies with a predicted high 'Class1.3' value, i.e. galaxies with a high likelihood of being considered as stars by GZ volunteers.

Figure 30: Galaxies with a predicted high 'Class6.1' value, i.e. galaxies with a high likelihood of being considered as "odd" by GZ volunteers

Figure 31: Galaxies with a predicted high 'Class5.1, Class5.2, or Class5.4' value, i.e. galaxies with a high likelihood of being considered as bulgeless (left), having a just noticeable bulge (center), or having a dominating bulge (right) by GZ volunteers

8 Conclusion

In this report, we explored the usage of an ImageNet-1K pre-trained EfficientNet-B0 model for galaxy morphology classification. We attempted to predict the vote fractions of the 79,975 testing images from the Kaggle Galaxy Zoo challenge. We evaluate this model using the standard competition metric i.e. RMSE score and rank among the top 10 on the public leaderboard on the Kaggle webpage for this competition.

9 Future Work

In our future endeavors, we would like to explore the synergy between Efficient Nets and other cutting-edge architectural frameworks. Our goal is to train these integrated networks on substantially larger datasets. Looking ahead, with the advent of expansive astronomical surveys like the Dark Energy Survey (DES), the Large Synoptic Space Telescope (LSST), and the Australian Square Kilometre Array Pathfinder (ASKAP), we anticipate an unprecedented influx of galactic imagery. The application of EfficientNets in this context holds great promise for revolutionizing the way we automatically classify galaxies, potentially leading to more accurate and faster results in astronomical research.

References

[1] Hubble, E. P., "Extragalactic nebulae.", The Astrophysical Journal, vol. 64, pp. 321–369, 1926.

[2] Naim, A., et al. "Automated morphological classification of APM galaxies by supervised artificial neural networks." Monthly Notices of the Royal Astronomical Society 275.3 (1995): 567-590.

[3] Bazell, D., and David W. Aha. "Ensembles of classifiers for morphological galaxy classification." The Astrophysical Journal 548.1 (2001): 219.

[4] Gauci, A., Zarb Adami, K., and Abela, J., "Machine Learning for Galaxy Morphology Classification", arXiv e-prints, 2010. doi:10.48550/arXiv.1005.0390.

[5] Kalvankar, Shreyas, Hrushikesh Pandit, and Pranav Parwate. "Galaxy morphology classification using efficientnet architectures." arXiv preprint arXiv:2008.13611 2020.

[6] Tan, Mingxing, et al. "Mnasnet: Platform-aware neural architecture search for mobile." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[7] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." International conference on machine learning. PMLR, 2019.

[8] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[9] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. pmlr, 2015.

[10] Dieleman, Sander, Kyle W. Willett, and Joni Dambre. "Rotation-invariant convolutional neural networks for galaxy morphology prediction." Monthly notices of the royal astronomical society 450.2 (2015): 1441-1459.

A Appendix: FITS data in Astronomy

FITS stands for Flexible Image Transport System. It's a file format mostly used for storing scientific data (e.g. images and tables), along with the necessary metadata (e.g. in the case of astronomy, atmospheric conditions at the time of the observation or instrumental setup) for its interpretation.

A given FITS file can contain multiple "extensions":

- Image Extension: This "extension" contains a 2D array of pixel values that make up an image.

- Table Extension: This one stores tabular data. It might include columns for different parameters, such as the position, brightness, and velocity of stars. Each row in the table corresponds to a different celestial object.

- Header Information: Each extension has a header which describes the data (that is, the header contains the metadata we mentioned above).

For instance, a FITS file can store more than one image, that is the case of the so-called FITS cubes which store a number of 2D arrays. Typically, in astrophysics, the third axis in those cubes represents a spectral dimension (wavelength).

FITS is file format governed by the International Astronomical Union (IAU) and has been widely adopted by the astronomy community because of its capacity to store large amounts of information (along with its metadata in a practical header file) and its support of different data types including integers, floats, and complex numbers.

Given the nature of the task approached during this project, this appendix aims to be a very brief introduction to the data acquisition and processing steps to build RGB images from FITS files obtained with CCD sensors.

