**Master Thesis Report (2020)**

# Read between the Tweets: A context-based approach to Social Media Manipulation Detection

Leonardo Aoun

March 12, 2020

| | |
|---|---|
| **Report Type:** | Final |
| **Master Student:** | Leonardo Aoun |
| **EPFL Supervisor:** | Dr. Mathieu Salzmann |
| **Industry Supervisor:** | Daniyal Shahrokhian |
| **Date:** | March 12, 2020 |

# Contents

# 1 Introduction

The 2016 US presidential election brought with it a wave of controversy regarding social media. News outlets were clattered with rumors about Russian interference after unveiling nets of fake bots trolling users on Twitter and Facebook, sharing hoax stories about candidates, with the top 20 most shared of them generating more than 8 million interactions on Facebook[1]. Ever since, fake news has become a concern for citizens across the western world. In Europe, a survey conducted on more than 25,000 people found that 37% of them encounter fake news almost once every day, and in total, 80% do more than several times a month[2]. This growing fear has not yet been attended by big social media corporations, claiming it's not the responsibility of the platform to police free speech[3]. This is why NGOs like Snopes and Politifact are championing this task, or newer yet smaller companies like Fabula AI[4], OptOutTools[5], or Astroscreen[6] are stepping up to combat social media manipulation. Fake news is not the sole component that falls under the umbrella of the term social media manipulation, but bullying and harassment too. A study in 2017[7] found that 41% of internet users in the United States were victims of online harassment at least once that year, 30% of teenagers have been cyberbullied more than once[8]. With 95% of teens being connected to the internet in this day and age with only 10% of them reporting these incidents, this trend is turning out to be a recipe for a social disaster in years to come.

These companies are employing hand crafted features to detect tampering or other forms of manipulation like disinformation but this fight is turning into a case of Catch 22, attackers are getting better and learn how to sidestep the decision rules set by the defence systems. Since the scope of the problem became so big, and attacks have become more frequent, so has the amount of data on it. That's why machine learning studies and especially deep learning have become more possible in that field. [8] reviews most of the advances in terms of analysing textual content and images and classify them as either fake or real, sexist, racist, etc.. it also discusses work that has been done in terms of using the context of social media posts, in addition to their early propagation, and not just settling for the content of a tweet or profile. This is what Fabula AI improved on, they built graphs that represent how a news article has been shared on Twitter and applied "Geometric Deep Learning" to reduce these graphs into a feature vector and classify the article as either fake or real [1]. This term has been popularized by the survey from Michael Bronstein and Yann LeCun [12] and means techniques of deep learning that don't handle the input data as if it's living in an Euclidean space, but rather as graphs and manifolds. This opens up a lot of possibilities when the data cannot be morphed into a grid shape, but should rather represent actions between entities, like atoms in a molecule or documents, papers, and patents in a legal bureaucratic setting.

I am personally a big advocate of social media, have and will work at Facebook Inc., and hold political views that are quite liberal so I felt a great motivational push towards helping the cause of making such platforms a cleaner environment. I found out about Astroscreen and applied to collaborate with them on a project. At the time of being hired, they were building products that protected brands from coordinated cyber-attacks, detecting fake accounts or bots and tracing them down to the entities behind them. My role was to replicate Fabula AIs innovative classifier and integrate it into the machine learning pipeline of Astroscreen.

[1] https://www.washingtonpost.com/opinions/without-the-russians-trump-wouldnt-have-won/2018/07/24/f4c87894-8f6b-11e8-bcd5-9d911c784c38_story.html

[2] https://www.statista.com/statistics/1076568/fake-news-frequency-europe/

[3] https://www.nytimes.com/2020/01/09/technology/facebook-political-ads-lies.html

[4] https://www.fabula.ai

[5] https://optouttools.com

[6] https://astroscreen.com

[7] https://www.pewresearch.org/internet/2017/07/11/online-harassment-2017/

[8] https://www.dosomething.org/us/facts/11-facts-about-cyber-bullying

This report traces the progression of my project as follows: Section 2 reviews the papers related to this subject and the tools I would use, section 3 explains the approach I would take to build my classifier, section 4 describes how I gathered a suitable dataset to work on, and finally section 5 presents the results I got and discusses them.

# 2 Previous Work

## 2.1 Reviews on Social Media Manipulation Detection

### Misinformation in Social Media

In order for me to achieve this project and help fight social media manipulation through machine learning, I need to review what has already been done towards this task. In 2019, a join effort from students at Arizona State University, USC, and CMU published a survey on Misinformation in Social Media [8]. It's the most recent on the topic and its comprehensiveness would give me a great reconnaissance of the field. Its main three contributions are:

- Defining Misinformation and various categories it could fall under.
- Understanding how spreaders propagate the information
- Describing individual methods used to detect these attacks

Firstly, misinformation is the process of spreading fake or inaccurate information, but it could take many forms, figure 1 lists the most common ones. Disinformation is the special case where this inaccuracy is deliberate. Most of the tweets and news circulating back in 2014 about Ebola were subjects of unintentional misinformation. Web citizens were sharing the widespread articles without the intention to harm communities hit by the virus but were definitely indirectly harming the fight against it. Paul Horner, a notorious fake news articles author, admitted in 2017 that he helped the Trump campaign in 2016 distort the image the current president had while running for president. Disinformation itself comes in many forms. It could be an urban legend, which has entertainment as its main goal, or straight up fake news articles for many reasons. It could also be unverified information or a catchy rumor, that is sometimes true. Some other sophisticated tactics are Astroturfing (inspiring the name Astroscreen to the company) which consists of masking the supporters and sponsors of a campaign to make it appear to be launched by grassroot participants (a bit like doping at the Olympic Games...), crowdturfing is similar but these fake participants are hired on the Internet via platforms like Zhubajie, Sandaha, and Fiverr. Spamming and Trolling is the act of spreading unsolicited information to respectively overwhelm the recipient or cause disruption and hatred towards a certain group of people. Hate speech and cyberbullying are other ways, potentially coupled with disinformation, that can spread hatred in the online community.

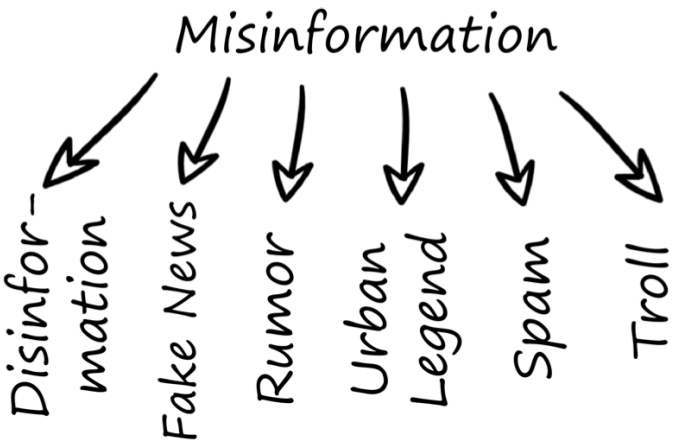



Figure 1: Different categories of misinformation,[8]

Secondly, there are two big classes of manipulation types, those that are content-based, and those that are network-based. Content-based manipulators focus their attacks on the building blocks of the social media platform, posts on Facebook, tweets on Twitter or videos on Youtube. They put their efforts into morphing the most plausible content that can reach a widespread adoption before raising any eyebrows. They can also work on the profiles to make their contents more believable. This is actually something that doesn't scale well for attackers and a good way so far of fighting these techniques is to classify their profiles like [22] by analysing their bio, longevity of their profiles, or their posting behavior and coherence of their content like [23]. The urls they share can help generate a signature to label similar accounts and detect bursts of fake articles. Attackers are themselves learning too, developing a technique [8] calls as "camouflage" and showcase in figure 2, in which the bots mimic the behavior of legitimate users for most of their posts, and then slip in a few misinformation posts, it makes the problem more of a post classification task while also making it harder for scientists to get labels for training data. However, with enough of it, Naive Bayesian methods, SVMs or Decision Trees couple with AdaBoost have proven to do the trick. Modeling the contents of different users and bots has also found posting fingerprints that can be used to detect anomalies in posting behaviors or contents. On the other hand, network-based manipulators leverage the mechanics of the social media platform to fool legitimate users. Pools of bots are created that all follow each other to make it look like a natural cluster of people, and individuals from it follow legitimate users who follow back just as most people are indifferent to tailgaters. These bots then unfollow each other to limit their chances of being detected and would have thus infiltrated legitimate communities. This sophisticated technique annuls most feature engineering approaches like number of followers, ratio of followers to followees, Homophily (which was based on the assumption that pair of friends are more likely to have the same label.). Unsupervised approaches have been developed like [24] using a HMRF probabilistic model and GMMs to group profiles together and label the outliers as misinformation spreaders. However, it's easy to get false negatives in this approach due to the uniqueness and irregularity of social media behavioral patterns which makes it hard to predict where to draw the lines between clusters. [25] proposes a solution that is adaptive in terms of count of sizes of the clusters, beating previous approaches that relied on the RFF (Ratio of Followers/Followees) or neighbourhood structures.

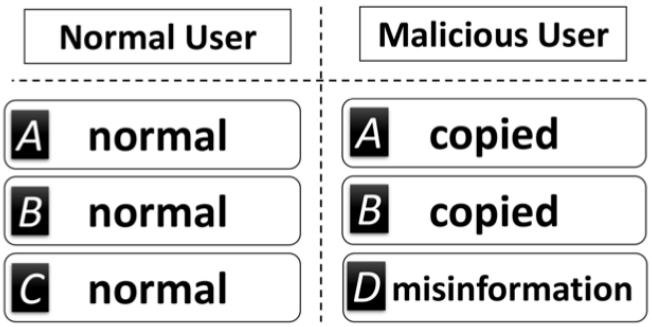



Figure 2: Example of how a malicious user would copy most contents from a legitimate user to "camouflage" misinformation contents, [13]

Thirdly, they introduce most state of the art methods seen so far to detect these misinformation exploits. They first define the problem as similar to traditional text classification ones but warn of the difference with them. Text classification projects usually aim at differentiating between various document types, article topics which are mostly organic and compiled to be distinguishable. In misinformation detection, however, content is deliberately made to look real and organic, and the decisive points can be contextual, outside of the article itself. Content based approaches have the highest rate of false positive, especially if focusing on single posts, clustering the messages by content, creation time, authors, etc. has proven to give better results on popular topics. Context based approaches like [26] and [27] propose to detect bursty patterns in spreading to detect respectively fake news and rumors advocating that misinformation spreads temporally in bursts on social networks. A framework called

TraceMiner [28] classifies the propagation of a message based on the network embedding of the users sharing it. As any machine learning problem, and even more increasing because of the adversarial nature of this field, the detection of misinformation needs a lot of data, which has to stay up to date with the new types of attacks used by manipulators. Works like [29] have come up with innovative ways to circumvent this, aggregating posts very early on, to train models to detect manipulations with as few insight as possible. By being generic in their aggregations, they allow the model to focus on common features while also having more sources of training data. The problem however is in knowing how and when to group them, and their main breakthrough was to look at the reactions to the content, instead of it alone. While for labelling, a good approach for many topics like sports or business articles is to use data gathered up to a certain month to classify the ones from the upcoming one.

Overall, they found that across the methods they reviewed, the most commonly successful tweaks to feature engineering were to consider profiles and posts as documents, coupled with Bag of Words or TF-IDF to give weights to some words, using word embeddings because posts are usually small, which caused previous methods to give sparse representations unsuited for learning, focus on keywords from urls, hashtags, sentiment, emojis. This last point gives the impression that for now, it's still very beneficial to get the hands dirty in the process of manipulation detection and keep hardcoding some decision rules. Additional ways of gathering data have been reviewed: The first is to analyze lists of users suspended by a social network, either by waiting for public ones to be released, or monitor a lot of users periodically and be alerted when they get banned. While this approach provides large quantities of usernames, their qualities are highly influenced by the rules of communities from which they were banned which makes it less likely to be transferable from one platform to another, let alone group or page type to another. [22] and [9] deployed "Social Honeypots" on MySpace and Twitter respectively to attract spammers and content polluters. Legitimate users can rarely be caught by honeypots but are sampled from the main population of the social network. They monitored their preys for months before classifying them, but achieved high accuracy results of legitimate vs fake accounts classification. Unfortunately it's not an approach I could follow in my thesis as it requires a long time to get initial results but could be a worthy approach to try in a more stable setting. [30] went for a more traditional approach of gathering Twitter data, by simply searching for keywords and hashtags, then asking a woman who studied gender studies to annotate their 16K-tweets-large dataset, had I had the resources to do something similar, I could have went for an approach like this but it was definitely too time consuming for a part of a semester project.

**Attributing Fake Images to GANs**

Specifically for image data, considerable enhancements have been achieved in generative approaches of fake imagery. Ever since Generative Adversarial Networks [31] came up, fake images and videos resulting from GANs have become more popular, they're being called Deep Fakes and have gathered big concerns from the general public [9]. But just as the tools for attackers are becoming more potent, researchers are advancing the ways to detect them. This year at CVPR, [11] have showcased their approach that beat benchmarks in deep fakes detection. Until then, the research community has been using GAN forensics to detect manipulations in images, like copy-move manipulations or face swapping but these approaches are specific to certain GAN architectures or original input spaces and won't scale with more and more source GANs. One technique was banking on the fact that most Human Face datasets used in GAN trainings don't have image of people with their eyes closed, which made the forensic of analyzing eyelids very successful. Newer adversaries just had to add some periodic or random eye closing for their videos to circumvent this flaw. [11] uses deep learning to keep their approach generic enough to combat any GAN architecture attackers might use. They train their discriminator against most generative models seen so far, and have arrived to the conclusion that

[9] https://www.youtube.com/watch?v=AmUC4m6w1wo and https://www.youtube.com/watch?v=gLoI9hAX9dw

these models leave stable fingerprints on their output images and even minor differences in GANs can lead to different fingerprints. These fingerprints persist over different patches and frequencies of the image. In addition to beating benchmarks like the Inception Features benchmark, they also achieve great results when images are perturbed (blurred, noised, etc...). Their work, and following advancements in the community will be very valuable in not only preventing misinformation from spreading on social media platforms, but also protecting the intellectual property of authors there, especially since content creators are becoming more valuable for all social media platforms.

