Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Jonas Arilho Levy

Using Deep Learning to Detect Galaxy Mergers

São Paulo
November 2019

Using Deep Learning to Detect Galaxy Mergers

Final monograph for the course
MAC0499 Supervised Capstone Project.

Supervisor: Mateus Espadoto
[Co-supervisor: Prof. Dr. Roberto Hirata Junior]

São Paulo
November 2019

Abstract

This work investigates the use of Deep Learning techniques to detect galaxy mergers using astronomical imaging data from photometric surveys. We analyse three Convolutional Neural Networks architectures found in the literature and compare their performances training from scratch and using transfer learning. The models outperformed previous automatic detection methods and transfer learning showed a slightly better performance over training from scratch. A reliable approach to detect galaxy mergers is presented with different models achieving a precision of 0.97 on the dataset used.

Keywords: deep learning, galaxy mergers, convolutional neural networks.

Resumo

Este trabalho investigata o uso de técnicas de Aprendizagem Profunda (*Deep Learning*) para detectar fusões de galáxias usando dados de imagens astronômicas de levantamentos fotométricos. São analisadas três arquiteturas de Redes Neurais Convolucionais encontradas na literatura e comparamos seus desempenhos treinando do zero e usando *transfer learning*. Os modelos demonstraram desempenho superior a métodos anteriores de detecção automática e *transfer learning* mostrou um desempenho ligeiramente melhor do que treinar do zero. Uma abordagem confiável para detectar fusões de galáxias é apresentada com diferentes modelos atingindo uma precisão de 0,97 com o conjunto de dados utilizado.

Palavras-chave: aprendizagem profunda, fusões de galáxias, redes neurais convolucionais.

Contents

Chapter 1

Introduction

In astronomy, the phenomena of gravitational interaction between galaxies is called a galaxy merger, which happens when at least two galaxies get very close to each other. The study of merging galaxies is important because astronomers can use the merger rate as a measurement of galaxy evolution (Lotz *et al.*, 2004) as well as study the transformation of massive galaxies (Toomre and Toomre, 1972) and the triggering of quasars (Lin *et al.*, 2008). In this work, we study how Convolutional Neural Networks can be used to detect merging galaxies, based on image data from photometric surveys, which can enable astronomers to look for galaxy mergers in larger datasets.

1.1 Objectives

The main goal of this work is to investigate the use of Deep Learning techniques, particularly Convolutional Neural Networks (CNN), to detect galaxy mergers using astronomical imaging data from photometric surveys. We compare three image classification models found in the literature and use two different training setups, namely training from scratch and transfer learning, *i.e.*, using a pre-trained model as a starting point for the training process, and we compare our results to the ones obtained by a recent work by Ackermann *et al.* (2018), which obtained good results by using transfer learning. To enable this comparison, we used the same dataset as Ackermann *et al.* (2018) in all experiments.

1.2 Contributions

The main contribution of this study is to present a systematic comparison of different Deep Learning models applied to galaxy merger detection, expanding on the work of Ackermann *et al.* (2018). We present results on how different architectures perform on the selected problem, and we believe this work could be of great use to the astronomical community interested in the detection of galaxy mergers.

1.3 Organization

The text is organized as follows: in Chapter 2 we present a brief overview of Neural Networks, Gradient Descent and Convolutional Neural Networks, alongside a basic exposition of astronomy concepts, such as spectrometry, photometry, galaxy morphology and interactions. In Chapter 3 we present the experimental setup and dataset used. In Chapter 4 we present and discuss the results of the different experiments, and compare them with the results obtained by others. Chapter 5 concludes this work.

Chapter 2

Background

2.1 Astronomy

2.1.1 Galaxy Morphology

Astronomers divide galaxies into groups based on their visual appearance, and the first widely adopted system for galaxy morphological classification was proposed by Hubble (1926) and is known today as the Hubble Sequence, in which galaxies are roughly divided into three main groups: elliptical, spiral and lenticular galaxies. Often galaxies that do not fit in any of those groups are called irregular. When two galaxies come close together, they start interacting with each other due to gravitational pull and this effect is called a galaxy merger. Since galaxies are extremely large and there is a lot of empty space between the stars, there is a low probability of star systems colliding with each other, and during the merging process it is mainly the gas and dust of the galaxies that come together and form tidal tails and emit a lot of light.

As detailed by Karttunen *et al.* (2016), in early stages of the Universe, galaxies used to be much closer to each other and interactions between them must have been much more common. This is why there are good reasons to believe that bright giant elliptical galaxies may have been formed by the merger of other smaller spiral galaxies at some stage in their history. Systems of interacting galaxies are usually classified according to their size. **Pairs** of galaxies that interact with each other are the simplest form of organization and where most of the mergers occur. **Groups** are the most common type of galaxy systems, comprised by small, irregular associations of a few tens of galaxies. **Clusters** are galactic systems that contains a larger number (at least 50) of bright galaxies. Groups and clusters of galaxies may form even larger systems, called **Superclusters**.

Between galaxies in those systems there can be different stages of interaction that are better illustrated by the images in Figure 2.1. Each of the 6 images is a different galaxy picked specifically to depict a particular stage of galaxy interaction:

1. The first sign of an interaction;

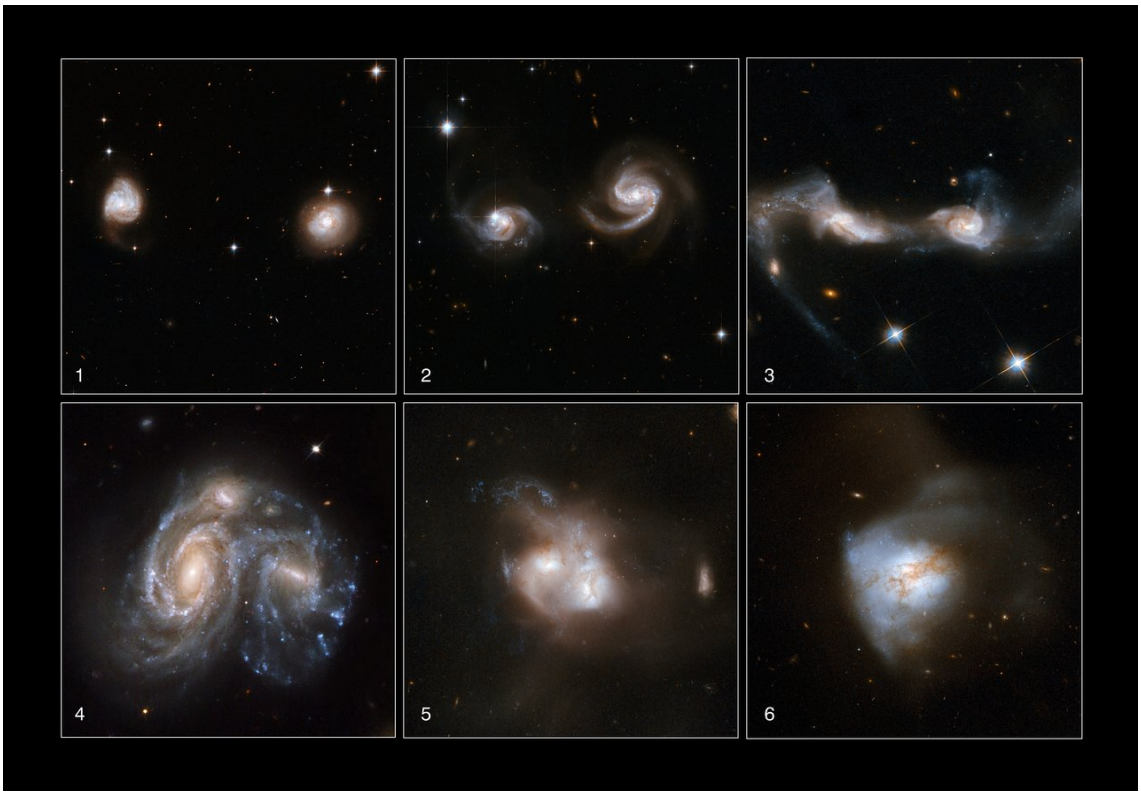

Figure 2.1: *Snapshots of galaxies at different stages of merging from ESA/Hubble, available at: www.spacetelescope.org/images/heic0810ac/.*

2. Tidal tails stretch out as the outer reaches of the galaxies begin to intermingle;
3. Long tidal tails are the signature of an interaction;
4. The galaxy cores approach each other and their gas and dust clouds are accelerated dramatically;
5. Gas and dust are siphoned into the active central regions and they become some of the brightest infrared objects in the sky;
6. Stars form from large clouds of gas creating brilliant blue star clusters triggered by the interaction between the galaxies.

Astronomers have been using several different methods to automatically detect galaxy mergers. Some of them (Lin *et al.*, 2004) identify luminosity peaks by using the *close-pairs* algorithm. Other methods rely on hand-crafted feature detectors to classify the imaging data and are used by several researchers (Conselice, 2003; Goulding *et al.*, 2017; Hoyos *et al.*, 2012; Lotz *et al.*, 2004). Most of these works share a common issue, which is the difficulty of capturing all the types of galaxy mergers, and the variations in the detection at each different stage of the merging process.

2.1.2 Photometric Surveys

In astronomy, there are two main tools to collect data, namely Spectroscopy and Photometry. Spectroscopy is the measurement of light as a function of wavelength, divided in thousands of very narrow bands. This enables astronomers to study objects in much detail, but with the drawback of being able to explore only few objects at a time. Photometry, on the other hand, enables the capture of data from thousands of objects at once, but with only a few, wide, wavelength bands, typically 5 to 30. The current approach used by astronomers is to gather information for many objects using photometry, detect objects of interest among those observed, and then capture fine-grained data for those few selected using spectroscopy. This data is essential to study the evolution of galaxies.

2.2 Deep Learning

Over the past few years, artificial intelligence has attracted a good amount of attention following the success of machine learning and neural networks applications. The idea behind machine learning is to develop computer algorithms that can automatically learn patterns from data. There are several approaches, the most common being Supervised Learning (for data with labels), Unsupervised learning (for data without labels) and Reinforcement Learning (for agents taking actions in an environment). There are several machine learning algorithms, with artificial neural networks (ANNs) being one of the most successful. More recently, the idea of Deep Learning appeared, which is, in simple words, the capability of learning how to extract good features from data while learning how to solve the classification problem. This enabled a revolution in domains such as text, image and video, where classic machine learning techniques previously relied on hard-to-tune, hand-crafted feature extractors to work well.

2.2.1 Neural Networks

Neural networks have been a focus of research since the early 1940s by McCulloch and Pitts (1943) when artificial neural networks (ANNs) took a biological inspiration by the way neural networks in animals work for visual processing in the cortex and how this information flows inside the brain through its neurons. In their paper, they introduced the idea of neural networks as computing machines and latter were backed up by Hebb (1949) for postulating the first rule for self-organized learning, which is an attempt to explain the adaptation of brain neurons during the learning process (in a biological standpoint).

A decade later Rosenblatt (1958) would be responsible for proposing the perceptron as the first model for supervised learning. It is an algorithm that can predict whether a vector of numbers belongs to a class or not. This makes the perceptron a type of linear classifier that combines a set of weights together with the input vector and makes predictions. Formally, this can be defined as follows (Nielsen, 2015):

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{2.1}$$

This simple model takes the input vector $x_1, x_2, \ldots$ and weights $w_1, w_2, \ldots$ that measure the importance of each input to the final sum and then finally uses an activation function to decide the output according to some threshold (in this case it is either 0 or 1). This model can be better visualised in Figure 2.2.

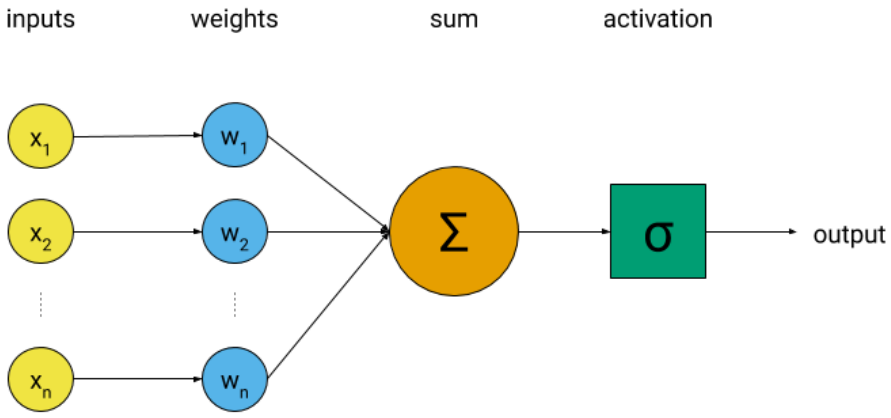

Figure 2.2: *A simplified representation of the Perceptron.*

These concepts were generalized by Minsky and Papert (1969) with their book titled *Perceptrons* that outlined several limits to what perceptrons could do and led to a so called *AI Winter* that would last for almost two entire decades and would halt the research efforts on neural networks. One of the problems with perceptrons that they described is the XOR (*exclusive or*) problem, that showed that using a neural network made of one layer of perceptrons could not accurately predict the outputs of XOR logic gates because this problem is not linearly separable, and perceptrons can only classify data with linear separability.

The limitations of the single layer perceptron were overcome by the introduction of the Multilayer Perceptrons (MLP) by Rumelhart *et al.* (1988). They consist of one input layer, one output layer and one or more hidden layers, as illustrated in Figure 2.3. The Multilayer Perceptrons solved the previously outlined problems with Perceptrons because they are able to distinguish data that is not linearly separable. This model also has an activation function for each individual neuron and became the building block for the modern architectures of neural networks.

