SAPIENZA
UNIVERSITÀ DI ROMA

Bot Detection Leveraging Image Techniques

Faculty of Information Engineering, Computer Science and Statistics
Master's Degree in Computer Science

Edoardo Di Paolo
ID number 1728334

Advisor
Prof. Angelo Spognardi

Academic Year 2021/2022

Bot Detection Leveraging Image Techniques
Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: dipaolo.1728334@studenti.uniroma1.it

A me.

Abstract

In the last decade there has been an evolution of Online Social Networks (OSNs). These OSNs allow users to discuss trending topics, exchange opinions and to interact with many other people but they also allowed the sharing of fake news.
The spreading of fake news has grown exponentially; this has directly affected part of society with changes of opinion about important topics. Often the spreading of fake news occurs as a "mass sharing" carried out not by real human account, but by many bots. In this study, a bot is also referred as a "Twitter bot" that is an account that performs actions such as tweeting, re-tweeting, following and liking automatically.
In this thesis, existing approaches regarding the bot classification will be shown and a new method of classification using images is presented. The images created are based on the users' DNA sequences that are strings based on a pre-selected alphabet. After the images were generated, a transfer learning approach with different pre-trained models was used to evaluate the effectiveness of the proposed method. The results obtained by this new approach are in line with those of the state of the art and, in some cases, are even better.

Contents

Chapter 1

Introduction

In the last decade there has been an evolution of Online Social Networks (OSNs). These OSNs allow users to discuss trending topics, exchange opinions and to interact with many other people. Some examples of OSNs are Twitter, Facebook and Instagram. In this thesis Twitter will be taken as the OSN of reference.

In parallel with the growth of these OSNs, the spreading of fake news has grown exponentially; this has directly affected part of society with changes of opinion about important topics. Often the spreading occurs as a "mass sharing" carried out not by real human account, but by many **bots**. A bot is a software that runs automated tasks over the Internet; in this thesis a bot is also referred as a "**Twitter bot**" that is an account that performs actions such as tweeting, re-tweeting, following and liking automatically. In recent years, the topic of bots on OSNs has risen in both the academic research world and the media world and, for example, the **$44 billion** deal to acquire Twitter has fallen through due to the unknown number of bots that are on the social. Fukuda *et al.* estimated, with a method based on a random walk, that, in 2021, the bot population on Twitter is between 8% and 18% [14]. However, these results are hard to obtain since Twitter's APIs are limited and do not offer a call that indicates whether a user is a bot or not; thus, it is impossible to report an exact percentage.

With the evolution of OSNs, the bot classification has become a crucial task; for example, *Antenore et al.* [3] analyzed the proliferation of fake news during the COVID-19 pandemic and some bot detection models.

The study done in this thesis aims to provide a possible new functional

approach to bot classification leveraging the remarkable advancements in image recognition. Indeed, as far as this task is considered, there is no approach that uses images. Nowadays, image classification is a widely studied and used task, and thanks to convolutional neural networks, it is possible to achieve amazing results. The general idea in the thesis is to exploit the potential of CNNs to classify accounts by passing in input images representing users. In fact, the assumption is that images of bot accounts are similar to each other and different from those of real accounts. The proposed novel approach is built on user extracted digital DNA sequences. After having all the DNA sequences, based on a predetermined alphabet, each symbol is assigned to a color and then each image's pixel represent one symbol of the sequence. Finally pre-trained networks were used over the generated images. Some examples of CNNs used are the ResNet50, VGG16 and WideResNet50. A further step was to consider the features set of an account along with the images in order to improve the promising results; therefore, it was tried to combine DNA-based images with account features. *Sun et al.* [30] proposed the SuperTML method, which uses the idea of Super Characters [29] and two-dimensional embeddings to address the problem of classification on tabular data. It was exploited in the bot classification task to check whether it works or not.

1.1 Thesis structure

In **Chapter 2**, existing approaches employed in bot/human classification of Twitter accounts are analyzed. In addition, there will be references to papers regarding image recognition in general.
In **Chapter 3**, the novel approach with a background is proposed. Moreover it is presented the theoretical and practical implementation.
In **Chapter 4**, the benchmarked datasets are presented and the results obtained are reported.
In **Chapter 5**, there are the conclusions and some possible future directions regarding the bot detection.

Chapter 2

Related work

In this chapter will be presented some of the most popular methods used in the bot classification task. These methods will be analyzed and will be reported some of the results that the authors obtained on the datasets that they benchmarked. In addition, there is a section regarding image recognition in which some of the most important papers are reported.

2.1 Bot detection

The bot detection problem has been going on for more than a decade. In fact, the exponential growth in the use of OSNs has made bots a problem to be solved. One of the first attempts, if not the first, reported in the literature is presented in [39] in which the authors tried to detect spammers on Twitter based on the individual charactersitics of the users. Approaches based on supervised learning have been developed over the years, and since the thesis work is also based on this, some related techniques will be seen more in detail in this chapter. However, there are approaches based on unsupervised learning. In these the aim is to detect not only a user as a bot, but rather to detect groups of bots. *Chavoshi et al.* [5] proposed a bot detection system that works on activity correlation without requiring labeled data. This approach detects thousands of bots per day with an high precision equal to 94%.

One of the problems related to supervised learning is that there is a need to manually label accounts in the dataset. It is also increasingly complicated to recognize some bot accounts since they are more and more similar to valid accounts. So, it has become even more important to create approaches that can adapt to this continuous evolution as proven by the new TwiBot20 [12] dataset, published in 2020: approaches that on previous datasets had 90%

of accuracy, on TwiBot20 only achieve 70%. The largest dataset concerning Twitter is the one presented in [13]. It is a graph-based dataset with more than 1 million of accounts labelled as human or bot. Usually, bot detection datasets rely on manual annotation or crowdsourcing such as in the work of *Gilani et al.* [15] which assigned with a human annotation task the groundtruth for each sample. In TwiBot22 the authors in a first moment invited bot detection experts to annotate 1000 accounts and then, with the help of some model, they generated the annotations with the help Snorkel [25].

Antenore et al. [3] analyzed the proliferation of fake news during the COVID-19 pandemic and compared some bot detection models. The results of the different approaches reported in this study will be taken as reference in this thesis.

The next sections will report some of the approaches related to the digital DNA, which have been widely used, and other approaches from which some ideas have been taken. Moreover, some of the most important work on image recognition are reported.

2.1.1 DNA approaches

A new and innovative approach was proposed by *Cresci et al.* in [8]. The study done in this paper is the basis for the work done during the thesis period.

The biological DNA contains the genetic information of a living being and is represented by a sequence which uses 4 characters representing the four nucleotide bases: A (*adenine*), C (*cytosine*), G (*guanine*) and T (*thymine*). The **digital DNA** is the counterpart of biological DNA and, in this case, represents the actions of a given Twitter account. To give a general idea of the proposed study, an account's digital DNA is a sequence consisting of L characters taken from a predefined alphabet. The latter is formally defined as $\mathbb{B}$ and it contains different symbols; mathematically $\mathbb{B}$ is defined as follows:

$$\mathbb{B} = \{\ \sigma_1, \sigma_2, \sigma_3, ..., \sigma_N\},\ \sigma_i \neq \sigma_j\ \forall i, j = 1, ..., N\ \wedge i \neq j \tag{2.1}$$

In Equation 2.1 each σ is a symbol of the alphabet and a digital DNA sequence will be defined as follow:

$$s = (\sigma_1, \sigma_2, ..., \sigma_n),\ \sigma_i \in \mathbb{B}\ \forall\ i = 1, ..., n \tag{2.2}$$

Each symbol in the sequence denotes a type of actions; in fact, it is possible to

define, for example, an alphabet based on the types of tweet:

$$\mathbb{B}_{\text{type}} = \left\{ \begin{array}{c} A = \text{tweet}, \\ C = \text{reply}, \\ T = \text{retweet} \end{array} \right\} \tag{2.3}$$

and an example of sequence could be $s = ACCCTAAACCCCCCCTT....$ In this way, the account information is represented by a compact notation. Then, they leveraged the LCS (*longest common substring*) curves to detect social spambots and they reached very good results, like MCC of 0.952 and 0.955 on a dataset with genuine accounts and retweeters of the political candidate, and MCC of 0.867 and 0.940 on a dataset with genuine accounts and spammers of *Amazon* products.

Another study based on the idea of digital DNA sequence is proposed by *Gilmary et al.* [16], who splitted the sequences into fragments and then calculated the entropy through tf-idf for all unique DNA fragments. They reported that this approach achieved an average accuracy of 0.9357 on a new real-world Twitter dataset containing 2000 accounts with 994 bots and 1006 humans.

2.1.2 Other approaches

Hayawi et al. [17] proposed in their study a new framework called "DeeProBot" which stands for *Deep Profile-based Bot* detection. They used only the user profile-based features (i.e. the username's length and the number of followers) in order to classify accounts with a LSTM (*long short-term memory*) network. This approach achieved 0.97 of AUC as the best result.

Another work, presented by *Shaghayegh et al.*, uses a *generative adversarial network* associated with an LSTM network. In this approach, they used GANs to generate bot samples to obtain more information about their behavior. The results achieved by this approach are 0.951 of accuracy, 0.932 of recall, 0.987 of precision and 0.958 of F1 score on the *Cresci et al.* [7] dataset.