When an observation is made using a CCD, the resulting digital data (an array of pixel values) needs to be stored and then transmitted for further analysis. This raw data from the CCD, along with relevant metadata, is often saved in the FITS format. This makes the FITS file a standardized representation of the CCD's observation.

Therefore, in many observatories and with many astronomical instruments, the image-building process often goes like this: CCD captures the observation → Data processed and converted to digital format → Digital data saved in FITS file (plus any other step that may be considered useful).

A CCD is a semiconductor device, typically made of silicon. The surface of the CCD is partitioned into a large array of tiny cells or regions, often referred to as pixels (short for "picture elements"). Each pixel functions as an individual light detector.

The basic principle that allows CCDs to work is the photoelectric effect. This is a quantum mechanical process where photons ("particles" of light) that strike a material can excite electrons from that material and possibly liberate them from the material. When visible light falls on the silicon surface of a CCD, it can free electrons from the silicon atoms which the pixel "traps" using an electric potential gradient (∇V). The amount of freed (and trapped) or "photo-generated" electrons in a pixel is directly proportional to the amount of incident light on that pixel. So, brighter regions in an astronomical scene will produce more electrons in their corresponding pixels than dimmer regions.

In Section 4 we hinted at the tremendous amount of work necessary to "clean" even a few FITS files. In the following, we take a peek at some of the steps involved.

A.1 Data reduction

The "data cleaning" process raw FITS files have to go through is often termed "data reduction" in astronomy. The primary goal of this process is to remove instrumental and observational artifacts from the images, making them suitable for scientific analysis.

For large surveys like the SDSS used to build the Galaxy Zoo image samples, these calibration procedures are systematically applied to all data, and the resulting calibrated images are made available to the public and researchers. Thus, when we access SDSS data, we get images that have already undergone these crucial calibration steps and are ready for scientific analysis.

A.1.1 Bias estimation and subtraction

CCDs register an electronic signal even without any light exposure; with the shutter closed and zero exposure time (so the CCD does not accumulate any dark current). This inherent signal is called the bias signal (also just bias or level) of the CCD.

This bias signal is a consequence of the following:

To ensure that the digital value (analog to digital conversion) is always positive (even for pixels that capture no light), the electronics of the system (charge to voltage convertor, amplifier,...) introduce a constant offset voltage before the conversion to a digital value. This offset ensures that the digital values will always be positive and that the inherent electronic noise does not result in negative values. This constant offset is the "bias".

Ensuring that the digital values are always positive is important for several reasons:

In the digital storage system of the CCD camera, pixel values are often represented as unsigned integers. Unsigned integers can only take on non-negative values. If the signal was allowed to go negative due to inherent electronic noise or other factors, it could cause errors in the storage system. For instance, some electronic "noise" is expected in any electronic system, including CCDs. Without a bias offset, fluctuations from this noise could cause pixel values to oscillate around zero, leading some to go negative. By introducing a positive bias, even pixels that experience a negative fluctuation due to noise will remain positive in value.

It is worth saying that while the raw output from CCD detectors typically has non-negative pixel values due to the bias offset, several factors can lead to negative pixel values in processed FITS images.

Bias subtraction is critical not just for the science images but also for the calibration frames themselves (more on this later). For example, before a dark frame can be used to correct for dark current, it first needs to have the bias subtracted out.

A bias frame is an image taken with a CCD array with zero exposure time and with the shutter closed. Hence, this frame captures the electronic signal of the CCD without any light exposure. Multiple bias frames are usually taken to, for example, average out any random noise.

We often combine the individual bias frames to create a "master" bias frame. This can be done by averaging the pixel values across all the bias frames. Median combining is often preferred over a simple average, as it is less sensitive to outliers like cosmic ray hits.

For every image subtract the master bias frame from it, pixel by pixel. This subtraction removes the electronic bias signal, leaving behind the actual light signal from the sky (plus other sources of noise like dark current that we may have not considered up to this point).

We would like to say that readout noise is not a form of bias, but both are inherent electronic characteristics of CCDs that need to be accounted for in data processing, but they are not the same.

A.1.2 Readout Noise

Readout noise arises from the inherent uncertainty in the process of reading the charge from each pixel and converting it into a digital value. This includes the shifting of charges, amplification, and analog-to-digital conversion.