Instead of using Linear classifiers such as the original Perceptron, modern architectures use different activation functions to better determine the output of one neuron by modulating the output to an interval. These functions are very close to a linear classifier, but each of them has some unique property that makes it better in practice because they introduce non-linearity to the network. Most of them are differentiable in their domain, and those functions alongside the addition of more hidden layers made Neural Networks a viable option once

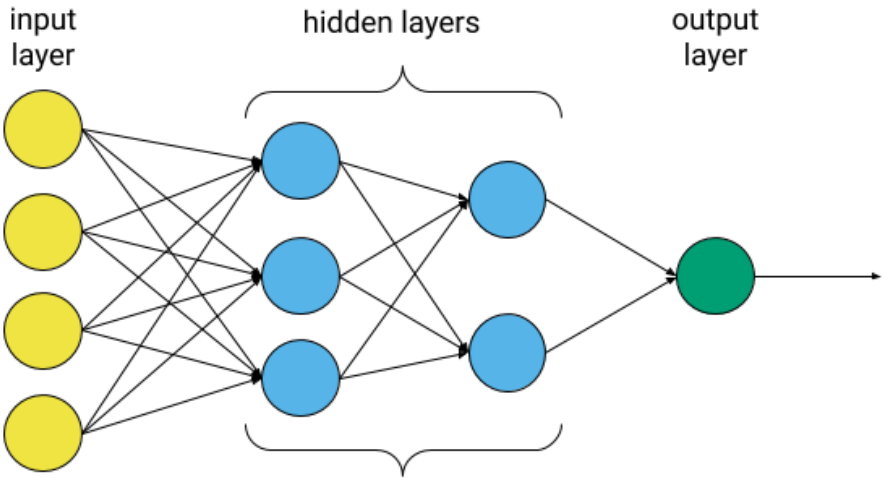

Figure 2.3: *A simple Multilayer Perceptron example.*

again. The Sigmoid and hyperbolic tangent functions were extensively used as activation functions in the past, but most modern architectures nowadays use ReLU (Nair and Hinton, 2010) or even Leaky ReLU (Maas *et al.*, 2013), which are faster to compute, and even though the ReLU function is not differentiable in all points of its domain, it does not make a difference numerically. These activation functions are plotted below:

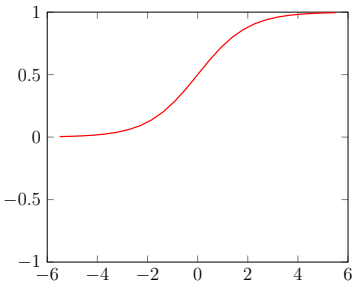

Figure 2.4: *Sigmoid.*

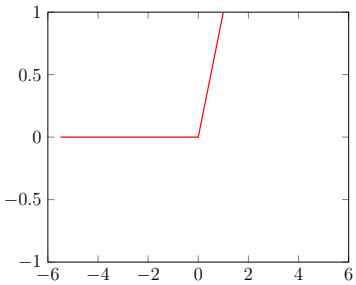

Figure 2.5: *ReLU.*

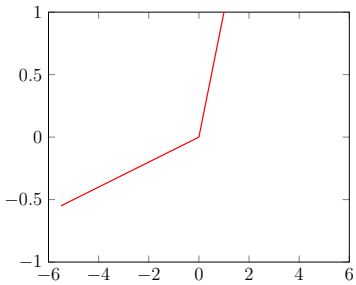

Figure 2.6: *Leaky ReLU.*

2.2.2 Gradient descent and backpropagation

To understand how a network is trained, an overview on how the gradient descent and backpropagation algorithms work is necessary. In neural networks, gradient descent is used to minimize a cost function iteratively by moving in the direction of steepest descent to update the weights of the model. A good analogy is thinking about a ball rolling downhill until it finds a deep valley and stops.

This algorithm should be able to determine the weights w (and biases) so that the output $a(x, w)$ calculated by the model is closer to the expected label $y(x)$ for each input x on a training set of size n. To determine how good the outputs are approximated, the most popular cost function can be defined based as the mean squared error (MSE) of the

calculated outputs in comparison with the expected classes. This function can be represented by a simplification of the notation used by Nielsen (2015):

$$J(w) = \frac{1}{2n} \sum_{x} \|y(x) - a(x, w)\|^2 \tag{2.2}$$

Another popular loss function, used specifically for binary classification problems, is the binary cross-entropy function. It measures the performance of a classification model whose output is a probability value between 0 and 1, and it is also known as *log loss* because it uses the logarithm of the predictions $a(x, w)$ together with each expected label $y(x)$ for the weights w (and biases). A simplified representation of this function can be defined as follows (Nielsen, 2015):

$$J(w) = -(y(x) \log a(x, w) + (1 - y(x)) \log (1 - a(x, w))) \tag{2.3}$$

The goal of the gradient descent training algorithm is to minimize the cost $J(w)$ such as the ones defined in Equations 2.2 or 2.3, depending on the loss function chosen. Since the error surface of the chosen loss function may be non-convex, there is no guarantee that a global minimum will be found, but modern implementations of gradient descent are known to be able to find good local minima. Using the deductions explained in details by Nielsen (2015), it is possible to define the variation of the weights w as a function of the gradient of the cost function $J(w)$ and the learning rate η.

$$\Delta w = -\eta \nabla J \tag{2.4}$$

A good way to iteratively calculate the gradient of the cost function is to use an algorithm called backpropagation. The modern implementation of the backpropagation learning algorithm used in convolutional neural networks was defined by LeCun *et al.* (1989) and follows a very detailed mathematical demonstration of the equations necessary for the algorithm to work. For simplicity only an overview of the general method is presented here.

Backpropagation is the name of the algorithm because it propagates the error on the output back from the last layer until the first one. A simple representation of the algorithm based on the Perceptron is illustrated in Figure 2.7 and followed by the main steps necessary for the implementation of backpropagation.

1. **Input** x: Set the activation a_1 for the input layer;
2. **Feedforward**: For each layer l, compute the activation $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$;
3. **Output error**: Compute the error of the output layer;

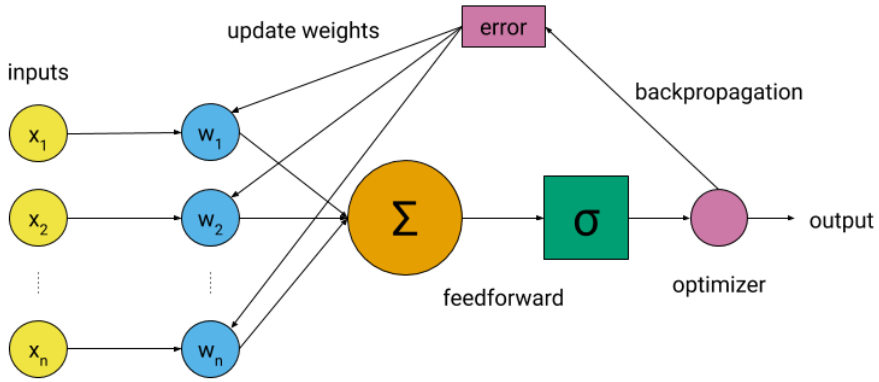

Figure 2.7: *A simplification of the application of backpropagation on the previous Perceptron example in Figure 2.2*

4. **Backpropagate**: Compute the error for each layer and propagate it to the previous one;

5. **Update weights**: Calculate the gradient of the cost function to update the weights for each layer.