[11] bring forward the notions of Model Fingerprints and that of Image Fingerprints. The first vary from model to model because of the architecture or the initializing weights, it is a reference vector that interacts with the generated images. The second is a feature vector encoded from each image that will help attribute it to a certain GAN instance. To do this, they train a fully convolutional network and three variants,

1. a pre-downsampling network that applies Gaussian downsampling first and then a fully convolutional network.
2. a pre-downsampling residual network that performs downsampling a few times, links the outputs via some skip connections and then finish with a fully convolutional network.
3. a post-downsampling network that starts with convolutions to get features maps with labels per pixel for its receptive field then applies successive average pools to get overall labels.

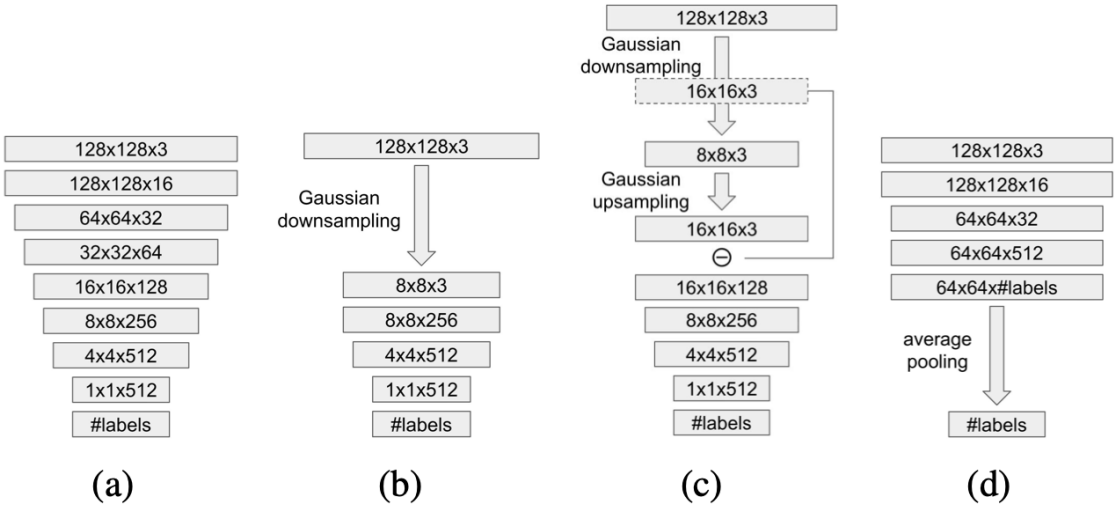



Figure 3: Tensor sizes of the networks trained in [11]: (a) The main Fully Convolutional Network (b) Pre-Downsampling Network (c) Pre-Downsampling Residual Network (d) Post-Pooling Network, [11]

These variants, whose model graphs are listed in figure 3, will aim to train the processes that generates these fingerprints: the image fingerprint is given to each generated image **I** by passing it through an autoencoder and computing the residue when subtracting the reconstructed image from **I**, while the model fingerprint will be trained per GAN type. Then, as shown in figure 4, the image fingerprint is then pixelwise-multiplied with each model fingerprint to decide the most likely source of **I**. The decision to go for this fingerprinting generation through an autoencoder comes from the limitation that there is a lack of ground truth images. The objective of the training pipeline is to maximize the correlation between the two fingerprints and the label. The label being whether or not an image is original, and if not, which GAN created it.

I spent some time at a point in the thesis considering whether it would be useful to pursue deep fakes detection both to my thesis and the company. The paper advertised two datasets they were

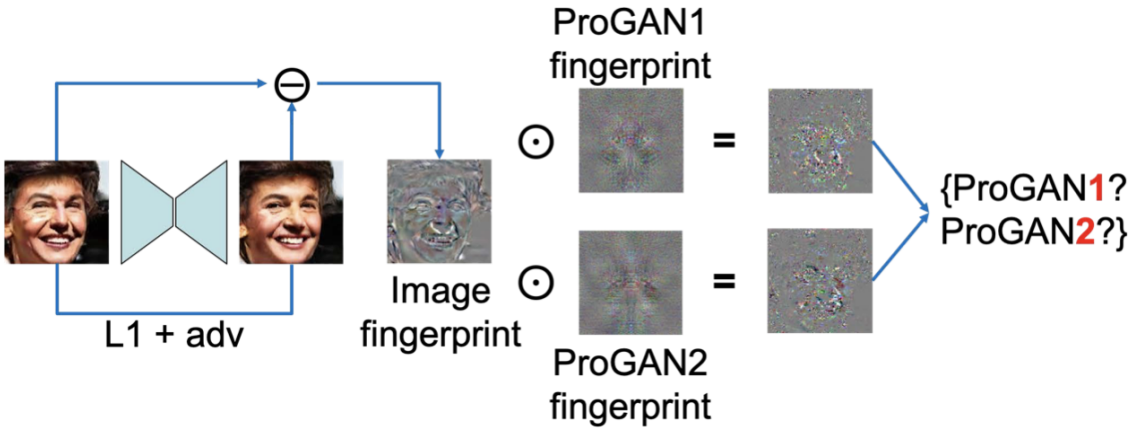



Figure 4: Diagram explaining how the image and model fingerprints are used to decide which GAN is the most likely source of the image, [11]

using: CelebA and the LSUN bedroom scene, I also alternatively found Fakeddit [32] which contains 800K samples of text+image fake news examples. They practiced their model against 4 popular GAN architectures: ProGAN, SNGAN, CramerGAN, MMDGAN and evaluated their results against 3 common benchmarks used for deep fake detections: kNN on raw pixels, Eigenface, PRNU-based fingerprint from [33]. Both their classification accuracy was higher and their ratio of inter-class and intra-class Frechet Distance was better, which means the feature representation across classes of GANs were more distinguishable. Their results showed high attribution performances even when overlapping the training dataset with different proportions of random real images. They also got insight on how image fingerprinting is affected by image frequencies and downsampling: Wider frequency bands carry more fingerprint information, and even though both low and high frequency components carry effective information, higher ones have more fingerprints which personally seems logical, as "mistakes" are easier to find in generated images around edges. Finally, they tried constraining the dataset to consist of the 10% most similar images, and found that the attribution task was more challenging.

Applying this work on the company's product was going to be challenging but I was up for the task. I could not retrain GANs to generate weird training data like and image of Trump getting intoxicated, or two controversial politicians shaking hands but I could have trained the pipeline on just their faces for now while waiting for a more creative dataset. It would have also been possible to focus on Fakeddit and reproduce their work on taking as input pairs of images and text instead of just images. A step closer to my initial thesis could be to handle pair inputs of graph and image to fight the content of deep fakes and their propagation on the web.

## 2.2 Geometric Deep Learning

Michael Bronstein named the term Geometric Deep Learning in [12] to reference any machine learning approach that doesn't consider euclidean input data. Figure 5 shows that while Euclidean convolutional networks would try to deform such data, or project its filters onto spaces that can handle them, Geometric approaches are constructed in a way that can handle a non-eucledian space and its filters are invariant to the deformations that could affect the input. As my thesis is operating on social network data, and how information spreads through it, which would be modeled as graphs, it definitely falls under this category.

The Microsoft Research team released an introductory video [10] on Graph Neural Networks, which are geometric deep learning techniques that take graphs as input. The presenter gives the example

[10] https://www.youtube.com/watch?v=cWIeTMklzNg

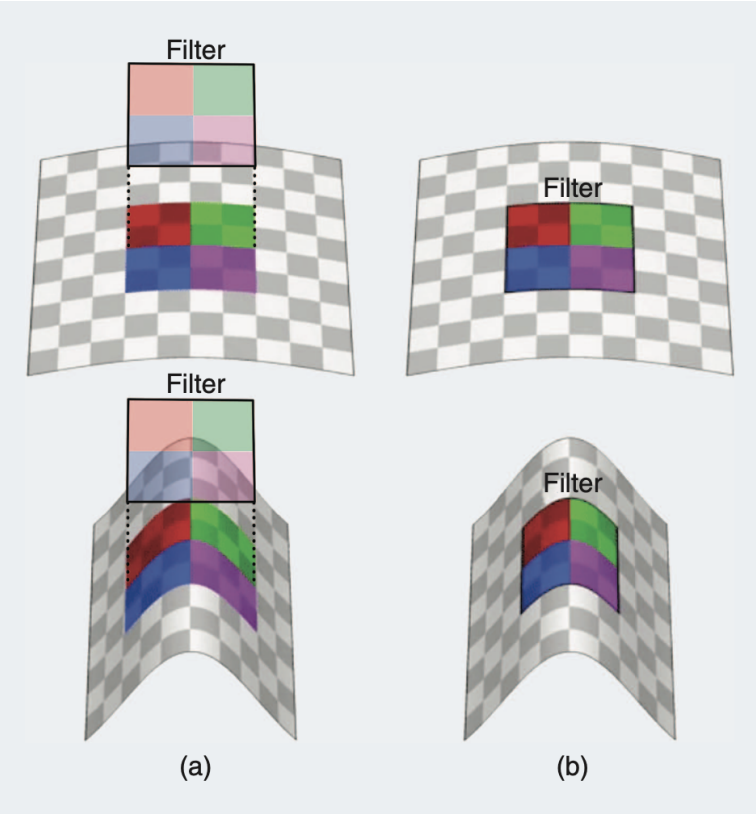



Figure 5: Image showing how a Geometric Deep Learning approach (b) can have a different view on an input space than the classical Euclidian approach (a), [12]

of graphs representing molecules, where a node is an atom that have a feature vector representing its periodic number and edges are the connections between them which could have two bond levels. The most vanilla version of a network he introduced was putting a neural network at each node, pulling messages (feature vectors) from its neighbours at each timestep, computing their sum or some function that should be rotation invariant to the messages collected. After some iterations, a node should have enough information about the whole molecule.

Xavier Bresson gave a talk [11] that stressed upon the importance of compositionality for convolutions, and how to achieve them when the data is in the form of graphs. The reasons why images, videos, sounds are compositional is because they combine the concepts of locality (fragments in data points of these types can be spotted by their location), of stationarity (the properties of a data point are the same no matter which part of it we look at) and of multi-scalability (the properties stay the same no matter how large the instance gets). If we examine a graph closely, we see that these 3 conditions can be applied to it so there should be a way to apply convolutions in that domain. Xavier goes on to explain [14]'s spectral approach which I'll introduce right after. After that, I'll talk about the second more applied spatial approach to graph convolutions and give examples of variants reviewed in the surveys [12] and [13].

**Spectral Networks**

In computer vision, a convolutional neural network takes in an image, which is represented by a grid of values. It takes advantage of its locality, stationarity, and multi-scalability to translate over it a convolutional matrix and pairwise multiply it with the pixels of the image. It does this operation as many times as the network architecture specifies it while also alternating with operations of downsampling

[11] https://youtu.be/v3jZRkvIOIM

the processed image by taking either the maximum pixels of their averages.
[14]'s take on graph convolutions was to use graph Laplacians to apply convolutional functions on the graph, which explains the name Spectral that comes from spectrum. The Laplacian **L** is a semi-definite n*n matrix, where n is the number of nodes in the graph, and its computation can come through multiple forms:

- Unnormalized where it's the difference between the Degree matrix and the adjacency matrix $D - A$
- Normalized where it's $I - D^{-.5}AD^{-.5}$ where I is the identity matrix
- Equivalent to a Random Walk $I - D^{-1}A$

where **D** the degree matrix means the diagonal matrix that counts the number of neighbours per node. To apply a convolutional function f on the graph, we take its Laplacian and extract the eigenvectors, decompose the function using Fourier Transform into as many functions as we have extracted eigenvectors, apply them all then multiply the results. When it comes to downsampling, there are many graph partitioning algorithms that can be used, like Balanced Cut with Heavy Edge Matching.
The convolutional operations are quadriatic and training them on one graph topology will not make them transferrable to others, as the Laplacian is dependant on the topology of the graph. However, if the graph is sparse, which would be the case in many applications, instead of getting the eigenvectors, we could multiply the Laplacian hop-times to a function, where "hop" is how far each node should know about which makes the computation linear in terms of the product of hop-times and the number of edges in the graph.

### ChebyNets

Xavier Bresson warns in his aforementioned talk that if the Fourier basis (represented by the eigenvectors of the Laplacian operator) is not orthogonal, the change of parameters during training will affect others, which is why an orthogonal basis had to be chosen, and the proposed solution in [17] is to use Chebyshev polynomials. Following a Chebyshev expansion algorithm, we can parametrize the Laplacian into a basis that makes it more computationally efficient to apply convolutions on the graph. Their work provide this way of applying fast localized convolutional filters on graphs with the same complexity as standard CNNs in a manner that learns local, stationary and compositional features from them. They enhance the work done by [14] to allow a linear complexity of convolutions in terms of the number of hops desired from the central vertices. They also come up with a more efficient graph pooling technique by rearranging the vertices as a binary tree which makes it analog to pooling in one dimension as shown in figure 6. Overall, they increase the accuracy and decrease the complexity of the models proposed in [14].

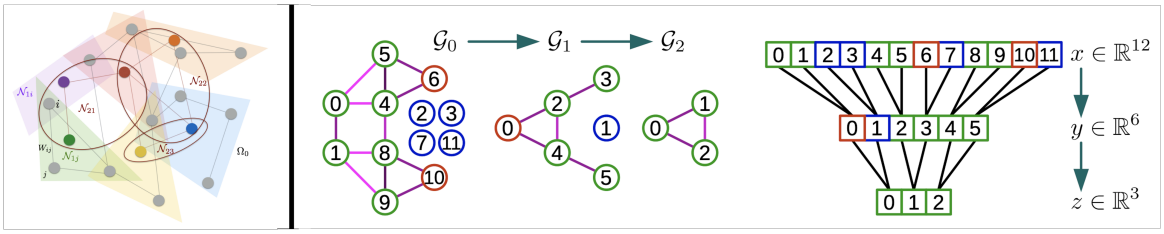



Figure 6: Pooling strategies in Graph Convolutional Networks. (Left) Based on neighbourhood clustering, [14]. Compared to (Right) which rearranges the nodes as a binary tree and simply pools using the indices, [17]

### Graph Convolutional Nets

Thomas N. Kipf and Max Welling argue against Chebyshev polynomials in [15] wanting to free their layers from the parametrization limitations of these polynomials. In [17], a layer's computation was defined as $\sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x$ where $T_k$ is the Chebyshev polynomial expression of degree $k$, and $K$ is the maximum number of node hops to consider from each central node on which the filter is applied. Hence, they are limiting each filter to $K$ maximums steps away from the center. Stacking multiple convolutional layers of this form alternatively with activation functions would build a graph neural network where each layer has a computational cost linear in the number of edges that a central node has in its $K^{th}$ order neighbourhood. [15] proposes to limit $K$ to 1, making the convolutional layer's computational cost linear w.r.t. the Laplacian spectrum. Such model would alleviate overfitting issues that come with explicit parametrizations like the Chebyshev polynomials and since the function is only linear in $\mathbf{L}$, it has a fixed computational cost and would allow them to stack a lot more of these filters and build a deeper model, also lowering the number of learnable parameters per layer ergo making it easier to load on memories.