It is a random noise, meaning it varies from readout to readout and it is not consistent like the bias is.

In data reduction, we would subtract the bias (using a master bias frame) to remove this constant offset. However, the readout noise remains a fundamental, random uncertainty in the data. While we cannot eliminate readout noise, averaging multiple frames can help reduce its effect, as the noise will average down with the square root of the number of frames combined.

A.1.3 Dark current

Over time, a CCD accumulates charge even without exposure to light, known as the dark current. How do we correct for this unwanted signal?

We first take several dark frames, which are images taken with the same exposure time as your science images but with the shutter closed, so no light hits the CCD pixels. After subtracting the master bias from these dark frames, we average them to create a master dark frame. We then subtract this master dark frame from our science images.

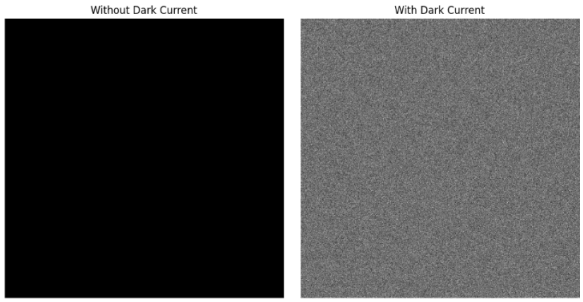


Figure 32: Simualted frames. We assume that the bias has already been subtracted from both frames, therefore, the left image appears fully black instead of grayish.

We wanted to show the reader an example of dark frame, as we do not dispose of any real one, we opted for simulating it.

The Poisson distribution is commonly used to model the number of events happening in a fixed interval of time, assuming that we know the (fixed) average rate of occurrence. The number of photons arriving at a detector in a given period of time, or the number of electrons generated in a pixel due to thermal activity (dark current), are inherently random processes that often follow a Poisson distribution.

The average rate of dark current (i.e., electrons generated per pixel per unit time) is usually provided by the CCD manufacturer or can be measured. This rate serves as the lambda (λ) parameter for the Poisson distribution.

We used a Poisson distribution to generate the image in Figure 32.

A.1.4 Cosmic Ray Removal

Cosmic rays are high-energy particles (mostly protons and He nuclei) originating from space that, when they hit the Earth's atmosphere or come directly into contact with a detector in space, can produce secondary particles (e.g. neutrinos, neutrons, muons).

When these secondary particles or the primary cosmic rays strike the silicon of the CCD, they can ionize many atoms. This ionization process means that these high-energy particles can bump a large number of electrons out of their normal positions, promoting them into the conduction band of the silicon detector.

As a result, a significant number of electrons can get collected in the pixel where the cosmic ray hit. This collection of electrons could then get read out as having a high-intensity signal, making the pixel appear unusually bright, sometimes creating a streak if the cosmic ray particle traveled through multiple pixels (if a cosmic ray hits the CCD at an oblique angle, it can travel through the CCD and affect a path of pixels, remember that CCDs are not just 2D grids on a flat plane; they have depth. Each pixel is essentially a small well dug into the silicon where electrons can be collected. A cosmic ray that penetrates the CCD can travel through this depth and, depending on its energy and incident angle, it might pass through the boundaries between adjacent pixel wells. As it travels, it can ionize silicon atoms along its path, producing free electrons in multiple pixels.)

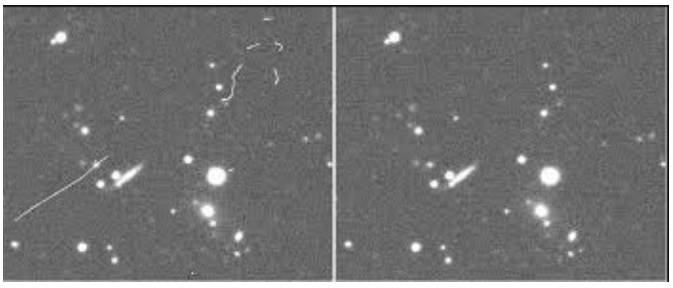


Figure 33: The left image has some "cosmic-ray" detections.