A problem with backpropagation is that it requires the previous calculations of the derivatives for the layers of the neural network. To mitigate this, there is an version of this algorithm called Stochastic Gradient Descent (*SGD*), which is a variation of gradient descent that uses randomly selected samples of training inputs instead of training the entire dataset at the same time. Nowadays there is also the commonly used mini-batch SGD, that makes an statistical estimation of the entire training set by using samples in mini-batches for each epoch and calculates the gradients for each mini-batch.

An improved version of the basic stochastic gradient descent algorithm is called Adam (short for Adaptive Moment Estimation) and was proposed by Kingma and Ba (2014). It adapts the learning rate η at each iteration and is considered to be a very robust improvement on previous optimizers.

2.2.3 Convolutional Neural Networks

In image processing, convolution is a very important operation that transforms an image by applying a kernel over each group of pixels. Adapted from the convolution operator defined by Bracewell and Bracewell (1986), each pixel of the image Y can be defined as a 2D matrix that each pixel $i \times j$ is the result of the convolution of the kernel filter K of size $m \times n$ and the original image X, as follows:

$$Y[i,j] = \sum_{m=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \sum_{n=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} K[m,n] \cdot X[i-m, j-n] \tag{2.5}$$

This concept was applied by LeCun *et al.* (1995) with the first Convolutional Neural Network, called LeNet, with 3 different types of layers: The **convolutional** layers use the convolution operation to process the image; the **pooling** layer reduce the dimensions of the data by combining the outputs of the previous layer into a single neuron in the next layer; and the **fully-connected** (also known as **dense**) layer that classify the images and work similarly as a Multilayer Perceptron. Large networks can have **blocks** that group up layers that serve a particular purpose and also specific *activation* functions for each convolutional or fully-connected layer. The most common layers, blocks and activation functions are represented on Figure 2.8.

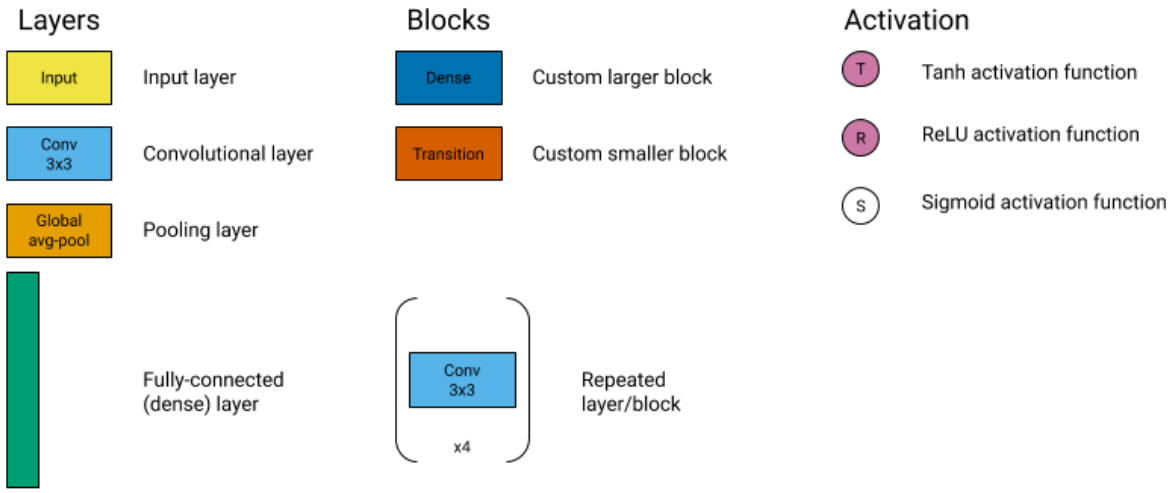

Figure 2.8: *Legend for the symbols used to illustrate the architectures inspired by this article from Raimi Karim at www.towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d/.*

Based on the original architecture LeCun *et al.* (1998) designed LeNet-5 (Figure 2.9), an improved version for the purpose of recognizing handwritten digits in 32x32 pixel images. At the time, the activation function used was the *tanh* and the average-pooling layer had trainable weights.

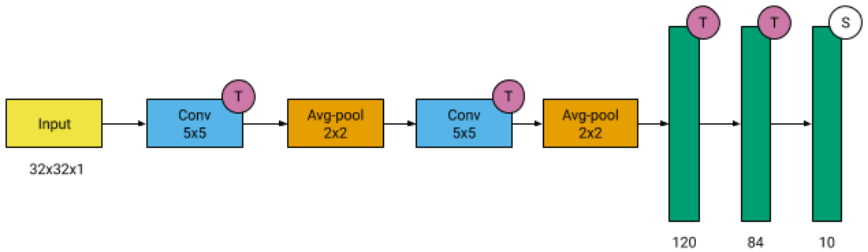

Figure 2.9: *LeNet-5 architecture (Legends on Figure 2.8).*

After over 10 years without major advances on this field, Alexnet appeared (Figure 2.10) designed by Krizhevsky *et al.* (2012). With 60 Million parameters, it has 3 convolutional layers more than LeNet-5 and was the first network to implement ReLU activation functions.

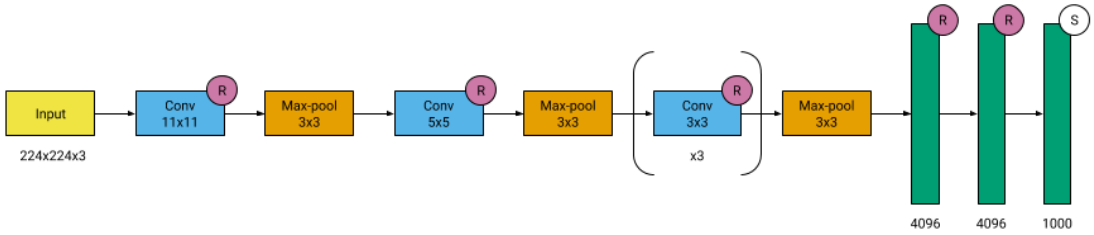

Figure 2.10: *AlexNet architecture (Legends on Figure 2.8).*

At this point the more recent architectures start to get deeper, namely VGG-16 (Figure 2.11) created by Simonyan and Zisserman (2014), who were part of the Visual Geometry Group (VGG) at the University of Oxford. The network consists of 138 Million parameters, has 13 convolutional and 3 fully-connected layers and also has a deeper variant, namely VGG-19.

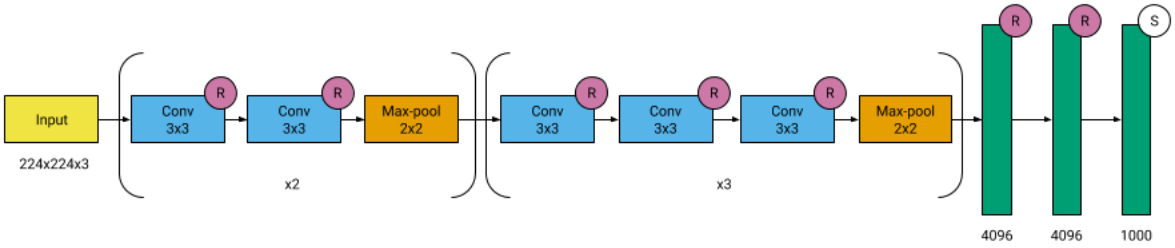

Figure 2.11: *VGG-16 architecture (Legends on Figure 2.8).*

With the popularisation of very networks, came the problem of designing even deeper Convolutional Neural Networks without compromising the generalisation power of the model. To do so, ResNet was developed by He *et al.* (2016) at Microsoft Research and used batch normalisation and skip connections to avoid degradation of the accuracy of deep networks and even though this was not the first architecture to use these techniques, ResNet50 (Figure 2.12) popularised them.