There are also approaches based on *graph neural networks*. These techniques interpret a social network as graphs: for example, two users are two nodes in a graph, and the edge would be the "following" relationship between them. *Alhosseini et al.* [1] assumed that in addition to the account features, the social graph must be taken in consideration so that we can find possible hidden patterns between bots. They obtained, on the dataset in [35], an area below the ROC curve of 94% which is 8% and 16% higher than the multi layer

perceptron and belief propagation methods that they compared.

Efthimion et al. [10] proposed an approach to identify Twitter bot accounts. They used complex machine learning algorithm exploiting a range of features such as the length of usernames, the reposting rate, some temporal patterns and the similarity of message contents based on Levenshtein distance.

Botometer [37] is an online tool that determines a probability that a given account is a bot or not. Since 2016, it is also considered among the state of the art bot detection models; it exploits 1000 different features and it is trained on different type of data. One of the sources was collected by *Lee et al.* [22] who used different Twitter honeypots that resulted in the harvesting of 36.000 possible bots.

Yang et al. [36], through an empirical study, proposed a list of possible features to exploit for the bot detection and some of them will be used in the experiments done during the thesis.

Another approach, called RoSGAS (Reinforced and Self-supervised GNN Architecture Search), is presented in [38]. In this case the authors proposed an approach based on a multi-agent deep reinforcement learning. There are also approaches in the literature based on NLP, *Natural Language Processing*. The purpose of NLP is to make a computer able to understand the content of any text, and research in this field is widely advanced. In [21] the authors proposed a deep neural network based on a LSTM architecture that exploits also the text of tweets. Often, the textual content of tweets created by bot accounts is similar to each other, and therefore, analysis of these can help the account classification. However, as NLP research as evolved, bot texts have also evolved in terms of complexity and meaningfulness. Also in [33] the authors proposed a text-based approach in which they used a bidirectional LSTM and achieved good results on the Cresci-2017 dataset: 0.94 in precision, 0.97 in recall, 0.96 in accuracy and 0.96 in F1 score.

2.2 Images recognition

One of the areas of computer science that has evolved the most in the past years is definitely what concerns image recognition. For humans, images are a powerful tool for communication, and therefore, making a computer efficent and accurate in image recognition is an important task. Image recognition is a part of the computer vision that deals with the identification and classification of objects in images. In particular, the goal of image recognition is to assign

categories to images.

The first attempts to recognize letters, numbers and symbols dates back as far as the 1960s [2]. With the advancement of technology, it was possible to create models that are able to recognize objects in images and much more.

One of the first model developed for image recognition was the AlexNet [20] in 2012. *Krizhevsky et al.* trained a large, deep neural network to classify the ImageNet [9] dataset and they improved the state of the art. ImageNet is an image database with 1000 different categories that is used as a benchmark for convolutional neural networks (CNNs). These CNNs are discussed in detail in [24] and in the next chapter of this thesis.

Over the years, new architectures have been proposed that have further improved the results[1] on ImageNet.

VGG [27] is a convolutional network architecture proposed in 2015. The authors investigated the effect of the convolutional network depth on its accuracy on the ImageNet dataset and proposed an architecture with very smal convolutional filters. They reached 0.745 in accuracy.

InceptionV3 [31] is an architecture proposed in 2015 and it was also benchmarked on ImageNet. *Szegedy et al.* explored ways to scale up the architectures withouth burdening the performance of the models. In this case they improved the state-of-the-art at that time reaching an accuracy equal to 0.78.

Other improvements were achieved through Residual Networks [18] in which the accuracy on ImageNet improved to 0.809. This work is one of the most important in the field of computer vision since the authors reformulated the layers as learning residual functions.

In 2019 EfficientNet [32], another architecture, reached 0.869. *Tan et al.* proposed a new scaling method that uniformly scales all dimensions of depth, width or resolution using a simple compound coefficient ϕ.

As seen, there are many deep learning models in the literature that have achieved amazing results in image recognition. However, one of the main problems that plagues these architectures is the overfitting [40]. Overfitting occurs on limited datasets since the model fails to generalize the data. To avoid this problem, there is the Data Augmentation [26]. The latter consists of a series of algorithms that allow for enhanced training on the data that a network uses to learn.

The study of this thesis made use of the transfer learning, which is discussed more in detail in the next chapter. *Weiss et al.* [34] formally defined the transfer

[1] List of benchmarks on ImageNet.

learning and reported some possible scenario in which it is useful to exploit the potential of this technique. Moreover, the transfer learning helps to prevent overfitting during the training [26].

2.3 Observations

The approaches regarding the bot detection, which are presented in this Chapter, are certainly valid. However, with the evolution of Computer Science in fields such as artificial intelligence, some models have become obsolete since they can no longer detect evolved bots. One of the most important works in bot detection is definitely the digital DNA [8], from which all the work done during the thesis period is based. The LCSs curves, used in digital DNA, struggle to scale in datasets such as [12, 13] since there are too much samples. This problem, through the use of images, does not arise since image recognition can also be done on millions of images.

In bot detection there is a wide use of pure machine learning [10, 36, 37]. As bots evolve, considering only a set of features and creating a machine learning model is no longer sufficient. It is more and more important to consider the relationships between users [1]. Graph-based approaches, in future, will always get very good results, and research in general is moving toward this type of approach.

As for text-based approaches [21, 33], these will also need to be updated. The texts of bot-generated tweets nowadays can be mistaken for those of real accounts. Therefore, it will be necessary to integrate the use of, for example, features regarding users to these approaches.

Chapter 3

A novel approach

In this chapter the approaches developed during the thesis period will be reported. In addition, the fundamental concepts behind the work done will be explained.

3.1 Background

The approaches proposed in this thesis make use of the *Convolutional Neural Networks* (CNNs). These CNNs have had remarkable advancements in recent years, that is why they are exploited in the thesis.

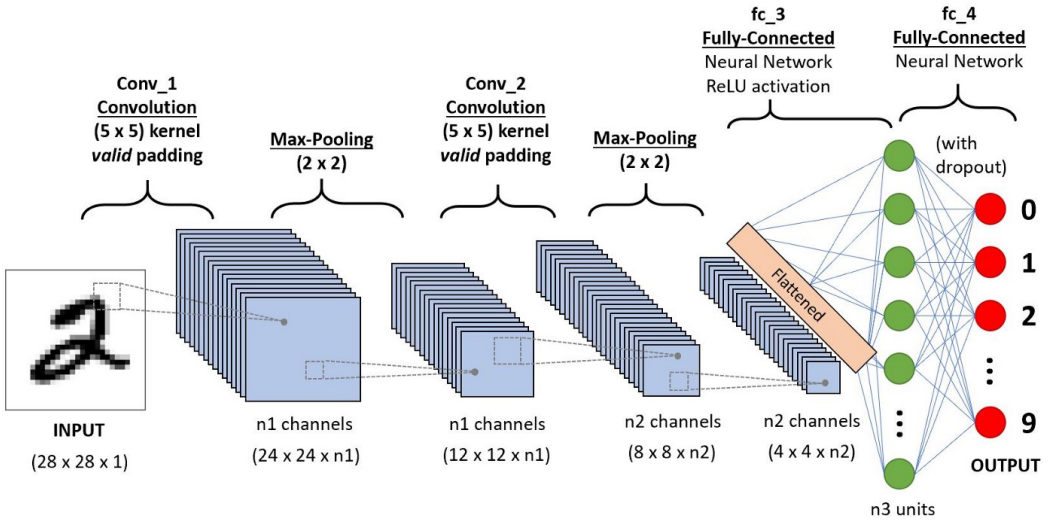


Figure 3.1. CNN architecture example.

Figure 3.1 shows an architecture for a CNN. In this example, the neural network is trained to classify handwritten digits into 10 different classes ranging

from 0 to 9. The input, as it is possible to see, is an image with 28x28 as width and height and with 1 dimension, so it is in gray-scale; if it was an RGB image there would be 3 dimensions: one for red, one for green and one for blue. This image will pass different **convolutional layers**. Convolutional layers make use of what is called **convolution** that is a mathematical operation:

$$(f * g)(t) = \int f(\tau)g(t-\tau)\, d\tau \tag{3.1}$$

In Equation 3.1, f and g are two functions and the results of the convolution between these two will be a third function. This new function will be the integral of the product of the two functions after one is reflected and shifted. In CNNs, it is possible to refer to the discrete version of the convolution that is defined as follow:

$$f[m,n] = \sum_{k,l} I[m-k, n-l]g[k,l] \tag{3.2}$$

In Equation 3.2 f will be the "filtered" image and I is the image given in input to the layer. The function g represents the **kernel**; it is a small matrix used for example to blur the image in input.

(a) Original Lena image **(b)** Lena blurred image **(c)** Lena sharpened image

Figure 3.2. Examples of kernel functions applied to a 2D image.

In Figure 3.2 it is possible to see some kernel applied to the first image, and they are the following:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(a) Blur kernel matrix. **(b)** Sharpen kernel matrix.