The most common and one of the most effective ways to remove cosmic ray hits is by taking multiple exposures of the same portion of the sky and then combining these images. Here is why this works:

Cosmic ray hits on a detector are a random process. This implies that the likelihood of a cosmic ray hitting the exact same pixel in two different exposures is extremely low.

If we take multiple frames of the same scene, each frame might have its own unique cosmic ray hits. But because of their randomness, these hits will appear in different places in each frame.

When we "stack" (or combine) these frames, we can employ algorithms that identify the pixels that are unusually bright compared to the same pixels in other frames. Because a cosmic ray hit is bright and isolated to a particular frame, these algorithms can easily identify and reject it.

A.1.5 Flat Fielding

Some CCD pixels might be more sensitive to incoming light than others, resulting in a brighter response even when exposed to the same amount of light.

A flat field frame is an image of a uniformly illuminated field. This could be the twilight sky (just after sunset or just before sunrise; because the sky is still somewhat bright during twilight, most stars are not visible, ensuring that they don't interfere with the flat image. This is essential as stars would introduce non-uniformities into the flat field frame), a white screen with a light behind it, or the interior of an observatory dome with its lights on. The goal is to have a smooth, evenly illuminated scene so variations in our observed frames are only due to the CCD and the optical system, not the light source.

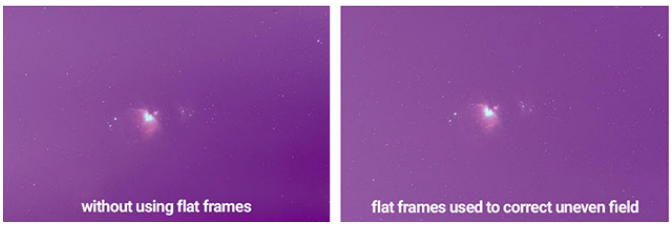


Figure 34: Correction with flat fields. Notice that the corrected image appears to have a more uniform distribution of light intensity.

Before employing the flat frames, we must apply the data reduction steps specified above. The master bias (and if needed, master dark) has to be subtracted from each flat frame to ensure any pixel-to-pixel variation is truly from the CCD/optical system and not from electronic noise.

Once the individual flat frames are cleaned out of noise, they are combined (averaged) to create a "master" flat. This averaging helps reducing random noise and gives a smooth representation of the system's non-uniformities.

Before we use the master flat to correct our science images, it's important to normalize it. This means ensuring that its average value is 1 (or some other standard value).

This way, when we divide our science image by the master flat, we are not changing the overall brightness of the image, just correcting for the non-uniformities.

After normalization, we can divide each of our science frames by the master flat. This process will correct the science frame by adjusting pixel values based on the variations captured in the master flat, thus ensuring uniform sensitivity and illumination across the entire image.

As mentioned in Section 4, the Kaggle Galaxy Zoo challenge provided volunteers with RGB, JPG images of galaxies and their immediate surroundings in the plane of the sky. We briefly explain how these color images can be built from clean, FITS files.

To build a color image (and then store it as a JPG image), we need three images from the same portion of the sky but corresponding to different wavelengths. Therefore, we need to choose three wavebands or filters corresponding to Red (R), Green (G), and Blue (B). In many astronomical datasets, these might not directly map to the visual RGB, but we can use the images from three filters and assign them to RGB channels as we see fit. For instance, the SDSS images in the GZ2 dataset were built after combining the 'i' band for Red, the 'r' band for Green, and the 'g' band for Blue.

But before we can combine the images, we need to "process them". Surveys like the SDSS do it for us, so, we do not have to do it ourselves, which would also take a very long time. I next outline some of the main steps we have to go through.

A.1.6 Image Processing

Adjusting the contrast

Adjusting contrast in an image means modifying the difference between the darkest and the lightest areas to make them either stand out or blend in.

At its core, changing the contrast of an image is no more than applying a transformation to the pixel values of that image. The goal of this transformation is to modify the range and distribution of pixel intensities, thus altering the perceived contrast.