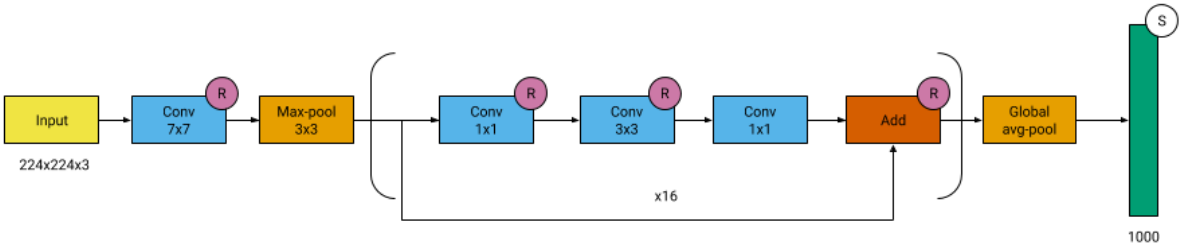

Figure 2.12: *ResNet-50 architecture (Legends on Figure 2.8).*

Inspired by the movie Inception (Nolan, 2010) and the famous quote *"We need to go deeper"*, Szegedy *et al.* (2016) designed the Inception architecture. The idea behind it was to stack dense blocks of convolutional layers and use batch normalisation in auxiliary layers.

The Inception-v3 (Figure 2.13) model became very successful and with 24 million parameters was considered a very deep network.

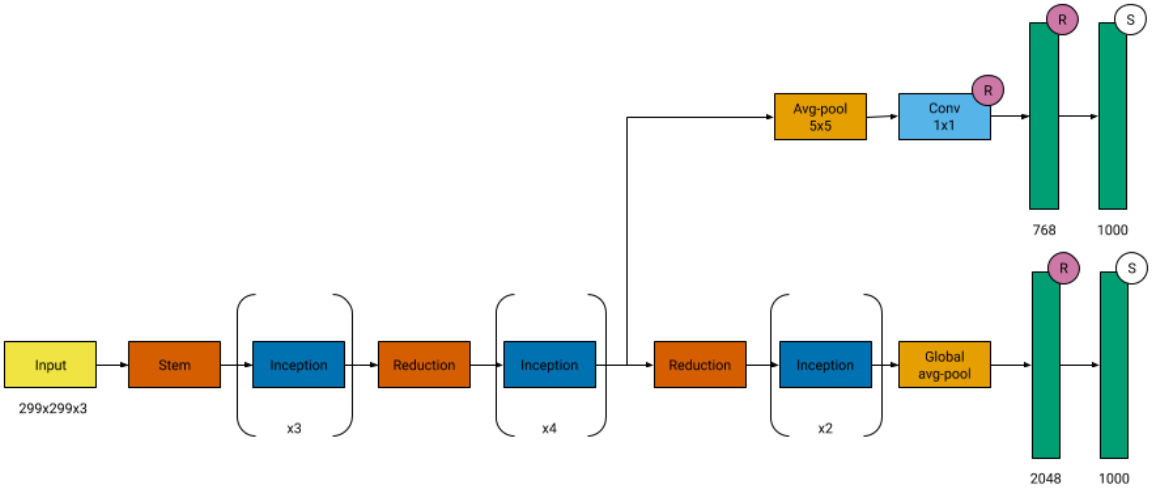

Figure 2.13: *Inception-v3 architecture (Legends on Figure 2.8).*

Similar to ResNet, Densenet by Huang *et al.* (2017) adds shortcuts among layers, but has less trainable parameters because each layer receives feature maps from all preceding layers, making a thinner and compact network. It also features a *growth rate* hyperparameter that defines how many additional channels are needed for each layer. With only 0.8 Million parameter, Densenet-121 (Figure 2.14) is the lightest of the modern architectures cited in this section.

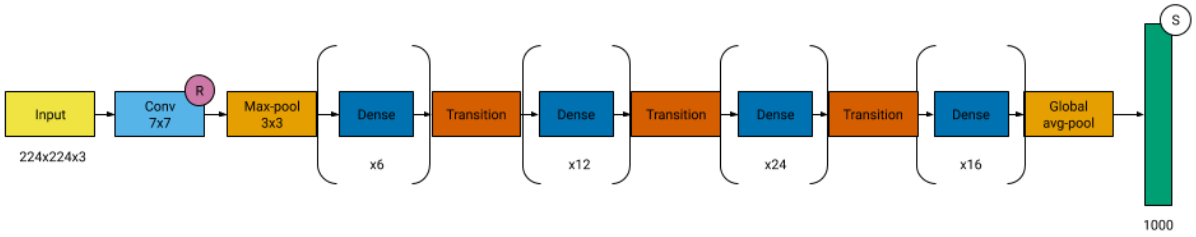

Figure 2.14: *Densenet-121 architecture (Legends on Figure 2.8).*

Chapter 3

Experimental Setup

3.1 Dataset

3.1.1 Galaxy Mergers Images

The dataset used in this work is comprised of 16000 RGB images from the Sloan Digital Sky Survey (SDSS) Data Release 7[1]. This dataset is the same used by Ackermann *et al.* (2018) in their work, which makes our results comparable to theirs. The dataset contains two classes, namely *merger*, the one we are interested in detecting, and *non-interacting*, as labeled by the Galaxy Zoo project Lintott *et al.* 2010.

There are 6000 images labeled as *merger* (see Figure 3.1) and 10000 images labeled as *non-interacting* (see Figure 3.2).