### Cayley Nets

Defferrard et al. went back to their idea in [17] and improved on the polynomial parametrizations by using the Cayley polynomials in CayleyNets [16]. These networks stay localized and have the same linear complexity per number of edges but have the ability to detect narrow frequency bands of importance and specialize on them during training. These polynomials have mathematical properties that are beneficial to spectral theory, the main ones being that any smooth spectral function can be approximated by a Cayley polynomial and doesn't need a matrix inversion of the Laplacian to be computed. Their results are on par with the ones in [15], even beating them in some benchmarks.

### ConvNets for arbitrary graph

All of these CNNs introduced in the previous section are spectral-based ones, using spectral graph theory to formalize the convolutional filters. [13] reintroduces spatial-based Graph Convolutional Networks by providing examples of papers that reignited the interest in Message-Passing graph convolution techniques. They share a list of 20 spatial-based CNN, it would be too long to go over them, but one that caught my attention $_{pun\ intended}$ thanks to a talk at ICLR 2018 published on YouTube [12] is the Graph Attention Network [34]. A more recent approach from the same laboratory is the Deep Graph Infomax [36], it only focuses on unsupervised learning through graph networks, but some of its contributions could inspire my thesis. [35] is one of the most recent papers aggregating knowledge on message passing graph convolutions and contributing a message passing algorithm that is at the time of writing, the most guaranteed to have the best theoretical results.

Inspired by the work of [37] who developed a self-attention mechanism in language processing models, [34] introduce an attention-based architecture to perform node classification of graph-structured data by computing the hidden representations of each node by attending its neighbours following a self-attention strategy similar to [37]. The strategy is efficient as it can be parallelizable, is able to handle nodes with various number of neighbours, and can be generalized to different topologies of graphs, performing good results even on unseen graphs. At each layer of the neural network, a shared linear and learnable transformation is applied to each node's feature $h_i$ to get its new feature $h'_i$, after that, an attention function is applied to each pair of nodes that are neighbours to compute the attention coefficient between them. This attention computation function is a simple single layer

[12] https://youtu.be/6hbWpbi0Z24

feed forward network whose input space's dimension is that of the features and is unrelated to the number of nodes in the graph, figure 7 might clarify how it is computed. Once all these attention coefficients are computed, they normalize them across the neighbours of each node and move on with the message passing and aggregation to get the new feature representations of the node. However, as illustrated in figure 8, they have found that the learning is more stable if they concatenate in parallel different attention functions to not restrict the training to one initialization of attention weights.

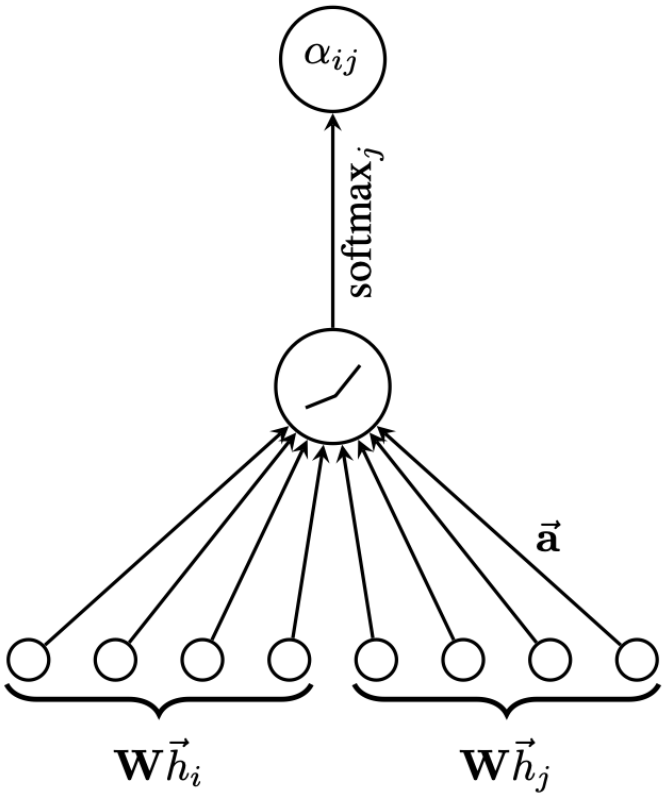



Figure 7: The feed forward network computing the attention coefficient between two neighbouring nodes, [37]

At the time of writing [36], the dominant algorithm for unsupervised representation learning with graph-structured data was based on random walks. However, the authors argue that they suffer from known limitations, like focusing too much on proximity information and not enough on structural information. This is why [36] apply [38]'s work on graph-structured input. The idea is to leverage convolutional neural networks to maximize the mutual information between a high-level "global" representation and "local" parts of the input, whether it's images in [38] or graphs in this new application. They thus train an encoder that takes the node features and adjacency matrix as inputs and return for each node a representation of the graph centered at this node. In order to maximize the local mutual information they leverage a readout function that takes these patch representations and creates a summary vector, and then employ a discriminator that evaluates the probability of a patch representation to be part of this summary vector's source compared to the probability of a patch representation coming from a "corrupted" graph being related to that summary. This adversarial setting is thus teaching the encoder to generate representations that are most likely to summarize the whole graph, we can better understand the setting through the figure 9. Even though my approach would need a graph classification algorithm, the fact that this work advances node encoding/classification doesn't mean it can't be used for graph approaches: This readout function could be morphed to provide a classification layer instead of a summary.

In the context of GNNs, aggregating functions applied on nodes are the ones that gather the features of their neighbours, while readout are the ones that take the final feature representation of nodes

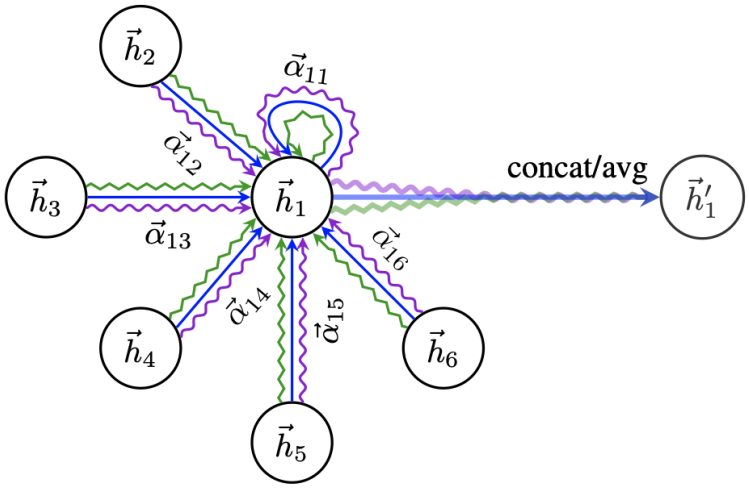



Figure 8: Illustration of how a multi-head attention network would be applied in a Graph Convolutional setting. Here, the 3 attention coefficients per pair of nodes will aggregate more features per neighbour which will be concatenated or averaged to obtain the new feature representation of node $h_1$, [37]

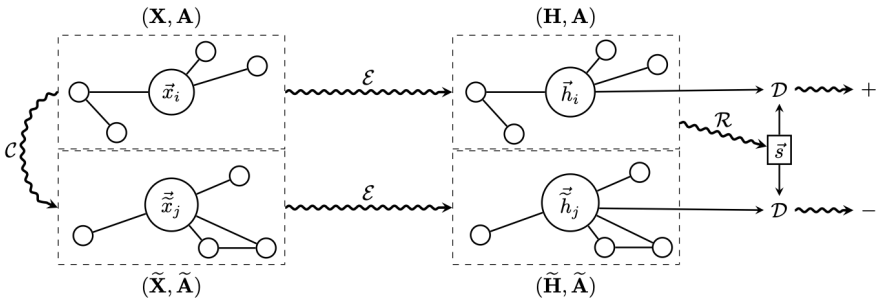



Figure 9: Diagram of the Deep Graph Infomax approach. A corruption function is used to generate a modified graph $\tilde{\mathbf{X}}$ from the given one $\mathbf{X}$. The encoder then encodes both version of the graph and creates a summary from the encoding of $\mathbf{X}$. The discriminator then guesses which encodings are more likely to be related to the summary, thus teaching the encoder to generate more representative encodings, [36]

and combine them into a representation for the whole graph. [35] prove that GNNs cannot be more powerful than the Weisfeiler-Lehman test [40] giving examples of graph structures that [15] and [39] architectures cannot distinguish but find the needed conditions for the aggregating and readout functions to have in order to match that maximal power and propose their own architecture. They found that these two architectures for example, are not injective, some different node neighbourhoods can map to the same encoding. They prove that there exist GNNs that can be as powerful as the WL test, ones that map isomorphic graphs to the same representations and non-isomorphic ones to different representations. Using MLPs and the summation of one-hot encoded vectors when possible to update the node representations and concatenating all the node representations as the readout function, they built a model that is as structurally powerful as the WL test.

### Non Convolutional Approaches

The survey [13] also reviews approaches in Graph Neural Networks that are not based on convolutions, like Recurrent GNNs, Graph Autoencoders, and Spatial Temporal GNNs. Kernel methods are briefly compared to these deep learning methods but a more comprehensive survey on Graph Kernel methods was recently released [21]. Microsoft Research also published a talk on Graph Kernels on YouTube [13] which explains concepts and tools used in most of the reviewed methods. Other than using them to

[13] https://www.youtube.com/watch?v=xwVOarJGD7Q

classify nodes and graphs, or to encode them, geometric deep learning methods can be used to cluster data for various reasons like indexing, community building in social networks, recommendation systems, etc. [10] made successful progress towards that goal by introducing DiffPool, which has also been used by other later models in [13] as a pooling strategy to use in between convolutions.

[21] categorizes graph kernels depending on their types of approaches, some compare two nodes by analyzing their neighbourhood structures and assigning them a label and comparing the label distributions of the nodes across the graphs, the most famous such kernel is the 1 dimensional WL kernel also known as the colour refinement algorithm. Assignment or matching based approaches try to find the best possible matching between two graphs in regards to a certain node similarity function. Subgraph approaches are ones that consider graphs as just bags of edges or vertices and then compare the statistics of these bags when given two graphs, a bit like TF-IDF in Natural Language Processing. The last category of approaches are the kernels that launch random walks on the two given graphs and look for certain attributes. They finally provide a decision tree that researchers can follow to decide what kernel to use, depending on whether or not the vertices have continuous attributes, if they are important at all, if the graphs are large, dense, or other specificities.

Recurrent graph networks were the first architectures that used message passing techniques to update node representations, which inspired spatial-based graph convolutional networks. Similar to LSTMs, Recurrent graph networks ask each node to not only produce an output that is fed into the next layer, but also to save an internal state that will act as some sort of memory in the node, the networks runs and keeps training until the node representations reach a convergence. What [18] did is to add Gated Recurrent Units, borrowed from the work of [19], to unroll the recurrence for a fixed number of steps, and not indefinitely until convergence, creating networks called GG-NNs. Running the network until convergence uses less memory of course, but since a convergence needs to be met, the initial parameters of the network have to be constrained, which reduces the expressivity of the network. As the figure **??** suggests, we can stack multiple GG-NNs in a row to allow the network to have a sequence of outputs for example modeling the interactions between users in a society. [18] tried that and named them GGS-NN (Gated Graph Sequence Neural Networks).

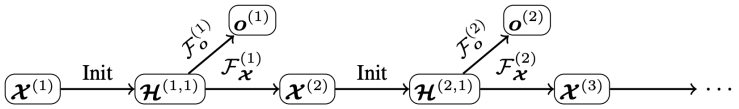



Figure 10: Architecture of a GGS-NN where each $F$ is a GG-NN predicting either an ouptut $o$ or node labels $X$ from feature representations $H$, [18]

DiffPool, proposed in [10], is a module that GNN architectures can employ as a hierarchical pooling strategy. Their strategy learns to output a coarsened graph with less nodes from an input graph, and be able to pass this result to another GNN, it generalizes across graphs with different edges, nodes, and topologies. They do that by building layers that don't just learn to provide for each node representation vectors that are good to achieve classification tasks but also other vectors that are used to assign a cluster for each node, these additional vectors per layer are stacked together and form what they call a Cluster Assignment Matrix.