When the input is processed by a convolutional layer, it is possible to add an amount of pixels; this is the **padding**. In addition to the padding,

it is possible to specify the **stride** parameter, that represents the amount of movement over the input. The size of output from a convolutional layer is given by the following formula:

$$W_{out} = \frac{W - K + 2P}{S} + 1 \tag{3.3}$$

In Equation 3.3, W is the width (or the height if $W = H$), K is the kernel size, P the padding and S is the stride.

After the convolutional layer, in Figure 3.1, there is a **pooling layer**. The pooling layer is used to downsample the given input and there are different pooling functions that can be used in this step.

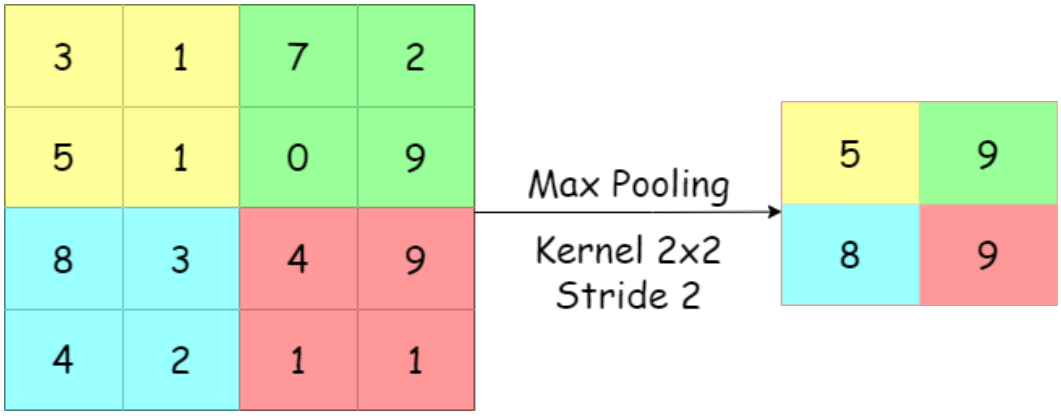


Figure 3.4. Max pooling layer example.

Figure 3.4 is an example of the **max pooling** with a kernel with 2 x 2 as dimensions and with a stride of 2. As it is possible to see, the left table is the input and the output will be a downsampled table with the max values in each 2x2 table. The size of the output is determined by the following formula:

$$W_{out} = \frac{W - K}{S} + 1 \tag{3.4}$$

In Equation 3.4, W is the width (or the height if $W = H$), K the kernel size and S the stride.

At the end, there are the **fully connected** layers and they are the last layers in the entire architecture. In this step, it is possible to use some activation function like the ReLU or the softmax.

$$\begin{aligned} &\text{ReLU: } f(x) = \max(0, x) \\ &\text{softmax: } \sigma_j(z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \ \forall j = 1, ..., K. \end{aligned} \tag{3.5}$$

Another technique used in this study, is the **transfer learning**. Training a CNN from scratch, initially with randomly weights, is a difficult process since it requires high-performance hardware. In order to avoid the step of training from scratch, it is common to use some **pre-trained** networks on a very large dataset such as ImageNet [9] which has more than 1 million of images and 1000 different categories.

In this work the transfer learning technique was used in two ways:

- **fixed feature extractor**: in this case the last fully connected layer of the network is replaced by a layer that has in input the number of features in output from the previous layer and 2 (the number of classes needed). The weights are freezed for all the layers except for the last one;
- **fine tuning**: this is similar to the previous one, but here the weights are not freezed and they are "fine-tuned" during the training on the custom dataset.

These ways of proceeding were tried during the thesis work and will be discussed in the following sections in detail.

3.2 Pre-trained networks

In this section, the pre-trained networks used during the thesis will be presented.

Inception_V3 This neural network is also known as "GoogleNetv3" [31] and is trained on ImageNet. The architecture is in Table 3.1. The structure of an inception block is represented in Figure 3.5.

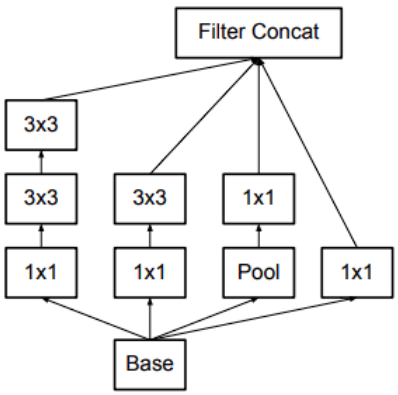


Figure 3.5. Inception block structure.

layer type	**patch size/stride** or remarks	**input size**
conv	3x3/2	299x299x3
conv	3x3/1	149x149x32
conv padded	3x3/1	147x147x32
pool	3x3/2	147x147x64
conv	3x3/1	73x73x64
conv	3x3/2	71x71x80
conv	3x3/1	35x35x192
3 x Inception block	3x3/1	35x35x288
5 x Inception block	3x3/1	17x17x768
2 x Inception block	3x3/1	8x8x1280
pool	8x8	8x8x2048
linear	logits	1x1x2048
softmax	classifier	1x1x1000

Table 3.1. Inception V3 architecture.

ResNet Multiple models based on the ResNet architecture were tested during the thesis, and these are: **ResNet50**, **WideResNet50** and **ResNet152**. The ResNet model is based on the work presented in [18] in which the authors proposed a residual learning framework to ease the training of networks with many layers. These residual networks, typically, have layers that contains the ReLU (Equation 3.5) as activation function and some batch normalization layer which is useful to speed up the training phase and to provide some regularization. In Figure 3.6 it is possible to see a regular block, the left one, in which the mapping function $f(x)$ is directly learned inside the dotted line box while, the right one, is a residual block used in ResNets and here $f(x) = g(x) + x$.

VGG16 Another model used in this study is the VGG16 proposed in [27]. This model is trained on ImageNet and in Figure 3.7 it is possible to see the network's architecture in which there are convolutional layer with the ReLU, max pooling layers, fully connected layers and at the end the softmax.

EfficientNet The last architecture used in this study is an EfficientNet. They were proposed in [32] and they achieved the state-of-the-art accuracy.

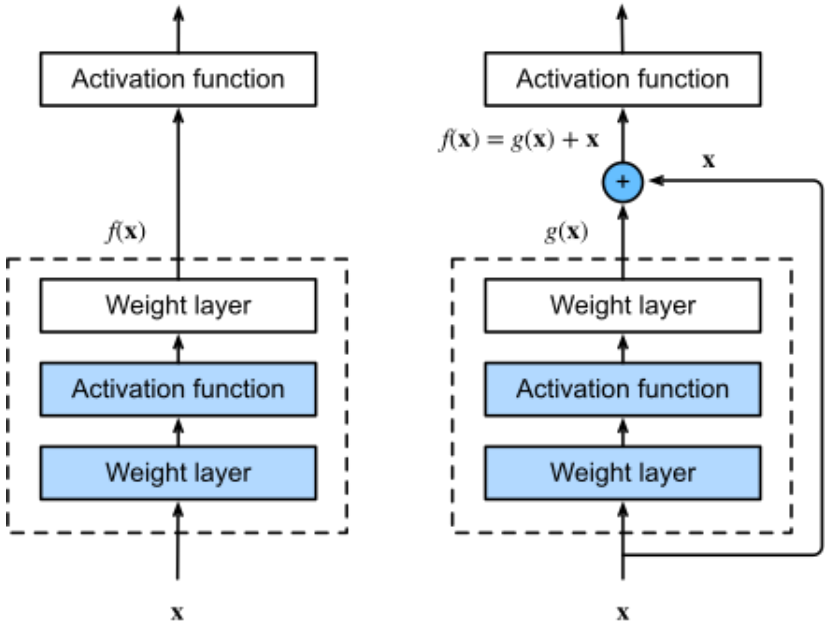


Figure 3.6. A regular block (left) and a residual block (right).

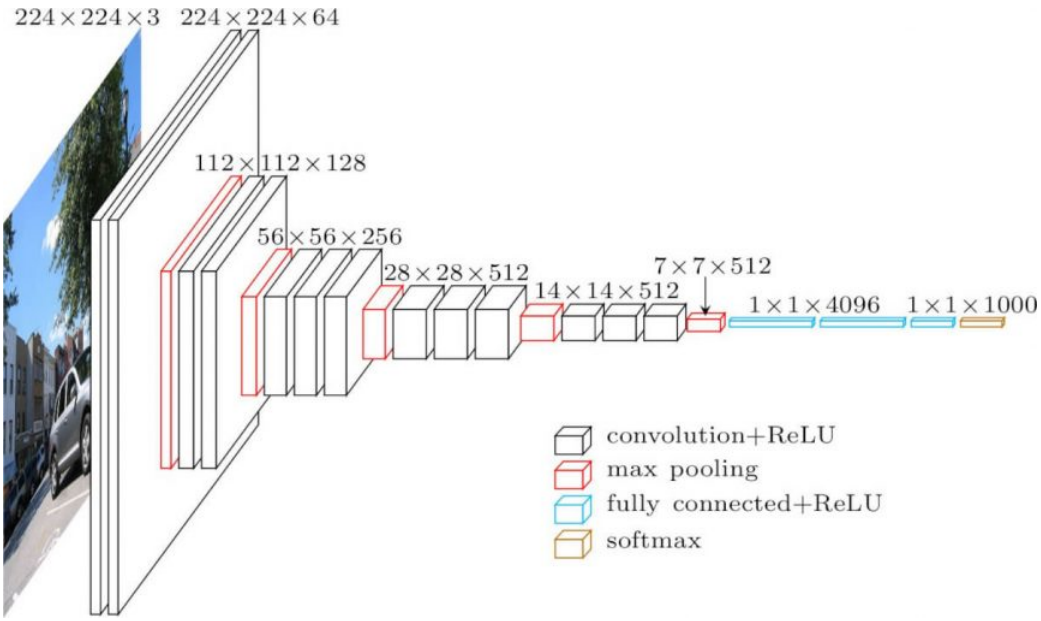


Figure 3.7. VGG16 architecture.