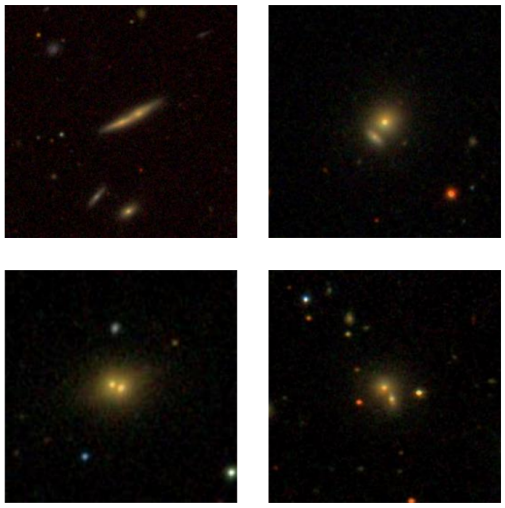

Figure 3.1: *Sample images of galaxy mergers.*

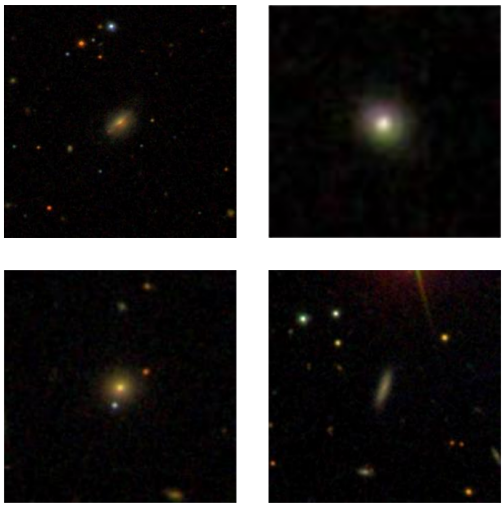

Figure 3.2: *Sample images of non-interacting galaxies.*

[1]SDSS dataset available at: http://skyserver.sdss.org/dr12/en/help/docs/api.aspx

3.1.2 Dataset preparation

The preparation consists of 3 steps[2]:

1. Loading and normalizing the images;
2. Resizing the images;
3. Splitting the dataset and augmenting the data.

The first step is to normalize the images, by dividing its values by 255, which is the maximum values of an RGB channel. The second step is to resize all images, to make their sizes uniform (96x96 pixels), since the dataset has images with different crops depending on the astronomical object observed and the zooming technique applied. The chosen size provided a good trade-off between image quality and size, with a resolution high enough for the selected CNN architectures to perform well.

Lastly, the dataset was split between training, validation and test sets at a 80/10/10 rate, since the dataset was large enough to allow for smaller test and validation sets. After that, 18 versions of each image from each set were generated by rotation in increments of 20 degrees.

Table 3.1: *Number of images in every split of the dataset after augmenting the data.*

Set	%	# of images	size (Gb)
Training	80	230400	24.8
Validation	10	28800	3.1
Test	10	28800	3.1

3.2 Experiment 1: Training networks from scratch

The first experiment consists in training 3 different architectures from scratch, *i.e.*, initializing the model with random weights and training them using the training data. The architectures used for this experiment were: VGG-16 (Figure 2.11), Inception-v3 (Figure 2.13) and DenseNet-121 (Figure 2.14). These are very popular models that are readily available, and were presented in Chapter 2. The process was done as follows:

1. Use random initialization to the weights;
2. Add top layers;
3. Train using mini-batch SGD with a standard learning rate.

[2]The scripts necessary to prepare the dataset are available at: github.com/jonasarilho/galaxy_mergers/tree/master/prepare_dataset

The top layers are basically a global average pooling layer and a fully-connected (dense) layer with 2048 units. As an activation function, Leaky ReLU was chosen as a fresh alternative to more classic functions. Explicit kernel and bias initializers were used, using the Glorot uniform initializer (Glorot and Bengio, 2010) and a constant bias $b = 0.01$.The optimizer chosen was stochastic gradient descent (SGD) with a learning rate $l = 0.01$.

As supported by Keskar *et al.* (2016), a mini-batch of size 32 was chosen for the training, and early-stopping was used to halt the process after when validation loss was no longer improving and to restore the best weights during the training process. To compensate for an unbalanced dataset, class weights were used to balance the training data.

To try to minimize overfitting, a few regularization techniques were used. Dropout is one of those, first used by Srivastava *et al.* (2014), that aims to prevent complex co-adaptations on training data by keeping or dropping-out a node given a certain probability P (in this experiment, $P = 0.5$ was used for the top layer). It is worth mentioning that this technique improves training speed significantly as well.

3.3 Experiment 2: Using pre-trained networks

The second experiment consists in the fine tuning of pre-trained convolutional neural networks. The architectures used are the same as on the first experiment, but in this one only the last convolutional blocks are trained alongside the top layers and the weights are loaded from a model pre-trained on the Imagenet dataset (Deng *et al.*, 2009). This was done as follows:

1. Load the pre-trained CNN with weights;
2. Add top layers to the CNN and use the ADAM optimizer to train only the top layers;
3. Fine-tune the last convolutional blocks using mini-batch SGD with a small learning rate and momentum.

The top layers were basically the same from the previous experiment, the only difference being the addition of more regularization in order to reduce overfitting. This was done by using a max-norm constraint, which limits the norm of the weights for a layer, and is a technique that pairs well with Dropout, as found by Srivastava *et al.* (2014). The use of L2 regularization was also used, a technique first used by Ng (2004) that incorporates penalties in the loss function.

Chapter 4

Results

4.1 Experiment 1 Results

4.1.1 Training process

We divide our experiment execution in two parts: firstly, we train the model using the training set, and check its accuracy on the validation. By doing this, we can have a good approximation of the generalization capability of the model, and also detect problems that may arise, such as overfitting. Secondly, after selecting the model that perform better on the validation set, we evaluate the model on the test set, to assess its generalization capability on yet another data set, which is important to avoid overfitting to the validation set.

The process of learning from scratch with the 3 different architectures as described in Section 3.2 takes a few epochs to complete and at the end of each training session, the training script generates a graph comparing the training and validation accuracy for each epoch. Those graphs serve as an overview of the process and each epoch took about 1 hour to complete in the setup used.

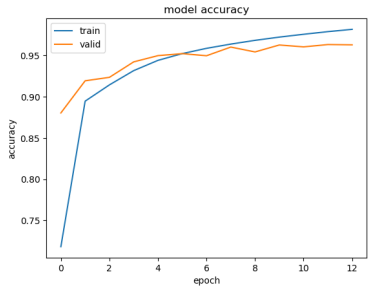

Figure 4.1: *VGG-16 training and validation accuracy from scratch.*

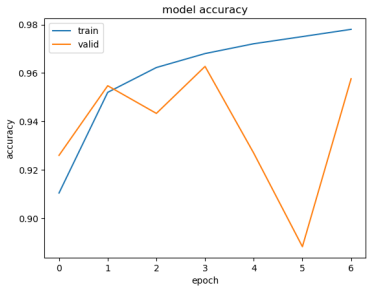

Figure 4.2: *Inception-v3 training and validation accuracy from scratch.*

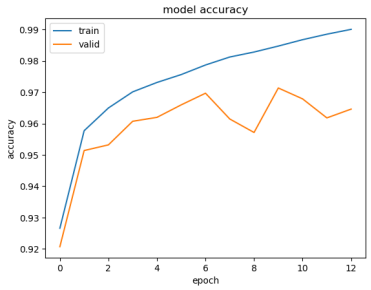

Figure 4.3: *DenseNet-121 training and validation accuracy from scratch.*

The training of the VGG-16 architecture (Figure 4.1), the shallowest model among the ones studied, took only a few epochs to converge and showed few signs of overfitting. In the last epochs, the model starts to overfit and the early-stopping callback halts the training

process. The Inception-v3 model (Figure 4.2) also took only a few epochs to train and showed some unstable results in terms of accuracy on the validation set. The DenseNet-121 architecture (Figure 4.3) showed a more clear sign of overfitting in comparison to the other models, and the early-stopping callback does well in stopping its training only after a few epochs.