## 2.3 Fabula AI

All the solutions I have presented in this section fall under the categories proposed in the survey [8] which are diagrammed in figure 11. Detection fake news circulating in tweets is a problem that requires a lot of context analysis, as the content itself is usually delivered by attackers in a subtle way to fool victims. It doesn't mean the analysis of content is something I will skip. So I need an approach that combines both aspects. In addition to the context happening around a tweet, like the rate at which a

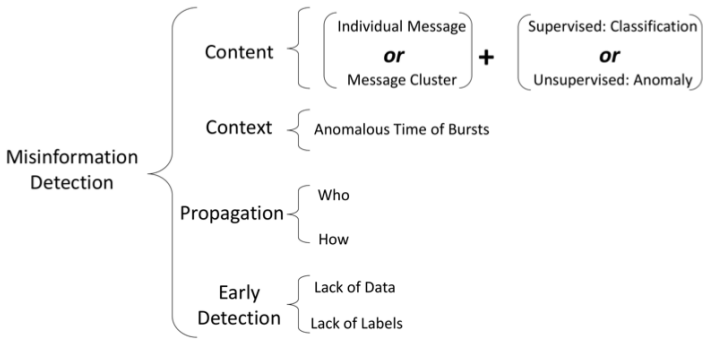



Figure 11: Overview of the categories of misinformation detection techniques available to choose from to tackle my problem, [8]

news is repeated or the local time of the tweet, the way it has been propagated and caught traction, and then entity that propagates it is sometimes a good tell of whether or not it's fake. [1]'s work combines all these aspects and in addition provides promising results when taking into account tweets that are at the early stage of a news' spreading. It is the work I'll mostly try to reproduce in this thesis. Unlike the deep fakes detection work in section 2.1, this work combines content and context with deep learning to classify data points as fake or real. Graph Convolutional networks will allow me to combine all these concepts when considering the features I want to feed my models and as we have seen in section 2.2, there are many recent implementations of Graph Convolutional Networks. Spectral networks, however, be it the ones introduced in [14], ChebyNets [17], [15] or CayletNets[16] must be trained on a specific graph size and/or topology which would mean we have to add dummy nodes to our graphs whenever they don't look like the chosen topology and that would add a lot of overhead complexity to the data processing without any guarantee that it would work as no paper have used this workaround as far as I know. Instead, [1] bases their work on top of ConvNets that work for arbitrary graphs, I did the same and could have merged the work of [37] but kept the aggregation function simple to begin with. It's definitely something to consider when improving the project but building the pipeline alone looks like an ambitious task already. As the availability of fake:real labels for the tweets is not a big problem for now, I steered away from semi or unsupervised learning, but it could be applied if I went for a bot classification of fake news spreader vs honest user where a node classification approach would most probably be the way to go. After looking at their github repository [14] it seemed to me that trying to implement [35]'s work and get closer to the performance of a WL test would have a smaller return on investment than going with a standard solution from a Graph Deep Learning library and concentrating on the data preprocessing. I played around with graph kernels for a few days in my internship using a library called GraKel [15] but didn't get results much better than a coin flip, and ran into a lot of runtime problems when trying to add more features to my graph representations that I gave it up and focused on the deep learning approach. Just like the attention mechanism, DiffPool and the Gated Recurrent Units could be tweaks to add to the model once I have more understanding of the behavior of my pipeline so I started without them by just tryign to replicate [1]'s work.
Fabula AI actually achieved the same thing that Astroscreen wanted to have as one of their products. The goal of their work is to train a Graph Convolutional Network that takes as input a tweet and its retweets, or a url and all the tweets or retweets sharing it, and classify it as fake news or real news. The difficulty of detecting fake-news is higher than existing content-based analysis because interpreting the news require knowledge of the political/social context or 'common sense' which is what prompted [1] to research context based approaches. Recent studies had pointed different spread patterns between fake and real news, so the context to be used here is the propagation of the news across a social media platform, justifying why they trained a GCN that takes as input a tweet and its retweets. Not only do

[14]https://github.com/weihua916/powerful-gnns
[15]https://github.com/ysig/GraKeL

propagation-based approaches leverage the context of the tweet instead of just its contents, but it also means the work is language independent and can resist much better to adversarial attacks, unlike what's happening with all the Deep Fakes wars previously described. It is much harder for an attacker to tamper the news spread in a network than to modify the GAN that is generating fake videos of Obama. Other approaches have been using propagation-based techniques, extracting features like centrality, degree, connectivity, cliques, etc. from graphs but the work has so far only been manual, and [1] are the first to implement them in a Geometric Deep Learning setting. Doing that, they achieved a ROC AUC score of 93%, their model is able to detect fake news when only looking at retweets that occurred within a few hours of the original tweet, and whatever it learned on old data is still applicable to more recent news, which makes it well stationary. They narrate how they gathered their dataset but don't publish it, and I won't explain their model setup here as I will introduce it in my approach section (3).

# 3 Approaches

## 3.1 Reproduction of Fabula's Work

The starting point of my project was logically to reproduce Fabula AI's work and see where I could go from there. Their paper describes their overall approach but didn't really go into details when talking about few keys decisions they had to take. Some of these points that I felt the need to shed more light on were:

- Neither their dataset was provided, nor their data collection code. They say that they follow the protocol in [2]. Unfortunately [2] did not release theirs either.
- In their "Features" point inside their "Dataset" section, they briefly mention the different features that they look at but don't explain how they extracted and used them in their models.
- They use a feature called "Credibility" that they get for each user using their data collection technique, but since their tool is closed source, it's not something that I could replicate in my work.
- Their model weights are not released. In addition, they don't share the parameters needed for reproducing the training.
- Their graph input generation is pretty straightforward, however, it requires deep knowledge of each user's followers or followees. And they don't share how they efficiently got around Twitter's stingy limitations.

#### Dataset creation

I had to either build my own dataset for the thesis or find one that could serve its purpose. The techniques used will be explained in section 4 but I initially started gathering recent tweets related to a certain real or fake statement using an app called Sysomos. Due to limitations in terms of manpower and time I could not afford in this data gathering, I had to settle for FakeNewsNet, a dataset released in 2017 which has some caveats that I will introduce later.

#### Features, Credibility

I did not have much choice when deciding what to do here. As I was using the Twitter official API, and couldn't know where [1] got their feature values from, I had to make do with whatever json objects the Twitter API returns me when I fetch information about a certain tweet. Fabula AI are using a feature called "Credibility", but the Twitter json objects do not contain any credibility score, the closest key-value pair to this concept would be whether or not a user is verified.
For a tweet **T** and its user **U**, I added to that a combination of: the number of followers, followees, statuses, favorites and listed tweets that **U** has, the retweet and favorites count of **T**, the difference in time with the original tweet if **T** is a retweet. The amount of time that **U** has had an account when tweeting **T**, and whether or not their account is protected.
Later in the project, I combined the content of **T** with these contextual features, by embedding **T**'s text and **U**'s bio and concatenating the resulting vector to each node's feature vector. Tables 1 and 2 list the keys present in the json objects returned by the Twitter API and indicates whether each one of them is used or not when building my feature space. Values that are just numbers are used as is, timestamps are combined with others to compute the delta, or the time of day is extracted. Categorical data is used as a one hop encoding, and text data is encoded into a feature vector. Then all these features are concatenated into one feature vector per node.

| JSON Key | Description |
|---|---|
| created_at | Timestamp of creation |
| id | Unique identifier |
| id_str | Same but in a string format |
| text | Text content of the tweet |
| truncated | Boolean indicating if tweet has been truncated |
| entities | Object containing urls, hashtags and mentions |
| extended_entities | Object containing media contents |
| source | Device or platform used to create this tweet |
| in_reply_to_status_id | Unique identifier of the status this tweet is replying to |
| in_reply_to_status_id_str | Same but in a string format |
| in_reply_to_user_id | Unique identifier of the user this tweet is replying to |
| in_reply_to_user_id_str | Same but in a string format |
| in_reply_to_screen_name | Username of the user this tweet is replying to |
| user | User object |
| coordinates | Latitude and longitude of the device when tweeting |
| place | Object containing information about the location of the tweet |
| quoted_status_id | Unique identifier of the quoted status |
| quoted_status_id_str | Same but in a string format |
| is_quote_status | Boolean indicating if this tweet is quoting another status |
| quoted_status | Object containing the status that this tweet is quoting |
| quote_count | Number of tweets quoting this one |
| reply_count | Number of tweets replying to this one |
| retweet_count | Number of tweets retweeting this one |
| favorite_count | Number of users marking this tweet as one of their favorites |
| favorited | Boolean indicating if the logged-in user marked this tweet as one of their favorites |
| retweeted | Boolean indicating if the logged-in user retweeted this tweet |
| possibly_sensitive | Boolean indicating if this tweet could be sharing sensitive content |
| filter_level | Mininum connection quality needed to stream this tweet |
| lang | Language of the tweet's content |
| matching_rules | Collection of keywords from the search query that matched this tweet |

Table 1: Keys of the tweet objects returned by the Twitter API. Blue: Value used directly to create a feature. Green: String value passed by a text encoder to create a feature-vector. Yellow: Value that was used to preprocess the data, either by helping build the graph connections or by leading to extra values. Red: Value unused in the feature building phase. Gray: Value that is redundant because the same features can be extracted from another key-value pair.

### Model Training

The field of Graph Convolutional Networks is still young and not much research has been done in applying GCNs to social network data. [3] gives an example of its applications on 4 datasets [4],

- AIFB modeling the relationship between staff in the AIFB research institute
- MUTAG describing different molecules that may or may not be mutagenic.
- The British Geological Survey BGS and
- The Amsterdam Museum AM contain information about archeological and geological artifacts that are related to each other depending on their various properties.

| JSON Key | Description |
|---|---|
| id | Unique identifier of the user |
| id_str | Same but in a string format |
| name | Name of the user |
| screen_name | Username of the user |
| location | User-defined location of the user |
| description | User-defined bio |
| url | User-defined url describing their account |
| entities | Object containing urls used in the user's description |
| protected | Boolean indicating if the user's tweets are hidden from strangers |
| followers_count | Number of accounts following this user |
| friends_count | Number of accounts this user follows |
| listed_count | Number of public lists this user is a member of |
| created_at | Timestamp of account's creation |
| favourites_count | Number of tweets that this user marked as one of their favorites |
| verified | Boolean indicating if the user has a verified account |
| statuses_count | Number of (re)tweets published by this user |
| profile_image_url_https | url of the account's profile image |
| profile_banner_url | url of the account's banner image |
| default_profile | Boolean indicating if the user modified the background or theme of their profile |
| default_profile_image | Boolean indicating if the user kept the default Twitter profile image |

Table 2: Keys of the user objects returned by the Twitter API. **Blue**: Value used directly to create a feature. **Green**: String value passed by a text encoder to create a feature-vector. **Yellow**: Value that was used to preprocess the data, either by helping build the graph connections or by leading to extra values. **Red**: Value unused in the feature building phase. **Gray**: Value that is redundant because the same features can be extracted from another key-value pair.

Most of the existing research on the matter is either using these datasets or ones similar to them. None of which resembles our case as much, and so I had no guarantee that their suggested models would be suitable for my approach.
This youth of the field naturally affects the arsenal of tools available to help achieve the training of a GCN. At the start of the project, the most advanced graph deep learning library is DGL[16] and yet the version[17] I am using didn't have the implementations of most of the state of the art approaches. That didn't stop me from using it, contenting myself with the use of the vanilla GCN introduced by [3] with simple the message passing approach of summing then averaging features of nearby nodes. DGL runs on top of PyTorch[18] and can seemingly be integrated in. As figure 12 illustrates, a 2D convolution applied onto an image could be viewed as a graph convolution if every pixel is interpreted as a node and connected to each of its 8 neighbouring nodes by an edge. In graphs, however, neighbours are unordered and their sizes could vary, so the filter function has to take that into account and be rotation invariant, [35] and [34] among others in the 2.2 section mention sophisticated functions to use but I simply went with the solution of averaging the feature vector of the neighbours. DGL makes it easier to represent the graph but I still had to define the PyTorch modules for each GCN layer, and the filter which is composed of the message passing function that fetches the neighbours feature vectors and the reduce function that combines them. I chose the same model architecture as the one Fabula AI had (figure 13) except that it was easier to integrate with Astroscreen's codebase if I had a Sigmoid output with one value instead of a Softmax one with a two-dimensional vector as the output. That

[16] https://dgl.ai
[17] Version 0.4.1 released in early November 2019.
[18] https://pytorch.org

doesn't affect the outcome of the training or predictions however. [1] don't specify the widths of their layers, so I empirically tried different ones, and their used activation function was a Scaled Exponential Linear Unit but I preferred to use ReLu for simplicity.

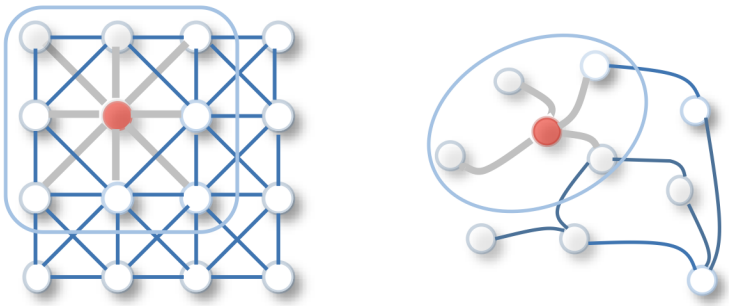


Figure 12: Image Convolution vs Graph Convolution computing the new value of the red pixel/node from its neighbours, [13]

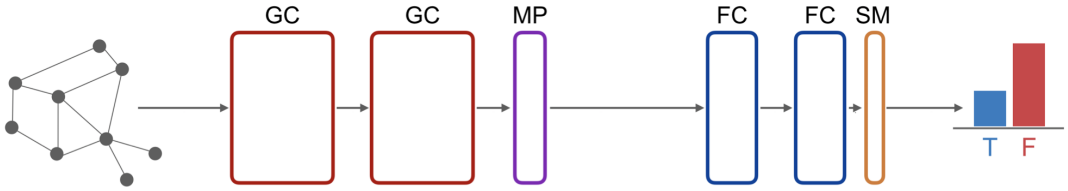



Figure 13: The architecture of the network used in Fabula AI's work. GC = Graph Convolution, MP = Mean Pooling, FC = Fully Connected, SM = SoftMax layer, [1]

I used the Ignite library[19] for an event-based training logic which made the code more concise and extensible for plugins and new methods. The most notable extensions I implemented to help with the training were: A cyclic learning rate scheduler that periodically changes the learning rate to fully explore the optimization space, a Reduce on Plateau learning rate scheduler that lowers the learning rate whenever it runs out of patience of waiting for the validation accuracy to improve, and an Early Stopping scheduler that stops the training if a certain number of epochs go by without any improvement to the model thus saving computational cost and time. In addition to DGL, I used nxgraph[20] to construct and visualize graphs, and Visdom[21], a tool released by Facebook Research and similar to Tensorboard, to visualize the training metrics.

### Graph Generation

In order to contruct these graphs with nxgragh, a pattern should be followed to know where to draw the edges connecting the different tweets.
[1]'s algorithm was to consider cascades of tweets. A cascade is a spanning tree with the head being a tweet and the other nodes are all of its retweets. To decide where each node **N** is connected, we search for the most recent other node **P** such that **N**'s author **A** follows **P**'s.
You might ask yourself what happens if **A** follows none of the others in the tree. Well in this case, they just link them to the retweet with the most popular author. This seemed pretty straightforward for me, and I went ahead with it, but encountered a few problems that they didn't provide a solution to.