3.3 Images classification for bot detection

In this section will be presented the novel approach proposed for the bot classification task. In particular, in this study the remarkable advancements in image classification have been exploited in order to correctly classify the bots.

Moreover, some python functions, used in images creation, will be described.

3.3.1 General idea

At the moment, there are no approaches in the literature that take advantage of images and of convolutional neural networks to classify bots. The idea behind all the work done during the thesis comes from the digital DNA [8]. This novel approach is based on transforming each user's DNA sequence into an image. The hypothesis is that the behaviors and actions of bots have certain patterns that are different from that of real accounts and that, through the use of images, it is possible to discover them.

Images algorithm

One of the difficulties encountered during the thesis work was figuring out an efficient algorithm that could transform a string into two dimensions. In particular, the aim was to create images that did not lose the value of DNA strings and, moreover, there are few DNA-to-image conversion algorithms in the literature. One of them is called "Chaos Game Representation" [19], but it was not fully exploited in the thesis since the preliminary results were not entirely sufficient.

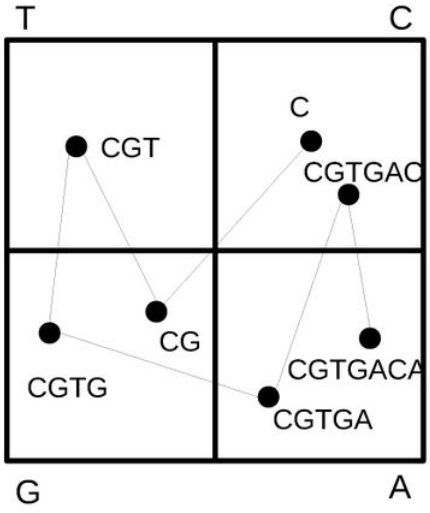
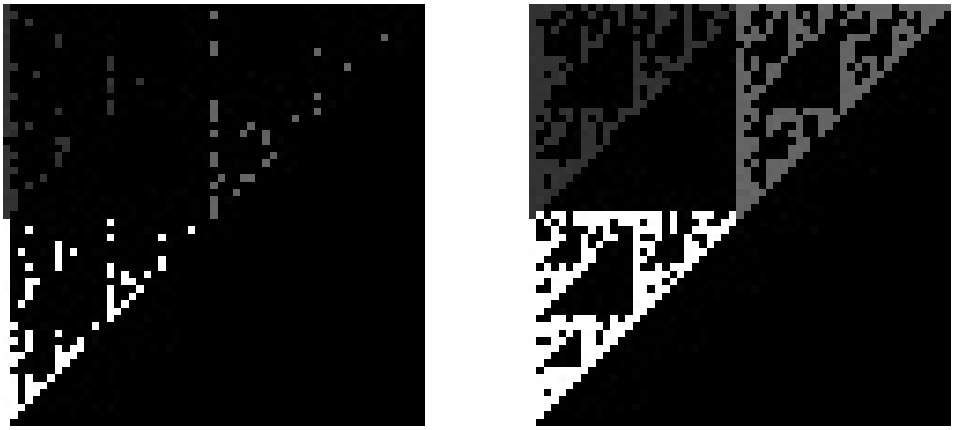


Figure 3.8. Chaos Game Representation algorithm.

Figure 3.8 shows how the algorithm works: each vertex of the image represents a letter. Suppose the first letter in our sequence is a C, as in the example, this will be placed halfway between vertex C and the center. The second letter in the example is a G, and this will be placed at half the distance

between the current point of C and the vertex of G. The algorithm will end once the entire sequence has been read.



(a) A human with CGR representation. **(b)** A bot with CGR representation.

Figure 3.9. Examples of the images generated with CGR.

In Figure 3.9 there are two images based on the "Chaos Game Representation" approach. As it is possible to see there is some pattern, but they are similar between bot and human and so a good classification will be too hard to obtain. Moreover, since the alphabet that is used in the DNA sequences has only 3 characters (A = tweet, C = reply and T = retweet), the images will be created based on only 3 vertices and not 4. However, with some changes in the algorithm, (i.e. being able to create images with more specific patterns, use a different alphabet), it can be better exploited.

Another algorithm tried to convert DNA sequences into images was the one presented by *Somodevilla et al.* in [28]. They proposed an approach in which there is a matrix of dimension NxN where N is the length of the DNA sequence and for each row there is the entire sequence. For example, suppose that there is the need to represent in image the following sequence: $ACTACT$. So the matrix will be:

$$\begin{bmatrix} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \mathbf{1} & A & C & T & A & C & T \\ \mathbf{2} & A & C & T & A & C & T \\ \mathbf{3} & A & C & T & A & C & T \\ \mathbf{4} & A & C & T & A & C & T \\ \mathbf{5} & A & C & T & A & C & T \\ \mathbf{6} & A & C & T & A & C & T \end{bmatrix} \tag{3.6}$$

Then, once a color has been assigned for each letter, the image is created. Unfortunately, this algorithm was not scalable in the case of bot classification; in fact, often there are DNA sequences longer than 1000 characters and thus would result in images with dimensions greater than 1000x1000. Obviously, such images require too much effort to be classified, so this algorithm was also discarded for the prosecution of this thesis.

The idea behind the algorithm used in this work comes from the two algorithms previously shown and it consists to "unroll" the string in 2 dimensions. In Algorithm 1 there is the pseudocode for the images creation. Since neural networks expect images with the same size, it was decided to take the string of maximum length and check whether this was a perfect square. In case it was not a perfect square, the closest and strictly largest perfect square would be calculated. By doing so, it was possible to represent all strings in images of equal size; in fact, strings as long as 10000 characters could be represented in images of size 100x100. After arbitrarily deciding on an RGB color to assign to each symbol in the alphabet, the image is colored pixel by pixel based on the current index. The coloring is done as long as the length of the input string is not exceeded; therefore, in case where the sequence is not the one with maximum length, it will result in a black part of the image. At the end, the function will return the image for the given DNA sequence. All images created are in grayscale; colored images were also tried but there was no major improvement in the final results. Since multiple datasets were benchmarked, images for each will be shown throughout this report. In order to avoid possible overfitting during the training of the CNNs, it was decided to use a validation dataset as well. In this way it was possible to check the validation and the training losses during the training.

Implementation of the approach

This section describes the technologies used for the implementation of the approach. In order to produce readable code, the PyTorch Lightning framework [11] was used with regard to neural networks. It is a fully flexible framework built on pure PyTorch and it allows to spend less time on engineering the code. In addition, the framework was used jointly with WanDB [4] to keep track of metrics.

One of the first classes defined in the framewok is the one concerning the dataset management and in Listing 1 there is the code. This class initialize the dataset and, in the `init` function, it applies the transforms (i.e. resizing,

Algorithm 1 Pseudocode images algorithm

Input: DNA sequence
Output: DNA image

```
n ← DNA string length
if n is a perfect square then
    L ← √n
else
    L ← get_closest_square_number(n)
end if
I ← create_image(width = L, height = L)
P ← dictionary with keys the alphabet symbols and RGB colors as values
for row in range(L) do
    for col in range(L) do
        k ← (row * L) + col
        if k < n then
            I[row, col] ← P[DNA[k]]
        end if
    end for
end for
return I
```

cropping and rotating). The `random_split_dataset` functions is a utility function in order to split the dataset in three different sub-datasets: the training dataset (60%), validation dataset (20%) and test dataset (20%). In the classification task, the label 1 is referring to bots while the label 0 to human accounts. In addition, a module that extends `LightningDataModule` has been defined to handle batches of the datasets[1] during the training phase.

In Listing 2 it is possible to see the core class of the implementation. Some lines of code, concerning the functions of the metrics, have been omitted for reasons of thesis formatting. The complete code is available on GitHub[2]. With this class, the framework performs the main steps and it performs the `forward` function. The functions `training_step`, `validation_step`, and `test_step` are used to perform training, validation, and testing of the dataset respectively. In each of these functions, loss computation takes place, and thus it is possible to understand whether the model is learning and classifying correctly. Regarding the loss function, the cross entropy was used during the work. It is used as measure of the classification model in which the output is a probability between 0 and 1. Since the bot classification is a binary classification, the cross entropy is calculated by the following formula:

[1]The code of this class is not reported in this relation, but it is available on GitHub.
[2]The full core class code is available here.

$$Loss = -(y \log(p) + (1 - y) \log(1 - p)) \tag{3.7}$$

In Equation 3.7, y is the binary indicator (bot or human) and p is the probability.