In general, the models trained from scratch presented slight to moderate overfitting and also showed signs of the training set being small and unrepresentative. Other than that, the validation accuracy was already good enough for the problem.

4.1.2 Test results

After training the different architectures with different hyperparameters on the training set and evaluating them on the validation set, we measure the accuracy of the best models on the test set, which was never used during training. The accuracy was measured for each architecture and the results are shown in Table 4.1.

Table 4.1: *Accuracy on test set for each architecture.*

Architecture	Accuracy
VGG-16	95.87%
Inception-v3	95.53%
DenseNet-121	96.10%

All three architectures showed good accuracy on the test set, only differing a little from each other. It is very interesting to note that training these models from scratch without much sophisticated Deep Learning techniques yielded consistent results with high classification accuracy.

4.2 Experiment 2 Results

4.2.1 Training process

The process of using transfer learning to train the network with the three different architectures as described in Section 3.3 takes a few epochs to complete the first training with the convolutional layers frozen, and a few more epochs to complete the fine-tuning step. Again, at the end of each training session, the training script generates a graph comparing the training and validation accuracy for each epoch. Each epoch took about an hour to complete in the setup used and the first step took from 4-10 hours depending on the model.

The VGG-16 model (Figure 4.4) took just a few epochs to converge and showed moderate overfitting, which was accentuated on the last epochs, just before the early-stopping callback halts the training process. The training of the Inception-v3 architecture (Figure 4.5) presented difficulties, as the network was unable to learn how to classify the data correctly. The

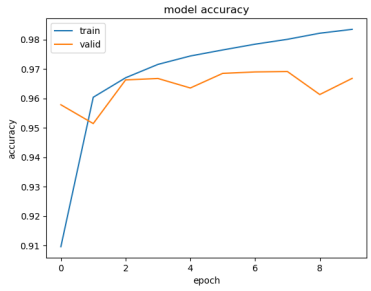

Figure 4.4: *VGG-16 training and validation accuracy with transfer learning.*

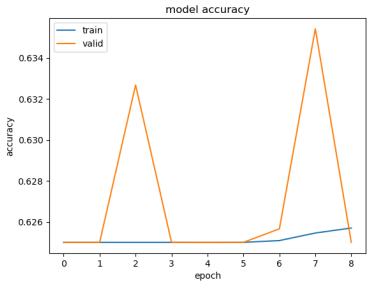

Figure 4.5: *Inception-v3 training and validation accuracy with transfer learning.*

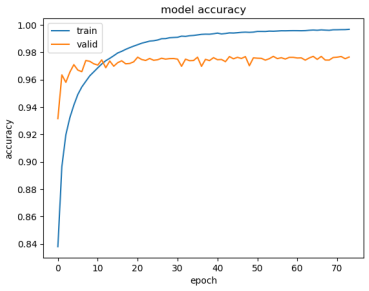

Figure 4.6: *DenseNet-121 training and validation accuracy with transfer learning.*

graph shows unstable peaks of validation accuracy in terms of accuracy on the validation set. The DenseNet-121 model (Figure 4.6) took the longest to train (over 70 epochs), taking about 1 hour for the completion of each epoch. This model showed a clear sign of overfitting, but the validation and training accuracies were almost parallel in the graph, which could be a sign of an unrepresentative training set.

4.2.2 Test results

After completing the training process, we evaluate the different architectures on the test set. The results obtained are shown in Table 4.2.

Table 4.2: *Accuracy on test set for each architecture.*

Architecture	Accuracy
VGG-16	96.81%
Inception-v3	36.82%
DenseNet-121	96.82%

As expected, given the test results, the accuracy of the Inception-v3 stands out as being very poor. Apart from this outlier, the other architectures displayed very good accuracy on the test set, both achieving very similar results.

4.3 Comparing experiments

In classification problems, there are a few metrics to measure the performance of a given classifier. The precision p of a class is the number of True Positives (*i.e.* the images correctly classified in this class) over all True Positives and False Positives (*i.e.* the images incorrectly classified as belonging to this class). The recall r of a class is the number of True Positives over all all True Positives and False Negatives (*i.e.* the images incorrectly rejected as belonging to this class). The F1-score F_1 of a class is the harmonic mean of precision p and the recall r. These metrics can be defined as follows (Olson and Delen 2008):

$$p = \frac{TP}{TP + FP} \tag{4.1}$$

$$r = \frac{TP}{TP + FN} \tag{4.2}$$

$$F_1 = 2. \frac{. r}{p + r} \tag{4.3}$$

For each experiment and architecture, these metrics were obtained by using the trained model and generating a classification report with the data from the test set. The reports are shown in Table 4.4.

Table 4.3: *Precision (p), Recall(r) and F1-score(F_1) for each architecture and experiment (E).*

Architecture	E	p	r	F_1
VGG-16	1	0.96	0.96	0.96
	2	0.97	0.97	0.97
InceptionV3	1	0.96	0.96	0.96
	2	0.25	0.37	0.20
Densenet121	1	0.96	0.96	0.96
	2	0.97	0.97	0.97

Most architectures had similar classification performance, and the main difference found was a slight improvement by using transfer learning instead of learning from scratch. The only clear outlier was the Inception-v3 model on the second experiment, which was not unexpected given previous results.

It is also interesting to compare the results obtained in this study to previous methods and other Deep Learning approaches. To do so, the best results from each experiment are shown alongside the other models on the table below:

Table 4.4: *Precision (p), Recall(r) and F1-score(F_1) for the best versions of each experiment in comparison with other classification methods.*

Method	p	r	F_1
Hoyos *et al.* (2012)	0.92	0.29	0.44
Goulding *et al.* (2017)	0.75	0.90	0.82
Ackermann *et al.* (2018)	0.96	0.97	0.97
Experiment 1	0.96	0.96	0.96
Experiment 2	0.97	0.97	0.97

Keeping in mind that only the recall r is the only invariant quantity between all the methods, because different authors used different class ratios to balance the data, it is the

most reliable comparison parameter. But even so, the comparison with the results found by Ackermann *et al.* (2018) can be done for all parameters, since the same dataset and a similar balance for the classes were used.

In comparison with Hoyos *et al.* (2012), that used an automatic classification process based on the morphological properties of the residual images of galaxies, the Deep Learning models in this study clearly show improvements. This holds specially for the detection of False Negatives, in this case images that are not galaxy mergers but were incorrectly classified as merging, as can be seen by comparing the recall metric.

The Machine Learning random forest decision tree technique used by Goulding *et al.* (2017) was also outperformed by our results, what can be clearly seen by analyzing the recall metric. In general, using Deep Learning to detect galaxy mergers showed better results overall than any other technique previously used for this problem.

Chapter 5

Conclusion

The results presented on Chapter 4 showed that a high accuracy can be achieved by using multiple Deep Learning techniques and architectures in this dataset. The performance obtained for detection of galaxy mergers in this study is a significant improvement over previous methods for automatic visual classification.

It is also worth noting that by using transfer learning from the *Imagenet* dataset to the images of merging galaxies there was a slight increase in performance. But considering the results of both experiments, it is safe to say that both approaches yielded good results for this dataset.