- First is that the Twitter rate limit per API key for getting the list of followers for a certain user **U** is 15 requests per 15 minutes. And each request can only get 5000 followers of **U**. Which means I

[19] https://pytorch.org/ignite/
[20] https://networkx.github.io
[21] https://github.com/facebookresearch/visdom

needed to get as many authentification keys for my project as possible, and I could only get 7, which gives me 7 users' lists of followers per minute, which translates to merely than 10000 per day, and that's if my gathering code is permanently running.
For this reason, I had to go with a shadier solution and scrap this list of followers from the web version of twitter mobile. I used an open source tool called Twint[22]. It's considerably slower than the official API but I wrote a concurrency wrapper around it to fetch multiple users' followers in parallel. In the end I was averaging 2000 lists per hour compared to the previous 420 or could even go faster when using two AWS machines instead of one. I ended up integrating the code into Astroscreen's codebase as a back-up when they needed to process more data.
This approach didn't work flawlessly, however. Requests were failing due to connection limitations and shutdowns from Twitter, or dropped packets so I had to make it fault tolerant and run the pipeline multiple times to cover all missed instances.

- Second is that some users have a ridiculous amount of followers, and fetching all their usernames would take an eternity. Others have their profiles set to private which prevents the tool from scraping their followers. In both these cases, I give up on getting their followers but still save their count and follow the same logic as Fabula AI when I don't find any parent node to a certain one: I still consider them as suitable nodes when looking for the most popular recent retweet.

In addition to a cascade one, [1] had a url-based classification, which considers all cascades related to a single claim when classifiying it. I also implemented that by connecting all the spanning trees heads to an empty root node when generating the graphs.
Figure 14 shows the different graph generating approaches I used. I reused the same cascade approach, but also added a more sophisticated way of generating a graph from a tweet and its retweets: "Spreads", which differ from cascades by not just considering tweets as either statuses and retweets, but also considering if it's a reply or a mention, which adds more connection possibilities to the graph, but at a cost of a complexity when building it. And then I did the Url approach by combining spreads instead of cascades.

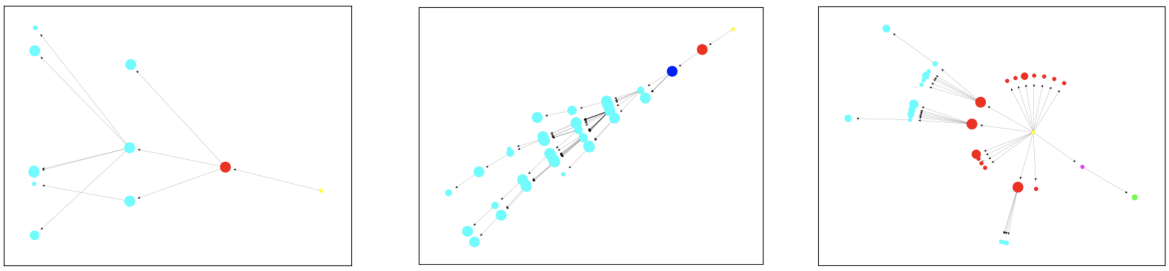



Figure 14: The three graph generating approaches that I used. (a) Cascade: Only considers a tweet and its retweets, connecting each retweet **R** of author **A** to the most recent re(tweet) with an author that **A** follows. (b) Spread: Also considers whether a tweet is a simple status, a reply, a quote, or a mention which adds more node types. (c) Url: Unlike [1] which connects cascades together to build a url-based graph, I connect consider all tweets and retweets regarding a topic and combine their spread. Additionally, if a tweet is replying, quoting, or mentioning another that isn't in my collection, I create a "shadow" node to connect it to.
The radii of the nodes grows the more an author of a tweet has followers. Color code: Yellow: Topic node to connect all original statuses nodes to. Red: Original status tweet node. Green: Reply node. Blue: Quote node. Teal: Retweet node. Purple: Shadow node.

[22] https://github.com/twintproject/twint

## 3.2 Trial of Text Based Approach

As there are no open benchmarks for my approach yet, I need to compare it to some other approach. The most commonly used fake news detection technique so far is content-based classification.
I would have to only consider the text of the tweets when classifying them, losing the contextual features described in 3.1. To be fair, we would still be looking at the content of the whole cascade and not the single tweet as the added opinion present in retweets would give beneficial information to the model.

### Natural Language Processing Classification

In order to classify the tweets by their text contents, I had to use some Natural Language Processing techniques. One state of the art approach is to use Transformers, more precisely, Bidirectional Encoder Representations from Transformers BERT.
The BERT [5] technique pretrains its architectures on two natural languages tasks: Masked Language Model MLM and Next Sentence Prediction NSP. These two intented skills would be very helpful to detecting contested opinions in a thread of retweets. NSP could make the model better understand the junction between sentences (here a thread of retweets) and MLM would signal discrepancies between them. This was all a hypothesis, and I had to test it.
So first thing to do was to convert my cascade data into text files that could be fed to BERT. The latter uses two keywords that formalize the input. [CLS] is used to signal the start of an input example and [SEP] to seperate between parts of that example. It was initially intented by the authors for modeling Questions and Answers, but in my case, I used it to seperate between different retweets in a thread.
Second was to choose a pretrained architecture of BERT and fine tune it on our fake news classification task and dataset. Having a modest compute power, I went with the Transformers DistilBert [6], which is a distilled replica of the original one by [5] and has only 66 million parameters compared to 340 million. Their pretrained model is offered as part of their open source librabry called Hugging Face[23].

### Transformer Training

I reused most of the codebase written for the graph neural network training pipeline and adapted the Ignite training loops to handle text input.
The general output of the pipeline is still the same, both models classify the tweet threads as either fake or real and I have gathered the same metrics as the graph case to be able to compare them.
One might wonder that using the pre-trained model DistilBert could bias my findings. After all, HuggingFace put in so much resources and data to train it, that adding a couple thousand tweet threads is negligible. The language model would have been trained so much more than the GCN even though it wasn't done on the task of fake news detection, but it still gathered enough language understanding. To fight that, I retrained DistilBert from scratch on just my dataset and compared the results.

### Bag of Tricks for Efficient Text Classification

I did not feel comfortable comparing the graph convolutional approach to just one content classifying approach. [5]'s first version was submitted in October 2018, less than 18 months ago and has built on the advancements of a lot of successful natural language processing research. Even when training from scratch, the Bert approach is benefiting from the freedom of millions of trainable parameters.

[23] https://huggingface.co

DistilBert has 6 layers with 748 hidden dimensions and Bert has more than double that. So I went with a slightly older paper [41], that introduce a lighter approach, called FastText, it considers sentences as bags of ngrams (where a ngram is n consecutive words), gets their feature vectors from a trainable embedding bag, while aggregating the results in one final sentence embedding vector. This sentence vector could then be used just like in the DistilBert approach and pass it through a linear classification layer. To be consistent though, I also added a pre-classifier layer that reduces the sentence embedding vector a lower dimension, before converting it to a single scalar value. Figure 15 illustrates how a simple text sentiment classification approach could be achieved with an embedding bag. The embedding bag layer is the core of this archtecture, and I will first try training it from scratch, and then load the pretrained vectors of the different ngrams from the FastText paper [41].

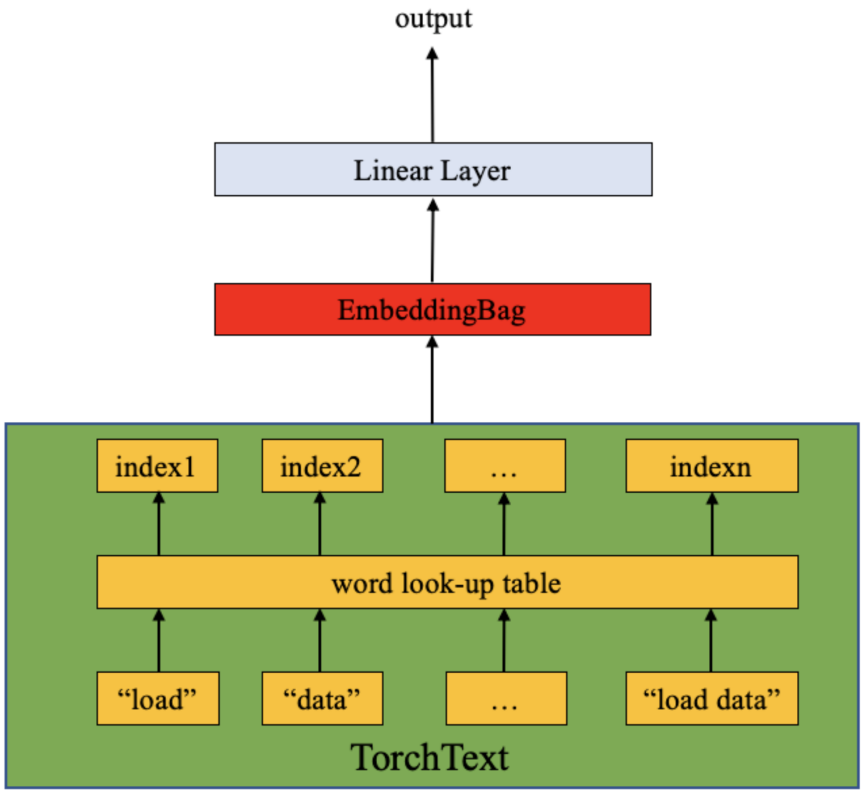



Figure 15: Core architecture of a Text Sentiment Classification model: When fed a sequence of ngrams, each one is translated into an index using a vocabulary lookup table, then a trainable embedding bag converts the indices to feature vectors and aggregates them, then fully convolutional layers follow to finally reach the desired output format, PyTorch documentation[24]

### 3.3 Combining the Two Approaches

After trying both approaches separately, the time came to combine them and see if they perform better. [1] actually used content information of tweets when creating their graphs, and any solution in the wild would not refuse content data if it's present.
[1] encodes the tweet content and the user bio, each into a vector, and uses them as input features to their GCN nodes. However, they gave no details on the size of these vectors, or which language model they were using, so I could not draw a perfect comparison plan.
I went with the most straightforward solution I could think of, and remove the classification layer from DistilBert to use the pre-classification layer and the embedding of each textual input. Even with a distilled version of BERT, this vector was still very big and having many nodes or graphs would mean they can't load on the GPU, I added a smaller layer between the pretrained 748-dimensional

pre-classification layer and the binary classification layer before training on my classification task. Finally, when building my graphs, I would pass each tweet content and user bio through this modified DistilBert classifier and extract the values after the added layer.

### 3.4 Scrapped Attempt at Hashtag Bombing Detection

Towards the third month of my time at Astroscreen, the leadership decided to have a new vision to the company and asked me if I could pause the work on my project and focus on a new type of attack that can affect their clients. Brand polluters can sometimes setup bots that wait for a certain hashtag to gain some traction in a community and then artificially boost it by retweeting or crafting tweets denouncing the brand. I spent a week trying to replicate the data gathering approach that I had followed for the fake news detection approach to get tweets and retweets sharing certain hashtags from the app Sysomos. I was restricting my search to the hashtags that are the most popular in the UK, as most of the clients are from there, gathering the ones that have a high concentration of Authority-1 users (which is a label that Sysomos gives to users they rate as likely to be bots.). However I ran into three problems that scrapped this whole approach.

- Sysyomos only lets free users download chunks of 50,000 tweets at a time, which makes it extremely time consuming to download the spreads of most manipulated hashtags. Also, having big graphs would make it impossible to load on my small GPU.
- I could not just sample tweets from these huge collections, as my technique focuses on analyzing the spread of a news. Sampling the tweets would remove most of the connections between the nodes of my graph and make it a bag of tweets instead. Focusing on the initial tweets of a burst could have been a solution, but finding that initial spike and downloading the tweets for it, all from the web version of Sysomos, without access to the CLI or the API was very time consuming.
- For a very popular hashtag, these bursts of tweeting never started at a point where no one was tweeting it, but there was always around 1000 per hour happening, so if I'm only considering 300 or 500 tweets to have in my graph, that spike isn't even going to matter.

With all that, even if I was able to find hashtags that are small enough to generate reasonable graphs, and have spikes that start at near zero tweets per hour before attracting attention, I needed to label them as either manipulated or organic, and halfway through the next week, all but one of the data analysts in the start-up were laid off (the last leaving less than a month later) which meant I had to do it all by myself and would never have been able to provide results for the approach.
Having analysed all these problems, I re-voiced my concerns and went back to fake news detection, but it would be an interesting project to tackle in the future, maybe with a different graph setup.

# 4 Datasets

## 4.1 Building My Own Dataset

[1] did not release their dataset with the paper. They mentioned the fact-checking websites they got their claims from, but didn't release the code used to gather the tweets related to them either.
I spent some considerable amount of time at the start of the project trying to replicate their work. The steps they had followed, and the ones I had to do again were:

1. Get articles from fact-checking websites, that prove or disprove some hot subject.
2. Extract the claim, and possible URLs defending or attacking this claim, from each article.
3. Get as much tweets as possible mentioning this claim or sharing the URL.
4. Fetch their tweet info from the Twitter API, and their retweets if possible.

Another logical step followed in their process, which was to judge whether or not a URL, or original tweet was supporting or rejecting the claim. This was definitely something I couldn't do, as I was working solo on the data gathering, and had no funds to pay an outsider's help like [1] did.
Even extracting the possible URLs from the article was hard to do on one's own, as many URLs in the article had nothing to do with the claim, sharing either some background on the subject, or explaining a similar case.

#### Scraping The Fact-Checking Website

I chose the website Snopes [25] to gather my claims from. It had very recent topics in their newest fact-checking articles, their Fact-Checking page [26] was very well structured (figure 16), and their articles (figure 17) had clear truth or false labels, sometimes even differenciating between rumors, misattributed claims, miscaptioned images, scams, etc.

This made their data easy to scrape, using a Python tool called Scrapy [27], with which I defined spiders, their behaviors when getting HTML responses, and what to look for in a page. Having a well structured front-end in their website, it was easy to identify the claim, and the rating. But getting the URLs would have been harder, as I'd have needed to redirect the link, and analyze whether or not the shared article is supporting the claim, neglecting it, or is unrelated. Other works like [7] have done something similar and used heuristics to filter them, but I would have ran out of time had I tried to replicate it.
As a start to my project, I fetched 50 claims of each False and True ratings to visualize the spreads graphs and see if anything clear strikes out.