In addition, there is all the code regarding the metrics that are summarized in the next Chapter. The core class have also a function called `configure_optimizers`; it is used to perform optimization of the models. The optimizers used are available in the Colab Notebook available on GitHub[3]. However, they are the Stochastic Gradient Descent (SGD) and the Adam algorithm. In particular, parameters such as learning rate, momentum, weight decay, step size and gamma were configured. In the Notebook there are also all the datasets benchmarked that are reported in the next Chapter. In general, models could perform training for at most 50 epochs but it was difficult for them to complete all of them given the use of `EarlyStopping`. Through the latter, in fact, the accuracy (or loss) of the validation set was monitored, and in case it did not increase (or decrease in the case of loss) for a predetermined number of epochs, the training would stop.

[3]The notebook is available here.

Dataset class in PyTorch Lightning.

```python
class BotImagesDataset(Dataset):

    def __init__(self, dataset_dir: str, transforms: Optional[Compose] = None) -> None:
        self.dataset_dir: str = dataset_dir
        self.labels: List[int] = [1 if _.split("_")[0] == "bot" else 0 for _ in os.listdir(dataset_dir)]
        self.images: List[str] = [_ for _ in os.listdir(dataset_dir)]
        self.transforms: Optional[Compose] = transforms

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index: int) -> Union[str, Tuple[int, Tensor]]:
        if not 0 <= index < len(self.images):
            return "Please provide a valid index."
        image_path: str = os.path.join(self.dataset_dir, self.images[index])
        image: Image.Image = Image.open(image_path)
        label: int = self.labels[index]
        if self.transforms:
            image = self.transforms(image)
        return image, label

    def random_split_dataset(self, train_size: float = 0.6, test_size: float = 0.2, val_size: Optional[float] = 0.2, generator: Generator = default_generator) -> List[Subset]:
        train_size = int(len(self.labels) * train_size)
        test_size = int(len(self.labels) * test_size)

        if val_size is not None: val_size = int(len(self.labels) * val_size)

        if sum(filter(None, [train_size, test_size, val_size])) < len(self.labels):
            train_size += len(self.labels) - sum(filter(None, [train_size, test_size, val_size]))
        return random_split(self, generator=generator, lengths=[train_size, test_size, val_size])
```

Listing 1

BotClassificator class.

```python
class BotClassificator(pl.LightningModule):

    def __init__(self, hparams = {}, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        self.save_hyperparameters(hparams)
        str_to_model = {} # dictionary with all models
        self.model = str_to_model[hparams["model"]]()

    def forward(self, x: torch.Tensor):
        return self.model(x)

    def training_step(self, batch, batch_nb):
        x, y = batch
        y_hat = self(x)
        loss = F.cross_entropy(y_hat, y)
        y_hat = torch.softmax(y_hat, -1)
        output = torch.argmax(y_hat, dim=1)
        y = y.cpu().numpy()
        output = output.cpu().numpy()
        accuracy = accuracy_score(y, output)
        f1 = f1_score(y, output, average='macro')
        recall = recall_score(y, output, average='macro',
        ↪  zero_division=0)
        mcc = matthews_corrcoef(y, output)
        precision = precision_score(y, output, zero_division=0,
        ↪  average='macro')
        return loss

    def validation_step(self, batch, batch_nb):
        x, y = batch
        y_hat = self(x)
        loss = F.cross_entropy(y_hat, y)
        y_hat = torch.softmax(y_hat, dim=-1)
        output = torch.argmax(y_hat, dim=1)
        y = y.cpu().numpy()
        # calculation of metrics (like training_step)
        return loss

    def test_step(self, batch, batch_nb):
        x, y = batch
        y_hat = self(x)
        loss = F.cross_entropy(y_hat, y)
        y_hat = torch.softmax(y_hat, dim=-1)
        output = torch.argmax(y_hat, dim=1)
        y = y.cpu().numpy()
        output = output.cpu().numpy()
        # calculation of metrics (like training_step)

    def configure_optimizers(self):
        return self.model.configure_optimizers(self.hparams)
```

Listing 2

Chapter 4

Experiments and Results

In this Chapter, the datasets benchmarked and the results obtained will be presented. For each dataset there will be a description and, in addition, some of the images created will also be shown.

4.1 Metrics used

This section will report the metrics used to benchmark the different datasets tested.

Accuracy score The *accuracy* is the proportion of correct predictions, considering both true positives and true negatives, among the total number of samples. The formula used to calculate the accuracy is as follows:

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

where **TP** are the *true positives*, **TN** *true negatives*, **FP** *false positives* and **FN** *false negatives*.

Precision score The *precision* is the ability of the classifier not to label as positive a sample that is negative. The formula used to calculate the precision is as follows:

$$\frac{TP}{TP + FP} \tag{4.2}$$

Recall score The *recall* is the ability of the classifier to find all the positive samples. The formula used to calculate the recall is as follows:

$$\frac{TP}{TP + FN} \tag{4.3}$$

F1 score The *F1 score* is the harmonic mean of the precision and recall. The formula used to calculate the F1 score is as follows:

$$\frac{2 \cdot (precision \cdot recall)}{precision + recall} \tag{4.4}$$

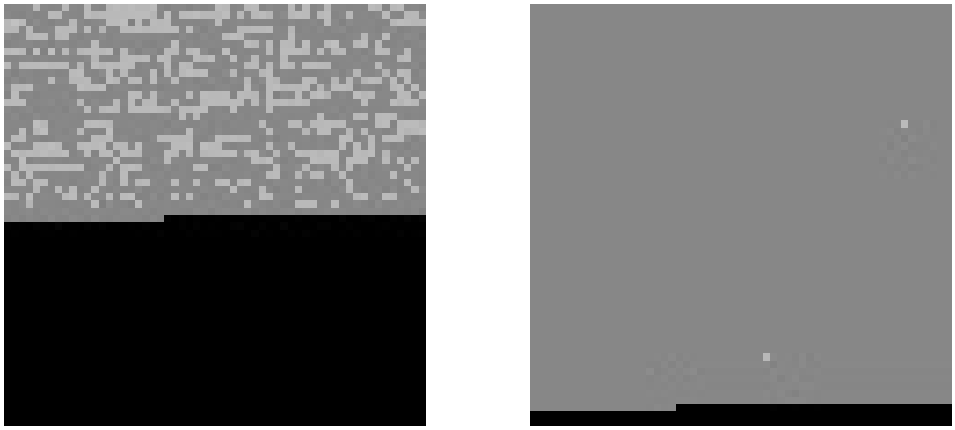
Matthews correlation coefficient The *Matthews correlation coefficient* (or φ coeffiecient) takes into account true and false positives and negatives and is regarded as a balanced measure which can be used even if the classes are of very different sizes. The formula used to calculate the φ coefficient is as follows:

$$\frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{4.5}$$

These metrics will be used to show the effectiveness of the approach proposed in this thesis in comparison with existing approaches.

4.2 Cresci 2017

It is a dataset presented in [7] and it contains genuine, traditional and social spambot Twitter accounts. For each user there is the corresponding label, human or bot. In Cresci 2017 there are 991 bots and 1083 users for a total of 2074 samples. Before moving on to image creation, there was the need to create the DNA sequences, and this function is in Listing 3. In this case, it was used the alphabet called "$\mathbb{B}_3$ *type*". After all the DNA strings were generated, the Algorithm 1 was used to create the images.

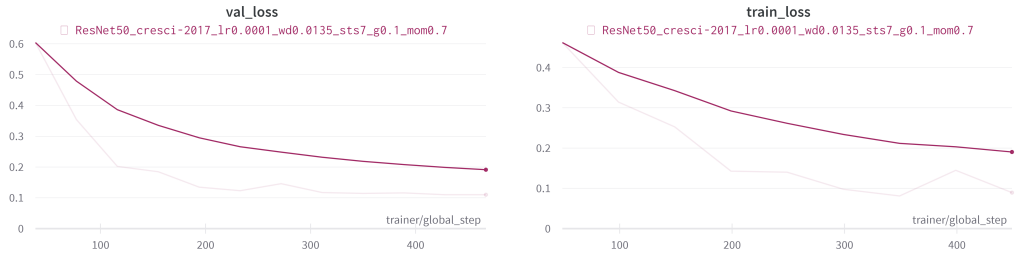


(a) A human account in image format. **(b)** A bot account in image format.

Figure 4.1. Examples of the images generated for Cresci 2017.

As it is possible to see, in Figure 4.1 there are two images: the left one represents a human account, while the right one represents a bot account. It is clearly visible that there is some kind of noise in Figure 4.1a that distinguishes this account from that of a bot. The idea of the thesis is that a CNN would be able to pick up these differences and thus be able to classify correctly accounts. In [12] there are some results obtained on this dataset and they are: 0.98 in accuracy, 0.98 in F1 score and 0.96 in the φ coefficient. With the approach proposed in this thesis, the following results were obtained: **0.98** in accuracy, **0.98** in recall, **0.98** in F1 score and **0.98** in φ coefficient. The best model for this dataset was the ResNet50, in which the loss decreased to 0.114.