Here are a few methods and the transformations commonly applied:

Linear Stretch:

$$\text{new pixel value} = \frac{\text{pixel value} - \text{min value}}{\text{max value} - \text{min value}} \times 255$$

Non-linear Stretch, for example, a logarithmic transformation:

$$\text{new pixel value} = c \times \log(1 + \text{pixel value})$$

where c is a constant.

Gamma Correction:

$$\text{new pixel value} = \text{pixel value}^{\gamma}$$

where γ is a value proposed by the user.

Of course, the transformation that is most convenient to use is a function of the desired outcome.

Increasing the contrast of an image results in a more pronounced brightness difference between the darker and lighter areas of the image. This can result in loss of detail in very dark or very light areas (e.g. the stretching can push the darkest tones towards pure black and the lightest tones towards pure white. When this happens, any subtle variations or details in those regions can be lost because multiple tones might get clamped to either the maximum or minimum values).

Figure 35: Low contrast vs High contrast.

A.1.7 Tweaking color balance

After aligning and combining the channels into an RGB image, we might need to tweak the color balance to ensure that the combined image has a desired or accurate color representation. For example, if an RGB image appears too "blue", we might enhance the red and green channels relative to the blue to achieve a more neutral color balance.

Here is an example of how this could be done:

The following technique is sometimes referred to as "color matrix transformations" or "color grading using matrices" (the use of a matrix for color transformations is not the only way to adjust colors).

To adjust the color balance of an image, we can use a 3x3 matrix to transform the RGB values of every pixel in the image. The idea is that each new color channel (R', G', B') is a weighted combination of the original channels (R, G, B).

Let us represent the original RGB values of a pixel as a column vector $\mathbf{v}$, and let $\mathbf{M}$ be the 3×3 transformation matrix for color balancing:

$$\mathbf{v} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \mathbf{M} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

The new RGB values $\mathbf{v}'$ are then calculated by:

$$\mathbf{v}' = \mathbf{M} \times \mathbf{v}$$

In other words:

$$\begin{aligned} R' &= a \times R + b \times G + c \times B \\ G' &= d \times R + e \times G + f \times B \\ B' &= g \times R + h \times G + i \times B \end{aligned}$$

To give a simple example, suppose we want to enhance the red channel and slightly decrease the influence of the green channel, we may use the following color matrix:

$$\mathbf{M} = \begin{bmatrix} 1.2 & -0.1 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In this example:

- The red value R' is 120% of the original R minus 10% of the original G.

- The green value G' is 90% of the original G.

- The blue value remains unchanged.

Once we are satisfied with the color of our three FITS frames, we can proceed to combine them. Each FITS image should be mapped to one of the RGB channels, and then the channels are combined to create a single color image.

Figure 36: Notice that the background appears to have a reddish hue rather than a deep black typical of space images. This image needs some color tweaking.

In other words, the intensity of each pixel in each FITS frame is mapped into the intensity of the corresponding sub-pixel of an RGB image. The blending of these sub-pixels at varying intensities produces the wide range of colors we see on our RGB images.

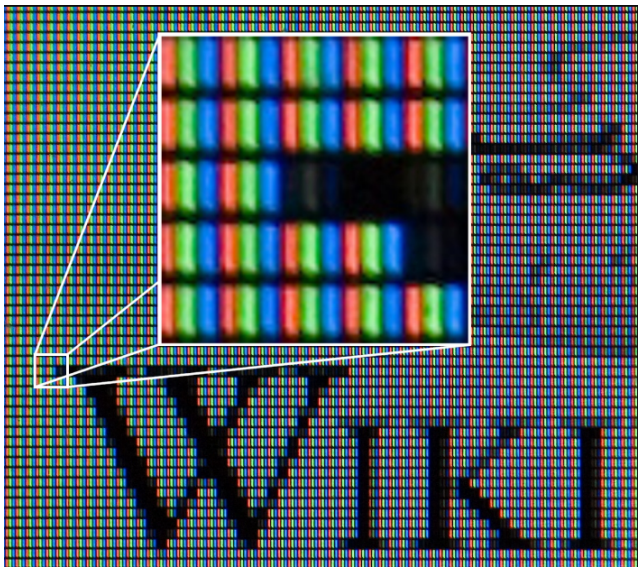


Figure 37: RGB pixels are made up of three subpixels.