[25] https://snopes.com
[26] https://snopes.com/fact-check
[27] https://scrapy.org

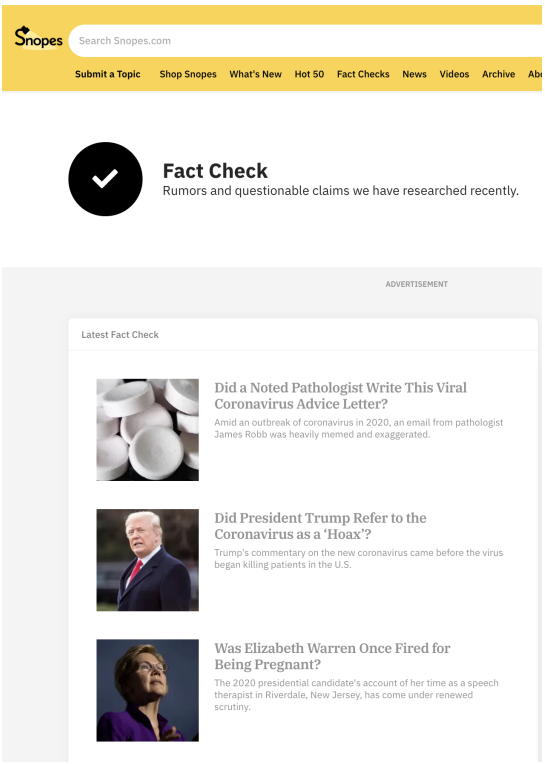


Figure 16: The Snopes website fact-checking tab

### Fetching the Tweet IDs

To do that I used a tool called Sysomos [28] that Astroscreen was using. It allows the user to search Twitter's content that is less than a year old. I would build queries containing the claims, any relevant URL I would find in the fact-checking articles and some combinations of words from the claims and it would return the ids of the tweets and some additional information for each. I discarded that information anyway as I was going to fetch the tweet info from the official Twitter API anyway.
One attribute that I could have used for training was an "Authority" score that Sysomos gave to each user, but since it was a black-box, and I wasn't sure if Atroscreen would have been able to use Sysomos for real time inference using my model, it was too risky to base my work on it.

### Hydrating the Tweet IDs

With the official Twitter API, unlike searching for tweets, getting their retweets, or the followers of users is way easier. The limit per 15 minutes windows is much higher. So once I had the ids, all I needed to do was write a scheduled script and fetch the allowed amount of tweet info every 15 minutes.
However, one problem arose, not as much in this case as in the case of the FakeNewsNet dataset (explained later 4.2): Some of these tweet IDs corresponded to tweets that have been deleted, or from accounts that are private on Twitter which forced me to just simply discard them.

### Outcome

These steps resulted in me having in a toy dataset to test whether or not my initial training pipeline works. I initially tried training a GCN model that just operates on the topology of the spread graphs

[28] https://sysomos.com

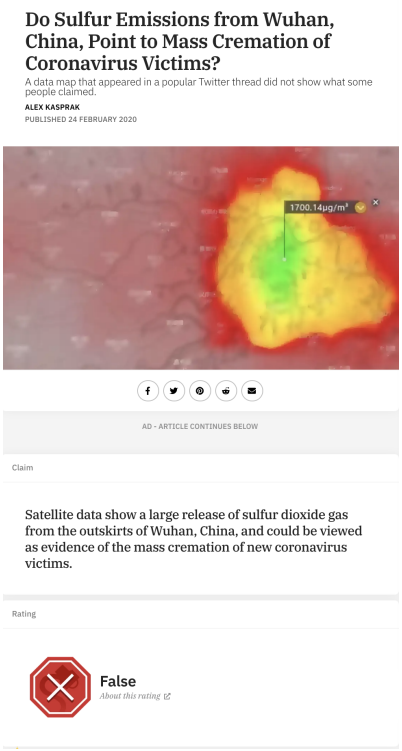



Figure 17: An example of a claim article on Snopes

and didn't use any social network context, then added these extra features for a finer outcome.
The acceptable accuracy I was getting when training the model on this dataset was only the training one. The validation accuracy was nowhere near as good and the reason was very probably the size of the dataset.
Since I only had 50 claims, I had just as much URL-graphs (as explained in 3.1). With so little training samples, it's clearly not suitable for deep learning, so I built cascades from these urls. However, as [1] mentioned, very short cascades are not following the spirit of this spread classification approach as analyzing them would go against the idea of using the graph structure. This meant that if I limited the size of cascades to be at least 6, like in [1], the number of cascades I had dropped from 34247 to 383, which indicates that my dataset was mostly built of tweets with little to no replies.
The figure 18 confirm that. The x axes in the graphs indicate the minimum length of a cascade or spread. Just by setting a minimum length on these cascades, the total number of cascades or spreads is divided by around one thousand. Towards a size of 3 or 4 tweets, the count of such cascades drops down to the order of $10^2$.
Without any doubt, a dataset of a larger size was needed to conduct a good deep learning experiment.

**Drawback**

Gathering these claims, extracting the search terms, waiting for the replies from Sysomos, extracting the csv files containing the tweet IDs and then running the hydration scripts proved to be very time consuming, taking around 5 mins per claim, in order to get an acceptable amount of related tweets for that single claim.
In addition, it's a very mind numbing task and not as exciting to add into a thesis project when I only had a few months to achieve a project. Had it been helpful for other future projects, I could have considered spending some time to build a dataset.

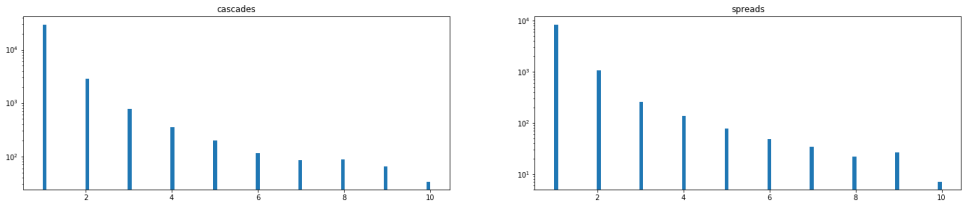



Figure 18: Tweets Count for In-House Dataset

## 4.2 FakeNewsNet

Projecting how much time it would take to get a reasonably sized dataset up and running, I searched for already gathered data about fake news and found [7]. It was not conceived for a geometric deep learning or a graph theory approach but since it lists for each claim all the tweet IDs that support it, I would link these IDs using the graph generation algorithm explained in 3.1 and have a bigger dataset than in 4.1.

### Initial Contents

FakeNewsNet was setup in 2017 and contains two datasets inside it.

- Politifact consists of claims gathered from the Politifact website [29] and the tweets supporting these claims. Similarily to 4.1 which was coming from Snopes. It has 624 real claims and 432 fake ones though and around 570,000 tweet IDs. FakeNewsNet only lists the IDs of original tweets and not their retweets or replies, so after gathering the retweets for these tweets, the count jumps to roughly 1,152,000 tweets.
- Gossipcop consists of rumor articles circulating about celebrities and the tweets sharing these articles. Most of the articles fact-checked on the gossipcop website [30] had a false label, so the authors of [7] had to gather extra real rumors from other websites and built this even bigger dataset. It contains 5323 false rumors and 16817 true ones for a total of 1,708,046 tweets, or 3.2 million when including their retweets.

### Hydrated Contents

When hydrating the tweet IDs, the sizes of these datasets fall to roughly 1.08 million tweets and 1.9 million respectively, losing a lot of tweets because they can't be fetched from Twitter.

Figures 20 shows that the Gossipcop dataset is mostly constituted of non-retweeted tweets. The total number of tweets drops to 730,000 after removing these cases, 38% of its starting size and even 520,000

[29] https://www.politifact.com
[30] https://www.gossipcop.com

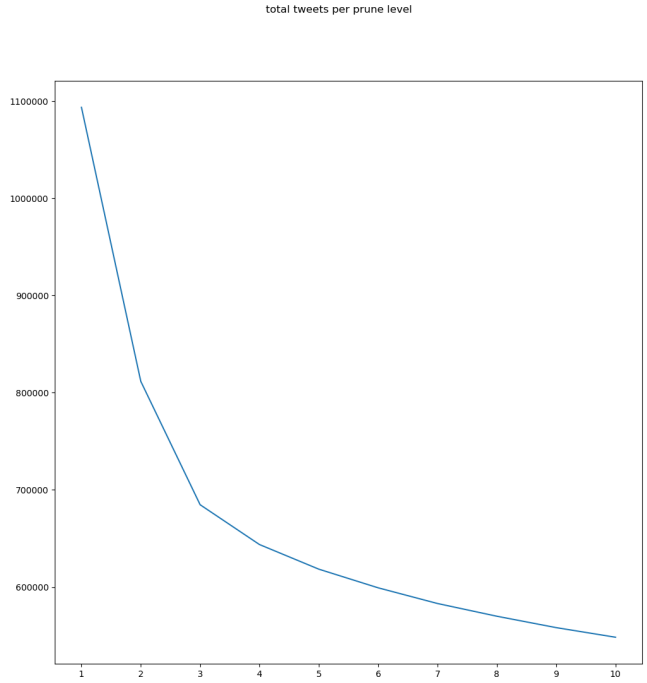



Figure 19: Tweet counts for Politifact when considering different minimal cascade lengths

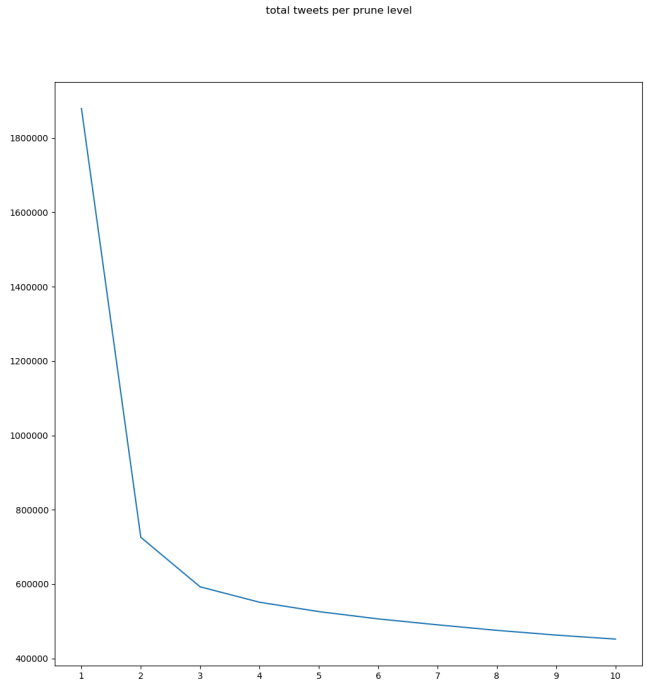



Figure 20: Tweet counts for Gossipcop when considering different minimal cascade lengths

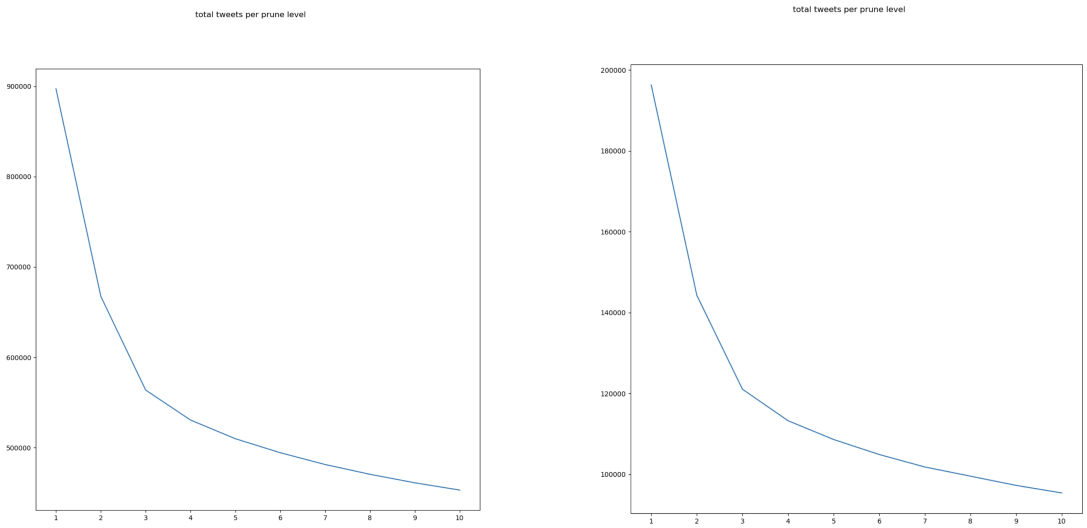
when I consider the same minimal cascade length of 6 as [1]. Politifact, however, is more spread out in terms of cascade lengths as the figure 19 shows, dropping to 810,000 when removing non-retweeted tweets, and 610,000 when the minimal cascade length is 6, that's 75% and 56% of the starting size.

**Splitting the Contents**

I kept both datasets to work with as they both have suitable sizes and the difference in their styles could be interesting later on. I wanted to split each one of them into train and test directories following a 80:20 ratio but still preserving the class balance and as much characteristics as the undivided one. These characteristics that I was looking for are the distribution of small and large cascades and the percentage of spreads with little followers information (the reason for most of them is because I couldn't fetch the followers of certain users in these cascades because of their profile privacy, or if they were banned, or the sheer number of followers they have).
To have good dataset splits, I split each one three times (A, B, C) and plotted the data statistics that I was looking for. I ended up choosing the split B for gossipcop and C for politifact because they had the most similar size distributions as the undivided ones, and between their test and train subsets.
The 3 plot figures for each dataset are:

- The tweet counts for different minimal cascade lengths show how much a dataset's number of tweets shrink when I increase the minimum size of cascades. If a cascade has less tweets than that, I just discard it.
- The spread size distribution contains 12 subplots each. Each subplot is a histogram representing the number of graphs in the dataset that have the size indicated on the x axis. The first row considers graphs that are cascades (left) and spreads (right) and the others are url-based at different minimal cascade lengths. "prune_{i}_size" means that the url-based graphs have been stripped off of their branches that have less than "i" tweets.
- Gathered followers ratio distribution show how well I've been able to gather followers of tweet authors for each dataset.
  The float numbers on the x axis in the top left subplot represent the percentage of followers that I was able to gather in respect to how much I needed to build the graph without loss of followers information. The more the histograms sways to the right, the better.
  The top right and bottom subplots just show histograms of the needed numbers of followers per graph. This should logically be correlated with the sizes of the graphs, but the higher this histograms sways to lower values on the x axis, the more the graphs are star shaped instead of being deep graphs. Also, when very low values on the x axis have a high count in the top right histogram, it means a lot of graphs needed just a low number of followers, and if this is very high compared to other values, it generally means there is a higher chance of having lower percentages of gathered followers in respect to the needed followers which may dampen the importance of the top left histogram.