Figure 4.2 shows the validation and training losses: as it is possible to see the two losses have a similar behavior and they decrease until they stabilize for a predetermined number of epochs. Since the model behaves similarly in both validation and training set, there is no overfitting. However, the results obtained with this dataset are in line with those of the state of the art and, in the case of the φ coefficient, are even better.



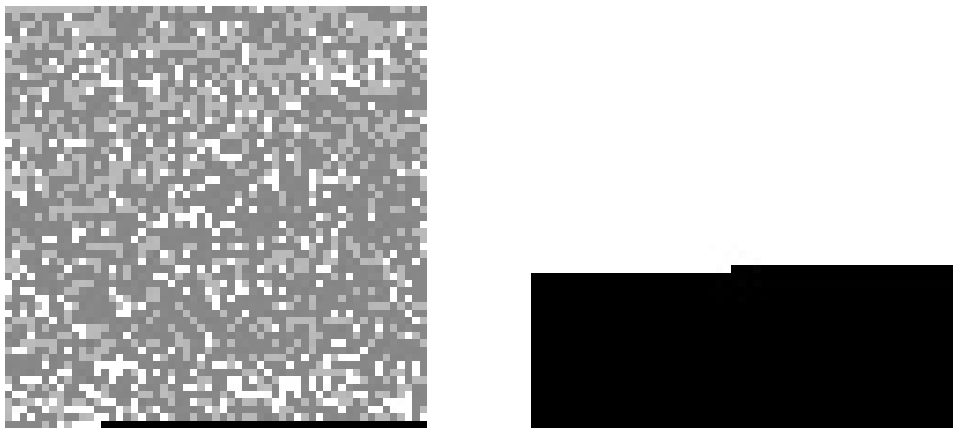
(a) Validation loss on Cresci 2017. **(b)** Training loss on Cresci 2017.

Figure 4.2. Train and validation losses on Cresci 2017.

4.3 Cresci stock 2018

In this dataset [6] there are 6842 bots and 5882 users, for a total of 12724 samples labelled in human or bot. The function used to extract the digital DNA is Listing 3. In Figure 4.3 it is possible to see two images:

- the left one: it has some noise and it seems a more "complicated" image. In fact, it represents a human account;
- the right one: it is completly white except for the bottom which is black. It is a simpler and cleaner image with respect to the first one and it represents a bot.



(a) A human account in image format. **(b)** A bot account in image format.

Figure 4.3. Examples of the images generated for Cresci stock 2018.

In this case, the results in [3] will be taken as reference. *Antenore et al.* reported as best results: 0.77 in accuracy, 0.96 in recall and 0.82 in F1 score. With the novel approach, there has been a remarkable improvement in results.

Function used to create the DNA sequences.

```
def alphabet_b3_type(tweets_per_users) -> List[str]:
    """
        B3 type alphabet.
        {
            A <- tweet
            C <- reply
            T <- retweet.
        }
    """
    users_dna_sequences = []
    for user in tweets_per_users:
        user_dna_sequence = ''
        for tweet in tweets_per_users[user]:
            if tweet['retweeted']:
                user_dna_sequence += 'T'
            elif tweet['in_reply_to_user_id']:
                user_dna_sequence += 'C'
            else:
                user_dna_sequence += 'A'

        users_dna_sequences.append({'user_id': user, 'dna':
        ↪  user_dna_sequence})
    return users_dna_sequences
```

Listing 3

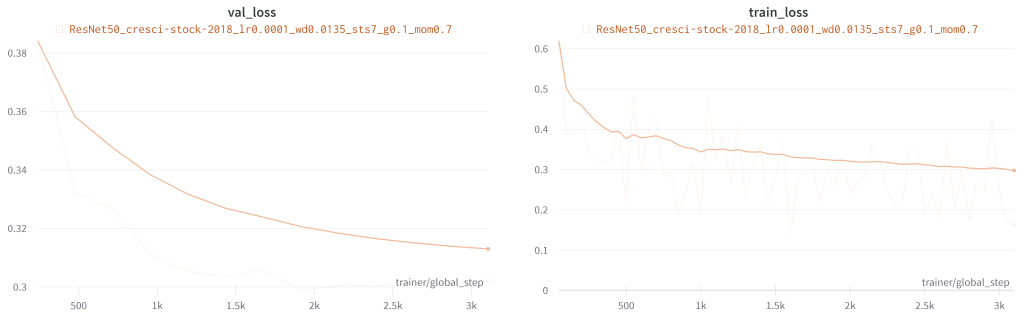
These results were obtained by the ResNet50 model and they are: **0.89** in accuracy, **0.89** in recall, **0.88** in F1 score and **0.78** in φ coefficient.

As it is possible to see in Figure 4.4, the two losses have a similar behavior and both decrease until the training is stopped.

4.4 Cresci rtbust 2019

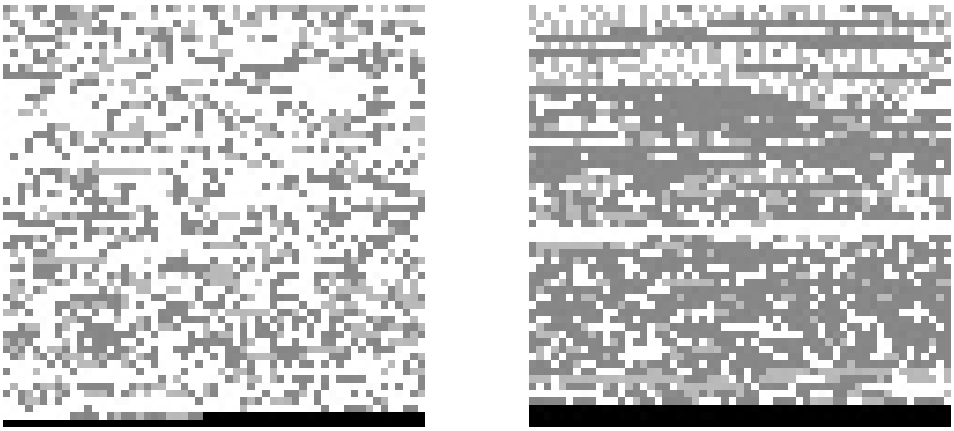
Another benchmarked dataset is the cresci-rtbust-2019 presented in [23]. This is a more recent dataset and thus it contains bots with behaviors more similar to those of real accounts. In fact, both the state of the art and the results obtained in this thesis are not like those of previous datasets. Cresci-rtbust-2019 has 315 bots and 317 users, for a total of 632 samples. Certainly the limited number of accounts collected affects the results.

In Figure 4.5 it is clearly visible that the image representing the bot is very similar to that of a real account; thus it will be more complicated for a neural network to find specific patterns related to only one of the two classes. In [3]



(a) Validation loss on Cresci stock 2018. **(b)** Training loss on Cresci stock 2018.

Figure 4.4. Train and validation losses on Cresci stock 2018.



(a) A human account in image format. **(b)** A bot account in image format.

Figure 4.5. Examples of the images generated for Cresci-rtbust-2019.

there are the results taken as reference: 0.58 in accuracy, 0.38 in recall and 0.47 in F1 score. The model that produced the best results is the WideResNet50, in particular it achieved: **0.81** in accuracy, **0.79** in recall, **0.81** in F1 score and **0.61** in φ coefficient.

4.5 TwiBot20

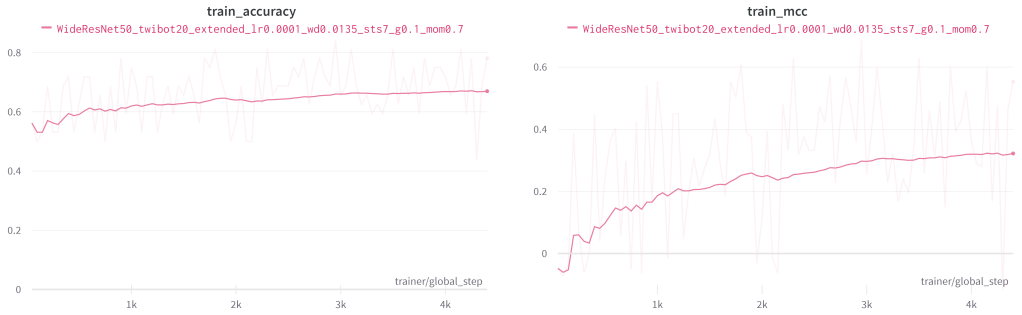
This is the most recent dataset benchmarked in this work. It is presented in [12] and it contains 6561 bots and 5185 users for a total of 11746 labelled samples. Nowadays, this is one of the most complete and up-to-date datasets available; the behaviors of the bots are very similar to those of valid accounts, and obtaining good results has not been easy. In fact, TwiBot20 has a maximum of 200 tweets per user and, therefore, the images created were really too small (15x15). Moreover, these images contain limited information that could

highlight specific patterns among the bots. An initial attempt was to enlarge the images, but the results obtained were not good.

(a) Human enlarged image. **(b)** Bot enlarged image.

Figure 4.6. Enlarged images in TwiBot20.

Figure 4.6 shows two examples of enlarged images. In Figure 4.7 there are the accuracy and φ coefficient obtained in the experiment with this type of images.