This work showed that the results achieved by Ackermann *et al.* (2018) can be reproduced and even slightly improved with alternative architectures and regularization techniques. Assuming that the dataset used is representative of images of merging galaxies generated by photometric surveys, then this study presents a reliable approach to detect galaxy mergers and could also possibly serve as a reference for other classification problems with astronomical imaging data.

Bibliography

Ackermann *et al.***(2018)** Sandro Ackermann, Kevin Schawinski, Ce Zhang, Anna K Weigel and M Dennis Turp. Using transfer learning to detect galaxy mergers. *Monthly Notices of the Royal Astronomical Society*, 479(1):415–425. Referenced on page 1, 13, 20, 21, 23

Bracewell and Bracewell(1986) Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York. Referenced on page 9

Conselice(2003) Christopher J Conselice. The relationship between stellar light distributions of galaxies and their formation histories. *The Astrophysical Journal Supplement Series*, 147(1):1. Referenced on page 4

Deng *et al.***(2009)** J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. Em *CVPR09*. Referenced on page 15

Glorot and Bengio(2010) Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. Em *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, páginas 249–256. Referenced on page 15

Goulding *et al.***(2017)** Andy D Goulding, Jenny E Greene, Rachel Bezanson, Johnny Greco, Sean Johnson, Alexie Leauthaud, Yoshiki Matsuoka, Elinor Medezinski and Adrian M Price-Whelan. Galaxy interactions trigger rapid black hole growth: An unprecedented view from the hyper suprime-cam survey. *Publications of the Astronomical Society of Japan*, 70(SP1):S37. Referenced on page 4, 20, 21

He *et al.***(2016)** Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep residual learning for image recognition. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 770–778. Referenced on page 11

Hebb(1949) Donald Hebb. The organization of behavior. emphnew york, 1949. Referenced on page 5

Hoyos *et al.***(2012)** Carlos Hoyos, Alfonso Aragón-Salamanca, Meghan E Gray, David T Maltby, Eric F Bell, Fabio D Barazza, Asmus Böhm, Boris Häußler, Knud Jahnke, Shardha Jogee *et al.* A new automatic method to identify galaxy mergers–i. description and application to the space telescope a901/902 galaxy evolution survey. *Monthly Notices of the Royal Astronomical Society*, 419(3):2703–2724. Referenced on page 4, 20, 21

Huang *et al.***(2017)** Gao Huang, Zhuang Liu, Laurens Van Der Maaten and Kilian Q Weinberger. Densely connected convolutional networks. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 4700–4708. Referenced on page 12

Hubble(1926) Edwin P Hubble. Extragalactic nebulae. *The Astrophysical Journal*, 64. Referenced on page 3

Karttunen *et al.***(2016)** Hannu Karttunen, Pekka Kröger, Heikki Oja, Markku Poutanen and Karl Johan Donner. *Fundamental astronomy*. Springer. Referenced on page 3

Keskar *et al.***(2016)** Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016. Referenced on page 15

Kingma and Ba(2014) Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. Referenced on page 9

Krizhevsky *et al.***(2012)** A. Krizhevsky, I. Sutskever and G. Hinton. Imagenet classification with deep convolutional neural networks. Em *Advances in neural information processing systems*, páginas 1097–1105. Referenced on page 10

LeCun *et al.***(1989)** Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551. Referenced on page 8

LeCun *et al.***(1995)** Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger *et al.* Comparison of learning algorithms for handwritten digit recognition. Em *International conference on artificial neural networks*, volume 60, páginas 53–60. Perth, Australia. Referenced on page 10

LeCun *et al.***(1998)** Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner *et al.* Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324. Referenced on page 10

Lin *et al.***(2004)** Lihwai Lin, David C Koo, Christopher NA Willmer, David R Patton, Christopher J Conselice, Renbin Yan, Alison L Coil, Michael C Cooper, Marc Davis, SM Faber *et al.* The deep2 galaxy redshift survey: Evolution of close galaxy pairs and major-merger rates up to z˜ 1.2. *The Astrophysical Journal Letters*, 617(1):L9. Referenced on page 4

Lin *et al.***(2008)** Lihwai Lin, David R. Patton, David C. Koo, Kevin Casteels, Christopher J. Conselice, S. M. Faber, Jennifer Lotz, Christopher N. A. Willmer, B. C. Hsieh, Tzihong Chiueh and et al. The redshift evolution of wet, dry, and mixed galaxy mergers from close galaxy pairs in the deep2 galaxy redshift survey. *The Astrophysical Journal*, 681(1): 232243. ISSN 1538-4357. doi: 10.1086/587928. URL http://dx.doi.org/10.1086/587928. Referenced on page 1

Lintott *et al.***(2010)** Chris Lintott, Kevin Schawinski, Steven Bamford, Ane Slosar, Kate Land, Daniel Thomas, Edd Edmondson, Karen Masters, Robert C. Nichol, M. Jordan Raddick, Alex Szalay, Dan Andreescu, Phil Murray and Jan Vandenberg. Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies*. *Monthly Notices of the Royal Astronomical Society*, 410(1):166–178. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2010.17432.x. URL https://doi.org/10.1111/j.1365-2966.2010.17432.x. Referenced on page 13

Lotz *et al.***(2004)** Jennifer M. Lotz, Joel Primack and Piero Madau. A new nonparametric approach to galaxy morphological classification. *The Astronomical Journal*, 128(1):163182. ISSN 1538-3881. doi: 10.1086/421849. URL http://dx.doi.org/10.1086/421849. Referenced on page 1, 4

Maas *et al.***(2013)** Andrew L Maas, Awni Y Hannun and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. Em *Proc. icml*, volume 30, página 3. Referenced on page 7

McCulloch and Pitts(1943) Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4): 115–133. Referenced on page 5

Minsky and Papert(1969) M. Minsky and S. Papert. Perceptrons, 1969. Referenced on page 6

Nair and Hinton(2010) Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. Em *Proceedings of the 27th international conference on machine learning (ICML-10)*, páginas 807–814. Referenced on page 7

Ng(2004) Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. Em *Proceedings of the twenty-first international conference on Machine learning*, página 78. ACM. Referenced on page 15

Nielsen(2015) Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:. Referenced on page 5, 8

Nolan(2010) Christopher Nolan. *Inception*. Warner Bros. Referenced on page 11

Olson and Delen(2008) David L Olson and Dursun Delen. *Advanced data mining techniques*. Springer Science & Business Media. Referenced on page 19

Rosenblatt(1958) F Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386. Referenced on page 5

Rumelhart *et al.***(1988)** D. Rumelhart, G. Hinton, R. Williams *et al.* Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1. Referenced on page 6

Simonyan and Zisserman(2014) Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. Referenced on page 11

Srivastava *et al.***(2014)** Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958. Referenced on page 15

Szegedy *et al.***(2016)** Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens and Zbigniew Wojna. Rethinking the inception architecture for computer vision. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 2818–2826. Referenced on page 11

Toomre and Toomre(1972) A. Toomre and J. Toomre. Galactic bridges and tails. *Astrophys. J.*, 178:623–666. doi: 10.1086/151823. Referenced on page 1