Figure 21 shows a quick example of how I compared the plots of dataset statistics for the Politifact Dataset and the chosen split for it. I compared the shape of the plots between the train subset and the test subset, but also with the respective plots of the whole dataset, for example, in this case it would be the plot in figure 19. The same comparison process has been repeated for the other statistics, like the sizes subplots (example figure 23) or the followers distributions (example figure 22)




(a) (b)

Figure 21: Tweet counts for the train (a) and test (b) sets of Politifact when split with the C indices.

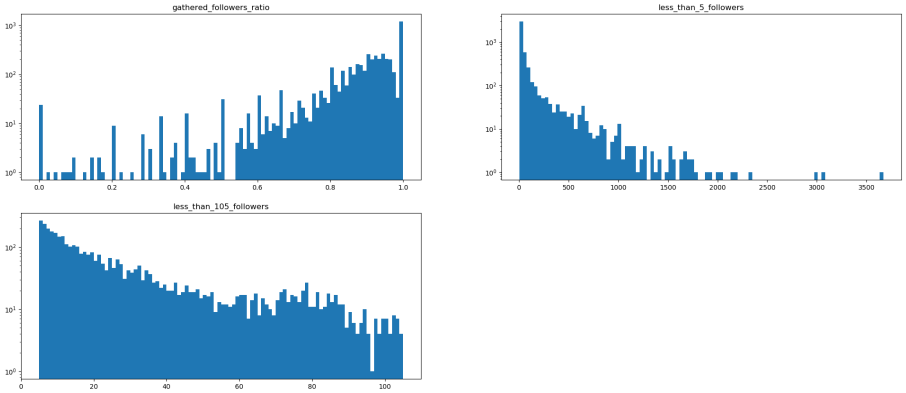



Figure 22: Gathered followers ratio distribution for Gossipcop

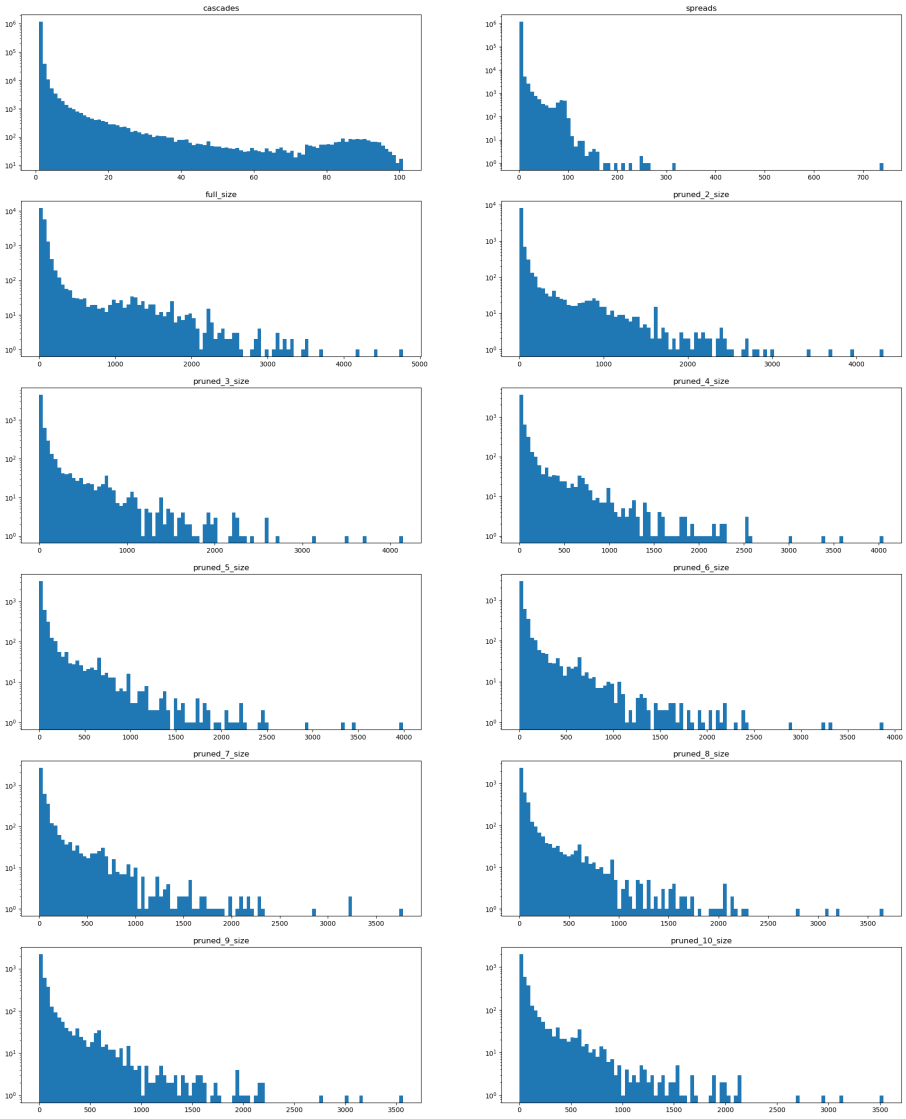



Figure 23: Spreads size distribution for Gossipcop when considering different minimal cascade lengths

# 5 Results

## 5.1 Training Setup

I conducted the training on each of the two train datasets (Gossipcop B and Politifact C) using an 80:20 cross-validation split. For each run, I saved a checkpoint on the best model in terms of validation accuracy and then tested each of the 5 models on the test set, choosing the one with the highest ROC AUC score as the best model. For each experiment, I will display the scores for that best model. For all experiments, I found that the model was most efficiently being trained at a learning rate of $5e^{-3}$ so I started with it but added both a cyclic learning rate scheduler that keeps multiplying it with 0.9 after each epoch until it restarts after 10, thus reaching $1.75e^{-3}$ before going back up. To help the training from getting stuck, I added another scheduler that reduces the learning rate by half if the validation accuracy doesn't improve after a certain number of epochs. And if all that fails to improve the model, another more patient scheduler will completely stop the run when it runs out of patience. The figure 24 shows an example of the tool Visdom plotting the metrics during training, along with the learning rate.

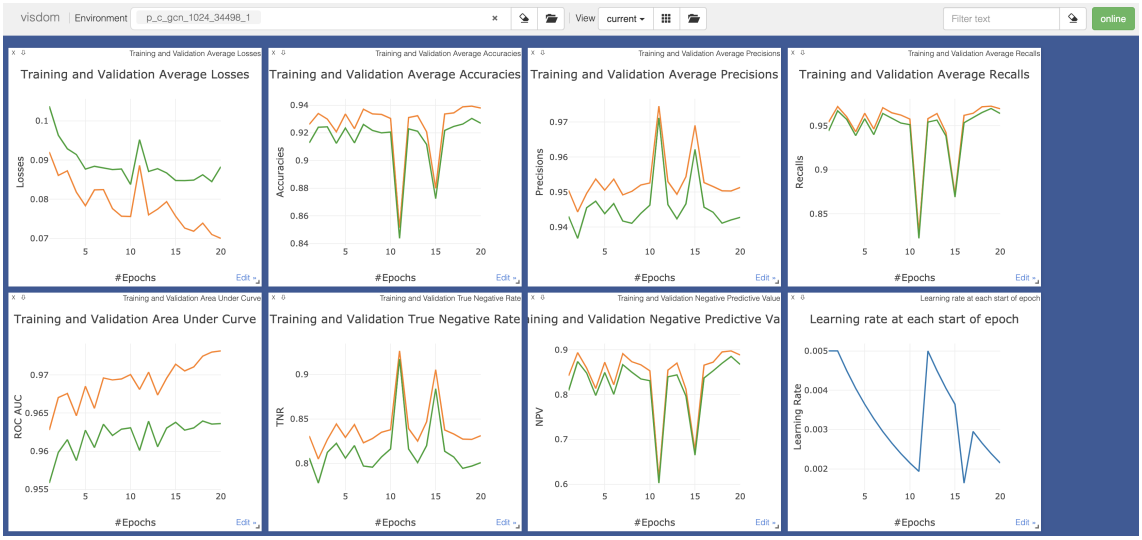



Figure 24: Screenshot of Visdom plotting the metrics of the model during one of the training runs. These metrics are:

- Top row: Binary Cross Entropy loss, accuracy, precision, recall where positive labels are real news and negative are fake
- Bottom row: ROC AUC values, true negative rate (which is the recall but for fake news), negative predicted value (the precision if fake news were positive labels), learning rate

## 5.2 Pure Context Classifier

I started the experiments with the model architecture that doesn't take into consideration the text embeddings of the tweet contents or the author bios, which means it doesn't care about the content of the tweet and only looks at the context. The results are displayed in table 3.

| Dataset | Accuracy | Precision | Recall | ROC AUC | TNR | NPV | Positive Ratio |
|---|---|---|---|---|---|---|---|
| Gossipcop | 0.701 | 0.587 | 0.728 | 0.778 | 0.685 | 0.803 | 0.351 |
| Politifact | 0.785 | 0.931 | 0.810 | 0.788 | 0.635 | 0.356 | 0.773 |

Table 3: Test metrics for the GCN model that doesn't use word embeddings.

The accuracies of these models are barely higher than what a random guessing strategy would have as an accuracy if it just looks at the proportions of the dataset.

### 5.3 Pure Content Classifier

After just looking at the context of tweets, I wanted to compare my results and look at the performance of a the DistilBert classifier operating on the tweets' text contents and authors bios. The results are displayed in table 4

| Dataset | Accuracy | Precision | Recall | ROC AUC | TNR | NPV | Positive Ratio |
|---|---|---|---|---|---|---|---|
| Gossipcop | 0.894 | 0.890 | 0.824 | 0.944 | 0.938 | 0.896 | 0.351 |
| Politifact | 0.877 | 0.947 | 0.907 | 0.884 | 0.694 | 0.551 | 0.773 |

Table 4: Test metrics for the DistilBert model that only uses word embeddings of tweet contents and author bios.

The results are much better than table 3 in every column of the table. However, as I explained in section 3.2, I retrained the DistilBert model and repeated the experiment because I wanted to make sure that the fact that HuggingFace Co. were able to train the model on huge datasets didn't impose a disadvantage on the Graph Convolutional approach. After repeating the experiment with a model trained from scratch, I got the results in table 5.

| Dataset | Accuracy | Precision | Recall | ROC AUC | TNR | NPV | Positive Ratio |
|---|---|---|---|---|---|---|---|
| Gossipcop | 0.827 | 0.782 | 0.758 | 0.861 | 0.870 | 0.854 | 0.351 |
| Politifact | 0.824 | 0.953 | 0.836 | 0.877 | 0.750 | 0.430 | 0.773 |

Table 5: Test metrics for the DistilBert model, trained from scratch on our datasets

As expected, this model is doing worse that the pretrained one, since it has seen way less data in terms of numbers and diversity. But it is still outperforming the pure context graph neural network.

In addition to the DistilBert model, I previously stated in section 3.2 that I would try a different text classification model, that is older and uses a less sophisticated architecture. I chose FastText proposed by Facebook AI Research, modified the module to work with my pipeline and trained it on the same datasets.

| Dataset | Pretrained | Accuracy | Precision | Recall | ROC AUC | TNR | NPV | Positive Ratio |
|---|---|---|---|---|---|---|---|---|
| Gossipcop | Yes | 0.764 | 0.658 | 0.791 | 0.836 | 0.747 | 0.853 | 0.351 |
| | No | 0.754 | 0.667 | 0.710 | 0.821 | 0.781 | 0.814 | |
| Politifact | Yes | 0.766 | 0.954 | 0.764 | 0.841 | 0.779 | 0.352 | 0.773 |
| | No | 0.724 | 0.926 | 0.737 | 0.762 | 0.643 | 0.287 | |

Table 6: Performance of the FastText model when pretrained or trained from scratch on both datasets.

As seen in figure **??**, FastText's results are considerably worse than DistilBert's. And as expected, the pretrained versions are performing better than the ones trained from scratch. We can see, however, than for the Gossipcop dataset, this approach is still performing better than the pure context approach (table 3) but not when applied to the Politifact dataset, hinting that text styles and sentiments of the tweets in the Gossipcop dataset could be more meaningful than in Politifact.

### 5.4 Combining context and content

As suggested in section 3.3, after testing both approaches seperately, I combined both architectures into a Graph Convolutional Network that has more dimensions for its input features, 32 for the user bio and 32 others for the text of the tweet. These text embeddings came by passing the corresponding fields through the best performing text classifier I had so far. The results are listed in table 7

| Dataset | Accuracy | Precision | Recall | ROC AUC | TNR | NPV | Positive Ratio |
|---|---|---|---|---|---|---|---|
| Gossipcop | 0.786 | 0.658 | 0.913 | 0.885 | 0.708 | 0.930 | 0.351 |
| Politifact | 0.840 | 0.940 | 0.869 | 0.846 | 0.664 | 0.456 | 0.773 |

Table 7: Test metrics for the full Graph Convolutional Network model, trained from scratch on our datasets

Overall, these results are way better than just taking the context of the tweets. It's still not as good as the ones from the pretrained DistilBert model but they're definitely on par with the untrained one. Not all the metrics are better here, but at least some are, like the accuracy when performing on Politifact, ROC AUC when performing on Gossipcop. The recall and the negative predictive value are higher on both which means the model is better at grabbing the real news in a bag of news and is more precise when predicting fake ones. However, the results are way better than the ones when using the FastText architecture (table 6) which means the model is able to compete with content-based approaches when computing power or gpu memory size is limiting as both the architecture I'm using and FastText are less expensive to use than Bert.