(a) Accuracy in first attempt on TwiBot20. **(b)** φ coefficient in first attempt on TwiBot20.

Figure 4.7. Training accuracy and φ coefficient in first attempt on TwiBot20.

As it is possible to see, the metrics are not the best. In fact, the accuracy does not improve much, compared to the initial phase, and the φ coefficient starts, even, from a negative value and then stabilizes just below 0.4. In detail, the results obtained with these images were: 0.34 for the φ coefficient, 0.66 for the F1 score, 0.66 for the recall, 0.67 for the precision and 0.67 for the accuracy.

At this point it was decided to consider not only digital DNA but also a set of features including, for example, the username length, the number of followers, and the number of tweets. In [30] the authors proposed the SuperTML algorithm to represent tabular data (i.e. the features set) into images. Since they had good results in their research and the approach seemed to be interesting and effective, it was decided to try this approach. In detail,

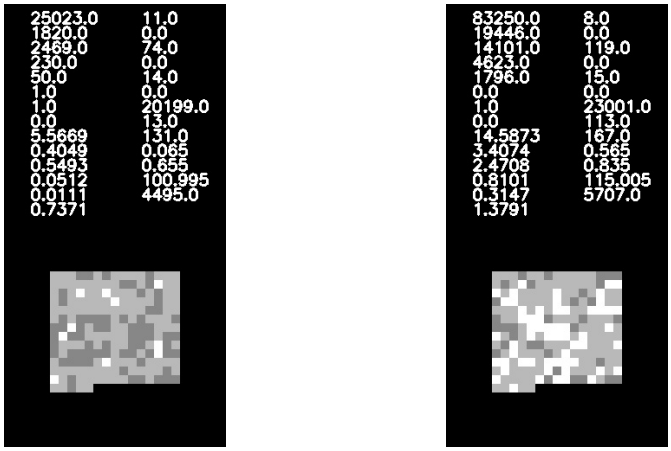
Feature	Description
statuses_count	number of tweets
followers_count	number of followers
friends_count	number of friends
listed_count	user's lists count
default_profile	if true the user has not changed the profile
favourites_count	number of favourites
profile_use_background_image	if user has a background image
verified	if is a verified account
followers_growth_rate	followers growth rate
friends_growth_rate	friends growth rate
favourites_growth_rate	favourites growth rate
listed_growth_rate	listed growth rate
followers_friends_rate	followers/friends rate
screen_name_length	username length
screen_name_digits_count	number of digits in username
description_length	description length
description_digits_count	number of digits in description
name_length	name length
name_digits_count	number of digits in name
total_tweets_chars_count	number of chars in tweets
total_urls_in_tweets	number of urls in tweets
total_mentions_in_tweets	number of mentions in tweets
urls_tweets_rate	urls/tweets rate
mentions_tweets_rate	mentions/tweets rate
chars_tweets_rate	chars/tweets rate
account_age	account age

Table 4.1. List of features used.

Sun et al. achieved state-of-the-art results on both large and small datasets. Some of the datasets that they used are the Iris dataset[1] and the Higgs Boson Machine Learning[2].

To proceed with the implementation of this new approach, a list of possible features to be used in images was compiled; these are summarized in Table 4.1. As it is possible to see, the values of these features are combined with the digital DNA image and the Figure 4.8 shows two images: the left one related to a real account while the right one to a bot account. In both cases, the DNA part of the image has been enlarged. The results taken as reference are in [12] and they are: 0.81 in accuracy, 0.85 in F1 score and 0.67 in φ coefficient. This

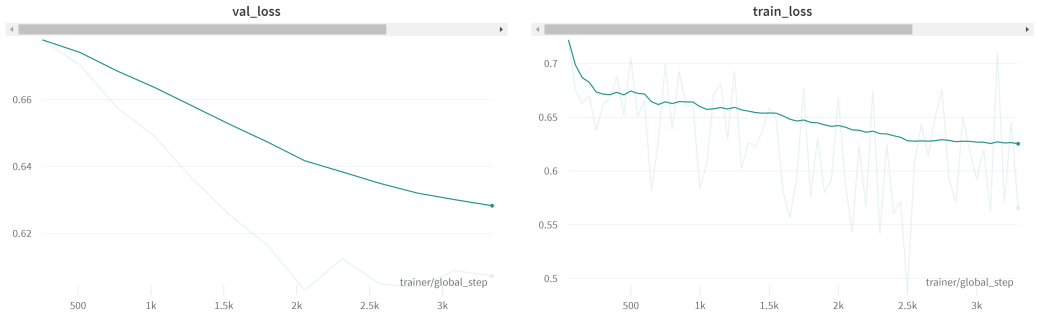
[1]Iris dataset homepage.
[2]Higgs Boson Machine Learning challenge on Kaggle.



(a) A human account in image format. **(b)** A bot account in image format.

Figure 4.8. Examples of the images generated for TwiBot20 with the SuperTML algorithm.

time, due to the limited amount of information regarding tweets in the dataset, the result of the approach proposed in this thesis are close to those of the state of the art: **0.82** in accuracy, **0.80** in recall, **0.80** in F1 score and **0.67** in φ coefficient.



(a) Validation loss on TwiBot20. **(b)** Training loss on TwiBot20.

Figure 4.9. Train and validation losses on TwiBot20.

In Figure 4.9 it is possible to see the two losses that decrease until they stabilize at about 0.6. The two losses have very similar behavior, which shows that the model is not overfitting.

In this particular case, a positive evolution of the results obtained could be seen; in fact the approach went from a 0.66 of accuracy to a value of 0.82, and this indicates how important it is to have well-structured datasets. One thing to point out is the fact that data on user relationships were not considered in these experiments. In the future, if these user relationships are considered in the approaches, better results can be achieved.

4.6 Overall results and discussion

The results obtained on the most common datasets are noticeable since they improve the state of the art and Table 4.2 summarizes them and compares them with what is the current state of the art.

	cresci-17	**cresci-stock**	**cresci-rtbust**	**TwiBot20**
Antenore et al.	- - - -	0.77 0.96 0.82 -	0.58 0.38 0.47 -	- - - -
Feng et al.	0.98 - 0.98 0.96	- - - -	- - - -	0.81 - 0.85 0.67
Novel approach	0.98 0.98 0.98 0.98	0.89 0.88 0.89 0.78	0.81 0.79 0.81 0.61	0.81 0.80 0.80 0.67

Table 4.2. Comparison of obtained results with [3] and [12]. - indicates that the metric is not available. The metrics reported are (from top to bottom): accuracy, recall, F1 score and φ coefficient.

The proposed approach certainly has proven to be efficient and effective obtaining promising results. Moreover, this is the first time that an image-based method has been proposed regarding bot classification. However, as can be seen some results could be further improved. In fact, since it is very hard to produce complete datasets (OSNs often block scraping and the APIs have a limited number of calls), it is necessary to take full advantage of the available data. For example, the results obtained by the proposed approach on the TwiBot dataset, which are still really promising, do not take into account the relationships between users. The results will definitely improve.

However, it can be seen from Table 4.2, for example, how the results regarding the Cresci-stock-2018 dataset are improved over those reported by *Antenore et al* [3]. In fact, this dataset is probably the most complete among those used, and this resulted in excellent image classification.

All the work was done on the Google Colab[3], which provides GPUs to do the training. However, the resources made available are often limited and it is common to run into possible timeouts. Certainly having hardware that is not

[3]Google Colab website.

too limited will help to improve the results and also to perform the training on many more images, as might be the case with the TwiBot22 dataset [13].

Chapter 5

Conclusions

The bot classification task is becoming increasingly important as seen in the case of Twitter's acquisition. Since this failed deal was worth $44 billion, efficient algorithms for bot detection can be worth a lot of money. Moreover, research in this field is still open given the continuous evolution of these accounts. The proposed approach could be further enhanced considering, for example, even more specific techniques such as PCA (*Principal Component Analysis*) regarding the features set to be used. However, in the future it will also be necessary to take advantage of relationships between users in a OSN because they are important data that should be exploited [1]. In fact, the results obtained in this study do not consider these relationships. Idealizing and creating models based not only on images but also on graphs could be another step forward. Research in this area will be also helpful in creating cleaner and safer networks, such as helping OSNs recommendation systems to avoid advertising posts generated by bot accounts and thus limiting for example the spreading of fake news. This study could also help to create new approaches not only with regard to Twitter, the social that is taken most into consideration in bot classification, but also others such as Reddit, Facebook and Instagram. Moreover, these approaches in the future could be useful in creating a tool, similar to botometer [37], that, in real time, is able to determine whether a given account is a bot or not with a certain probability.

However, it will be necessary to create datasets with more data such as TwiBot22 [13] so that approaches will be able to understand evolutions in bot behaviors and, consequently, correctly classify accounts. It will also be interesting to try to use future approaches based on the one proposed in this thesis on datasets for unsupervised learning in order to try to clustering groups of images. With unsupervised learning it will be possible to avoid all the

effort of manually annotating the datasets and at the same time obtain good results in detecting groups of bots with different behaviors on multiple OSNs. Moreover, it will be challenging to combine this image-based approach with a graph-based one so as to create an ensamble of valid approaches.

Ringraziamenti

I miei primi ringraziamenti devono andare a **Mamma** e **Papà** perché è soprattuto grazie a loro che sono arrivato dove sono ora. Ed è grazie a loro che ho potuto avere la possibilità di studiare senza alcuna preoccupazione. Un ringraziamento speciale va a mia sorella **Benedetta** e a mio fratello **Sebastiano**.

Un grande ringraziamento va al professore **Angelo Spognardi** che mi ha seguito, consigliato ed aiutato non solo durante questo periodo di tesi.

Infine, ringrazio tutti gli amici di Università, tra cui Andrea, Enrico e Riccardo; in particolare Andrea con cui ho potuto sviluppare molti progetti nel corso di questi ultimi due anni.

Bibliography

[1] Ali Alhosseini, S., Bin Tareaf, R., Najafi, P., and Meinel, C. Detect me if you can: Spam bot detection using inductive representation learning. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, p. 148–153. Association for Computing Machinery, New York, NY, USA (2019). ISBN 9781450366755. Available from: https://doi.org/10.1145/3308560.3316504, doi:10.1145/3308560.3316504.

[2] Andreopoulos, A. and Tsotsos, J. K. 50 years of object recognition: Directions forward. *Computer vision and image understanding*, **117** (2013), 827.

[3] Antenore, M., Rodriguez, J. M. C., and Panizzi, E. A comparative study of bot detection techniques with an application in twitter covid-19 discourse. *Social Science Computer Review*, **0** (2022), 08944393211073733. Available from: https://doi.org/10.1177/08944393211073733, arXiv:https://doi.org/10.1177/08944393211073733, doi:10.1177/08944393211073733.

[4] Biewald, L. Experiment tracking with weights and biases (2020). Software available from wandb.com. Available from: https://www.wandb.com/.

[5] Chavoshi, N., Hamooni, H., and Mueen, A. Debot: Twitter bot detection via warped correlation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 817–822 (2016). doi:10.1109/ICDM.2016.0096.

[6] Cresci, S., Lillo, F., Regoli, D., Tardelli, S., and Tesconi, M. $fake: Evidence of spam and bot activity in stock microblogs on twitter. In *ICWSM* (2018).

[7] Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., and Tesconi, M. The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. *CoRR*, **abs/1701.03017** (2017). Available from: http://arxiv.org/abs/1701.03017, arXiv:1701.03017.

[8] Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., and Tesconi, M. Social fingerprinting: detection of spambot groups through dna-inspired behavioral modeling. *CoRR*, **abs/1703.04482** (2017). Available from: `http://arxiv.org/abs/1703.04482`, `arXiv:1703.04482`.

[9] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee (2009).

[10] Efthimion, P. G., Payne, S., and Proferes, N. Supervised machine learning bot detection techniques to identify social twitter bots. *SMU Data Science Review*, **1** (2018), 5.

[11] Falcon et al., W. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, **3** (2019).

[12] Feng, S., Wan, H., Wang, N., Li, J., and Luo, M. Twibot-20: A comprehensive twitter bot detection benchmark. *CoRR*, **abs/2106.13088** (2021). Available from: `https://arxiv.org/abs/2106.13088`, `arXiv:2106.13088`.

[13] Feng, S., et al. Twibot-22: Towards graph-based twitter bot detection (2022). Available from: `https://arxiv.org/abs/2206.04564`, `doi:10.48550/ARXIV.2206.04564`.

[14] Fukuda, M., Nakajima, K., and Shudo, K. Estimating the bot population on twitter via random walk based sampling. *IEEE Access*, **10** (2022), 17201. `doi:10.1109/ACCESS.2022.3149887`.

[15] Gilani, Z., Farahbakhsh, R., Tyson, G., Wang, L., and Crowcroft, J. Of bots and humans (on twitter). In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pp. 349–354 (2017).

[16] Gilmary, R., Venketesan, A., Praveen, M., Prasath, H. R., and Vaiyapuri, G. Detection of twitter bots using dna-based entropy technique. In *2022 First International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, pp. 1–6 (2022). `doi:10.1109/ICEEICT53079.2022.9768516`.

[17] Hayawi, K., Mathew, S. S., Venugopal, N., Masud, M. M., and Ho, P. Deeprobot: a hybrid deep neural network model for social bot detection based on user profile data. *Soc. Netw. Anal. Min.*, **12** (2022),

43. Available from: https://doi.org/10.1007/s13278-022-00869-w, doi:10.1007/s13278-022-00869-w.

[18] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, **abs/1512.03385** (2015). Available from: http://arxiv.org/abs/1512.03385, arXiv:1512.03385.

[19] Jeffrey, H. Chaos game representation of gene structure. *Nucleic Acids Research*, **18** (1990), 2163. Available from: https://doi.org/10.1093/nar/18.8.2163, arXiv:https://academic.oup.com/nar/article-pdf/18/8/2163/7059915/18-8-2163.pdf, doi:10.1093/nar/18.8.2163.

[20] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (edited by F. Pereira, C. Burges, L. Bottou, and K. Weinberger), vol. 25. Curran Associates, Inc. (2012). Available from: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[21] Kudugunta, S. and Ferrara, E. Deep neural networks for bot detection. *Information Sciences*, **467** (2018), 312. Available from: https://doi.org/10.1016%2Fj.ins.2018.08.019, doi:10.1016/j.ins.2018.08.019.

[22] Lee, K., Eoff, B., and Caverlee, J. Seven months with the devils: A long-term study of content polluters on twitter. In *Proceedings of the international AAAI conference on web and social media*, vol. 5, pp. 185–192 (2011).

[23] Mazza, M., Cresci, S., Avvenuti, M., Quattrociocchi, W., and Tesconi, M. Rtbust: Exploiting temporal patterns for botnet detection on twitter. *CoRR*, **abs/1902.04506** (2019). Available from: http://arxiv.org/abs/1902.04506, arXiv:1902.04506.

[24] O'Shea, K. and Nash, R. An introduction to convolutional neural networks (2015). Available from: https://arxiv.org/abs/1511.08458, doi:10.48550/ARXIV.1511.08458.

[25] Ratner, A., Bach, S. H., Ehrenberg, H. R., Fries, J. A., Wu, S., and Ré, C. Snorkel: Rapid training data creation with weak supervision. *CoRR*, **abs/1711.10160** (2017). Available from: http://arxiv.org/abs/1711.10160, arXiv:1711.10160.

[26] Shorten, C. and Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *Journal of big data*, **6** (2019), 1.

[27] Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (edited by Y. Bengio and Y. LeCun) (2015). Available from: http://arxiv.org/abs/1409.1556.

[28] Somodevilla, M. R. L., Rossainz, M., et al. Dna sequence recognition using image representation. *Research in Computing Science*, **148** (2019), 105.

[29] Sun, B., Yang, L., Dong, P., Zhang, W., Dong, J., and Young, C. Super characters: A conversion from sentiment classification to image classification. *CoRR*, **abs/1810.07653** (2018). Available from: http://arxiv.org/abs/1810.07653, arXiv:1810.07653.

[30] Sun, B., Yang, L., Zhang, W., Lin, M., Dong, P., Young, C., and Dong, J. Supertml: Two-dimensional word embedding and transfer learning using imagenet pretrained CNN models for the classifications on tabular data. *CoRR*, **abs/1903.06246** (2019). Available from: http://arxiv.org/abs/1903.06246, arXiv:1903.06246.

[31] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. *CoRR*, **abs/1512.00567** (2015). Available from: http://arxiv.org/abs/1512.00567, arXiv:1512.00567.

[32] Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, **abs/1905.11946** (2019). Available from: http://arxiv.org/abs/1905.11946, arXiv:1905.11946.

[33] Wei, F. and Nguyen, U. T. Twitter bot detection using bidirectional long short-term memory neural networks and word embeddings. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 101–109 (2019). doi:10.1109/TPS-ISA48467.2019.00021.

[34] Weiss, K., Khoshgoftaar, T. M., and Wang, D. A survey of transfer learning. *Journal of Big data*, **3** (2016), 1.

[35] Yang, C., Harkreader, R., and Gu, G. Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Transactions on Information Forensics and Security*, **8** (2013), 1280. doi:10.1109/TIFS.2013.2267732.

[36] Yang, C., Harkreader, R., and Gu, G. Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Transactions on Information Forensics and Security*, **8** (2013), 1280. doi:10.1109/TIFS.2013.2267732.

[37] Yang, K., Ferrara, E., and Menczer, F. Botometer 101: Social bot practicum for computational social scientists. *CoRR*, **abs/2201.01608** (2022). Available from: https://arxiv.org/abs/2201.01608, arXiv:2201.01608.

[38] Yang, Y., Yang, R., Li, Y., Cui, K., Yang, Z., Wang, Y., Xu, J., and Xie, H. Rosgas: Adaptive social bot detection with reinforced self-supervised gnn architecture search (2022). Available from: https://arxiv.org/abs/2206.06757, doi:10.48550/ARXIV.2206.06757.

[39] Yardi, S., Romero, D., Schoenebeck, G., et al. Detecting spam in a twitter network. *First monday*, (2010).

[40] Ying, X. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, vol. 1168, p. 022022. IOP Publishing (2019).