### 5.5 Varying the Dataset Size

After testing the models on both Politifact and Gossipcop, I wanted to see how their performances scale when reducing the dataset sizes. That's why for each dataset $D$, I randomly sampled three datasets $D_{half}$, $D_{quarter}$ and $D_{eigth}$, dividing the size of the dataset by half every time. I then trained each model architecture on each of the different training sizes, but tested on the full test set. For the model that combines both the content and the context, I used the DistilBert model from its own dataset size to compute the embeddings of the tweets.

| Dataset | Model Type | Accuracy | | | | ROC AUC | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Full | Half | Quarter | Eigth | Full | Half | Quarter | Eigth |
| Gossipcop | Context | 0.701 | 0.690 | 0.687 | 0.660 | 0.778 | 0.773 | 0.766 | 0.743 |
| | Content | 0.894 | 0.891 | 0.854 | 0.851 | 0.944 | 0.925 | 0.919 | 0.909 |
| | Both | 0.786 | 0.725 | 0.698 | 0.689 | 0.885 | 0.795 | 0.802 | 0.786 |
| Politifact | Context | 0.785 | 0.753 | 0.684 | 0.547 | 0.788 | 0.802 | 0.752 | 0.621 |
| | Content | 0.877 | 0.830 | 0.688 | 0.573 | 0.884 | 0.858 | 0.767 | 0.535 |
| | Both | 0.840 | 0.632 | 0.549 | 0.633 | 0.846 | 0.832 | 0.674 | 0.578 |

Table 8: Performance of the three types of models when varying the size of the dataset it's training on.

The repercussion on the model performance due to the changes has different effects on the two datasets. We can see that with the Politifact dataset, as soon as we cut down on the dataset size, the performance of the model becomes worse than randomly guessing the labels, as the concentration of real news in the test dataset is 0.77. The DistilBert model is the only one that resists purging half of the training dataset but cannot go lower than that. In Gossipcop's case, the concentration of real news is 0.351, so any accuracy above 65% is better than random. All models are able to keep up

with it, but deteriorate with successive dataset size reductions. The content-based model is the most resistant however, and my hypothesis for this is that: In the case of Gossipcop, rumors have come from gossipcop.com, while real news came from famous magazines and that could affect the writing style of the tweets sharing them, since most tweets would be quoting content from the articles, so the model might have detected style differences early one with fewer data points. On the other hand, the ROC AUC is mostly improving the bigger the dataset is, so maybe if we had an even bigger dataset, we could reach higher values.

### 5.6 Varying the Gathered Followers Ratio

Since a considerable amount of users in my database miss their list of followers, either because I didn't fetch them due to their large size or because the accounts are private or don't exist anymore, I wanted to know how much this could be affecting my Graph Convolutional Networks' performances. I took the list of usernames that I have followers for, and then randomly took half of it, and then again half of that, and again and one more time. This gave me four folders of followers information, a full one, then half, quarter and eigth. Using these, I built different training sets of cascades, tested on the test set with the full followers infomation, then listed the results in table 9. The only model type I experimented with here is the one that used both context and content.

| Dataset | Followers Size | Accuracy | Precision | Recall | ROC AUC | TNR | NPV |
|---|---|---|---|---|---|---|---|
| Gossipcop | Full | 0.786 | 0.658 | 0.913 | 0.885 | 0.708 | 0.930 |
| | Half | 0.784 | 0.654 | 0.918 | 0.885 | 0.701 | 0.933 |
| | Quarter | 0.783 | 0.663 | 0.874 | 0.872 | 0.726 | 0.903 |
| | Eigth | 0.772 | 0.699 | 0.706 | 0.849 | 0.813 | 0.818 |
| Politifact | Full | 0.840 | 0.940 | 0.869 | 0.846 | 0.664 | 0.456 |
| | Half | 0.818 | 0.945 | 0.836 | 0.853 | 0.707 | 0.415 |
| | Quarter | 0.787 | 0.948 | 0.794 | 0.852 | 0.735 | 0.370 |
| | Eigth | 0.770 | 0.952 | 0.770 | 0.848 | 0.766 | 0.357 |

Table 9: Performance of the three types of models when varying the percentage of users with followers information.

The model is less affected on the Gossipcop dataset than on the Politifact one. I can barely see changes, except when the dataset becomes one eigth the size of the original one. And my hypothesis is similar to the one I had after looking at the results in 8: The differences in real celebrity news vs gossip about them in Gossipcop datasets might mostly be style differences in the articles and the tweet contents quoting them, more than the actual spread of the news over the Twitter network, debates over these news or rumors might not be as frequent or insightful for the model to learn from as the ones about news coming from Politifact, as these would be political news and controversies.

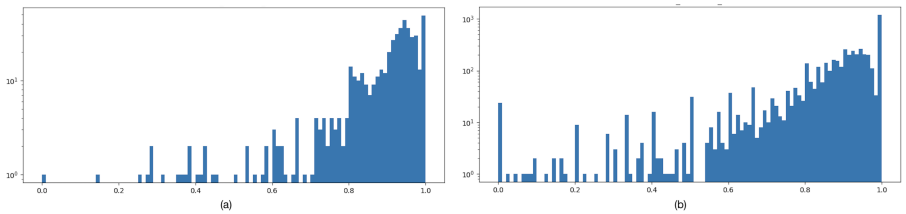



Figure 25: (a) Politifact, (b) Gossipcop. Histograms plotting the number of url graphs per percentage of their nodes having available follower information. We can see that overall, Gossipcop has graphs with more missing information about followers.

Another possibility could be the fact that the Gossipcop dataset originally had more tweets where the author's followers information is missing, so reducing that even more is not affecting the training that

much. Figure 25 shows that Politifact has a bigger concentration of url graphs with complete follower information than Gossipcop. Since with more information missing about user followers, we tend to get shallower graphs with nodes mostly connected to the original tweet, this might suggest that when training on Gossipcop, the model is focusing on other features than the propagation style.

## 5.7 Varying the Model Width

When comparing the Graph Convolutional Network model with the DistilBert one, I realised that even though DistilBert is a tiny version of Bert, it still has around 60 million parameters able to be optimized. Its pre-classifying layer has more than 700 dimensions and training it on one epoch is around 10 times slower than training the GCN model, even though the main code for it is by an established company, so I'm assuming it's more performance optimized than my implementation of a GCN. This is why I set out to see if increasing the dimension of my hidden layers could improve the model. As shown in figure 13, the model has 2 GCN layers and 2 fully connected layers, so I was varying their widths, on both graph neural network models, the one using only context and the one combining content with it. I tested them on the test sets and listed the results in table 10

| Dataset | Type | Width | Accuracy | Precision | Recall | ROC AUC | TNR | NPV |
|---|---|---|---|---|---|---|---|---|
| Gossipcop | Context | (64, 32) | 0.701 | 0.587 | 0.728 | 0.778 | 0.685 | 0.803 |
| | | (256, 64) | 0.686 | 0.561 | 0.808 | 0.781 | 0.611 | 0.838 |
| | | (1024, 128) | 0.693 | 0.574 | 0.786 | 0.786 | 0.641 | 0.830 |
| | Both | (64, 32) | 0.786 | 0.658 | 0.913 | 0.885 | 0.708 | 0.930 |
| | | (256, 64) | 0.783 | 0.655 | 0.912 | 0.884 | 0.703 | 0.929 |
| | | (1024, 128) | 0.787 | 0.660 | 0.911 | 0.884 | 0.711 | 0.928 |
| Politifact | Context | (64, 32) | 0.785 | 0.931 | 0.810 | 0.788 | 0.635 | 0.356 |
| | | (256, 64) | 0.792 | 0.928 | 0.821 | 0.775 | 0.616 | 0.362 |
| | | (1024, 128) | 0.824 | 0.930 | 0.859 | 0.781 | 0.611 | 0.417 |
| | Both | (64, 32) | 0.840 | 0.940 | 0.869 | 0.846 | 0.664 | 0.456 |
| | | (256, 64) | 0.850 | 0.942 | 0.879 | 0.841 | 0.675 | 0.479 |
| | | (1024, 128) | 0.849 | 0.946 | 0.874 | 0.858 | 0.698 | 0.477 |

Table 10: Performance of the three types of models when varying the widths of the models. The width of a model is represented as $(gcn\_hidden\_dimensions, fc\_hidden\_dimensions)$

Increasing the number of hidden dimensions barely improved, or even slightly worsened in some metrics, the performance of the Graph Convolutional Networks on the Gossipcop dataset. While when operating on the Politifact dataset, it mostly helped it, but not by much, and only when the size considerably increased, some slight progress of 5% can be seen in the accuracy, or a bigger one in the negative predictive value and recall. This might suggest that I already reached the best performance of the model and increasing the width further wouldn't help, but the fact that no overfitting is apparent yet could mean it's still possible to try larger or deeper models on a better machine, especially if I have more data.

# 6 Conclusion

In this thesis, I have reproduced the work that Fabula AI achieved and presented in [1]. Inheriting instructions that are quite blurry in that paper, I was able to recreate the data gathering phase and imitate their data pre-processing steps while being innovative to overcome some dataset limitations. Then, I setup an expandable training pipeline to train Graph Convolutional Networks similar to their models. I also trained Bidirectional Transformers and Sentence Embedders to compare the proposed geometric deep learning approach to state of the art methods in text classification.

The results I got in the end still defend the case of using a DistilBert implementation to classify a tweet and its retweets as either fake or real news, even if it means only looking at the content of the tweet or its users bio. The GCNs I implemented were using both the content and the context of these tweets and were doing worse in terms of testing metrics. However, by reducing the contextual information I have about the users represented in the graph, the performance of these models dropped even lower, thus suggesting that if it could be fed more complete contextual data, I could see improvements in their accuracies. The GCNs did perform better than the older text classification approach, FastText, which itself beat other approaches in many benchmarks like LSTMs and ConvNets applied on NLP.

The publicly available version of DistilBert, however, was pretrained for 90 hours on 8 GPUs over the datasets "English Wikipedia" and "Toronto Book Corpus". Even though a DistilBert model is 5 times smaller than the Google Research originally proposed Bert implementation, it weighs 254MB compared to 54KB for my GCN architecture (5MB if at full width). These reasons mean that my implementation is a viable option to choose over Transformers in some cases.

Varying the width of the models hasn't changed the result in many cases, but overall has slightly, very slightly, improved when increasing the number of hidden dimensions. Training the models on small datasets has sometimes made it considerably worse which could suggest that more data would mean better metrics. However, these two are just hypotheses, and having more trainable parameters or more training data might not change anything. Regardless though, fine-tuning DistilBert on the two gathered datasets from FakeNewsNet still provided a successful fake news classifier and that could for the time being, be used by Astroscreen or others to fight Social Media Manipulation.

Nonetheless, hope is not lost for Graph Convolutional Networks in the fight against Social Media Manipulation, as discussed in the previous work section (2), different polynomial parameters could be tested in Spectral Networks approaches, different aggregating and pooling functions could be compared when applying convolutions on the graph, and attention mechanisms could be implemented to better differentiate the relationships between the tweets. The road is still open to augment various Graph Convolutional ideas, or couple the architecture proposed by [1] with non convolutional approaches that have showed good potential.

# 7 Acknowledgements

I am grateful for the guidance of Dr. Mathieu Salzmann whom I would have been lost without at several panic points of the project. I also thank Astroscreen for its compute resources and the help of its employees, especially Daniyal Shahrokhian who set me up on the successful path of this research journey.

# 8 References


[1] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake news detection on social media using geometric deep learning," *arXiv preprint arXiv:1902.06673*, 2019.

[2] S. Vosoughi, D. Roy, and S. Aral, "The spread of true and false news online," *Science*, vol. 359, no. 6380, pp. 1146–1151, 2018.

[3] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[4] P. Ristoski, G. K. D. De Vries, and H. Paulheim, "A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web," in *International Semantic Web Conference*. Springer, 2016, pp. 186–194.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[6] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2019.

[7] K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu, "Fakenewsnet: A data repository with news content, social context and spatialtemporal information for studying fake news on social media," 2018.

[8] L. Wu, F. Morstatter, K. M. Carley, and H. Liu, "Misinformation in social media: Definition, manipulation, and detection," *ACM SIGKDD Explorations Newsletter*, vol. 21, no. 2, pp. 80–90, 2019.

[9] K. Lee, B. D. Eoff, and J. Caverlee, "Seven months with the devils: A long-term study of content polluters on twitter," in *Fifth international AAAI conference on weblogs and social media*, 2011.

[10] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in neural information processing systems*, 2018, pp. 4800–4810.

[11] N. Yu, L. S. Davis, and M. Fritz, "Attributing fake images to gans: Learning and analyzing gan fingerprints," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 7556–7566.

[12] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[14] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[16] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2018.

[17] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[18] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[19] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[20] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[21] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.

[22] S. Webb, J. Caverlee, and C. Pu, "Social honeypots: Making friends with a spammer near you." in *CEAS*, 2008, pp. 1–10.

[23] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia, "Who is tweeting on twitter: human, bot, or cyborg?" in *Proceedings of the 26th annual computer security applications conference*, 2010, pp. 21–30.

[24] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han, "On community outliers and their efficient detection in information networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 813–822.

[25] L. Wu, X. Hu, F. Morstatter, and H. Liu, "Adaptive spammer detection with sparse group modeling," in *Eleventh International AAAI Conference on Web and Social Media*, 2017.

[26] S. Kwon and M. Cha, "Modeling bursty temporal pattern of rumors," in *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[27] A. Friggeri, L. Adamic, D. Eckles, and J. Cheng, "Rumor cascades," in *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[28] L. Wu and H. Liu, "Tracing fake-news footprints: Characterizing social media messages by how they propagate," in *Proceedings of the eleventh ACM international conference on Web Search and Data Mining*, 2018, pp. 637–645.

[29] L. Wu, J. Li, X. Hu, and H. Liu, "Gleaning wisdom from the past: Early detection of emerging rumors in social media," in *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, 2017, pp. 99–107.

[30] Z. Waseem and D. Hovy, "Hateful symbols or hateful people? predictive features for hate speech detection on twitter," in *Proceedings of the NAACL student research workshop*, 2016, pp. 88–93.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[32] K. Nakamura, S. Levy, and W. Y. Wang, "r/fakeddit: A new multimodal benchmark dataset for fine-grained fake news detection," *arXiv preprint arXiv:1911.03854*, 2019.

[33] F. Marra, G. Poggi, C. Sansone, and L. Verdoliva, "Blind prnu-based image clustering for source identification," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 9, pp. 2197–2211, 2017.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[35] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[36] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[38] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," *arXiv preprint arXiv:1808.06670*, 2018.

[39] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[40] B. Weisfeiler and A. A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[41] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *CoRR*, vol. abs/1607.01759, 2016. [Online]. Available: http://arxiv.org/abs/1607.01759