DEPARTMENT OF ENGINEERING MATHEMATICS

A pipeline for Multimodal Approach To Misinformation on Twitter

Armand Kassai Koupai

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

Monday 13th September, 2021

Supervisor: Dr. Ryan McConville

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Armand Kassai Koupai, Monday 13th September, 2021

Contents

List of Figures

List of Tables

List of Listings

Abstract

In the recent years, social networks became one of the biggest sources of news consumption; people get information on social networks and share what they read with their network, which leads to the wide dissemination of fake news, i.e., news with false information, causing negative effects on the society. The detection, evolution and mitigation of fake news quickly became an important area of research – particularly because of recent events such as US 2016 elections and Covid-19.

This project aims to use the FakeNewsNet dataset [36] to build a model that could classify news, namely predict if a news is fake or true. The dataset is composed of diverse features such as the news content (text, images, videos), social context (user profile, tweets, likes, retweets, ...) and spatio-temporal information (users spatial and temporal information). Different approaches have been explored for fake news detection such as uni-modal and multi-modal machine learning, using state-of-art machine learning models such as graph neural networks.

This project deals with all the steps of the data science pipeline: starting with data collection of fake news data using Twitter API, we then pre-process the data using pre-trained models to extract text and image features. Once the features extracted, we build an advanced machine learning model to classify fake/real news present in social media and train it with the processed features and evaluate its accuracy. The purpose of this thesis is to build a pipeline that could outperform the performance of state-of-art models, such as the UPFD [11] framework.

The main achievements of the project are as follows:

- I have collected fake news data, including news information and data related to how news are spread on social media from the beginning of June to mid-August using the FakeNewt repository [36] and the Twitter API.
- I cleaned and pre-processed the dataset, which includes text and image features extraction using BERT [10] and VGG-19 [40] pre-trained models. I represented the pre-processed data as a network, where the root node represent the news and the others nodes can represent tweets, retweets and user profiles. Two types of graph datasets have been proposed: the first is composed of homogeneous graphs and the second is composed of heterogeneous graphs.
- I have succeeded to build a pipeline that outperforms the UPFD framework using the two types of datasets.
- I have built a novel machine learning model making use of heterogeneous graphs instead of homogeneous graphs as done by most of the fact-checking models using graph machine learning. Thus, each graph presents different types of nodes of multiple modalities.
- I wrote a total of 1700 lines of Python code to achieve this.

Supporting Technologies

- I used a Twitter Developer Account provided by the University to collect data using the Twitter API
- I used the *Pandas* [1], *json*, *os*, *re* [2], *Numpy* [3], *NetworkX* [4] open source Python Libraries for data cleaning and pre-processing.
- I used a part of the *transformers* library to extract features from the news content represented as text using BERT [10] and a part of pytorch library to extract features from news content represented as images using VGG-19 [40].
- I built a homogeneous graph neural network model using the Pytorch-geometric library [13], for fake news detection.
- I built a heterogeneous graph neural network model using Deep Graph Library [44], for fake news detection.
- I used a remote server using Secure Shell (SSH) for storage and processing of data. Specifically, I used a machine composed of 2x 1080Tis and 64 GB of RAM.
- I used LaTeX to format my thesis, via the online service *Overleaf*.
- A video [5] presenting my work has been published on the YouTube platform.

[1] https://github.com/pandas-dev/pandas
[2] https://github.com/request/request
[3] https://github.com/numpy/numpy
[4] https://github.com/networkx
[5] https://www.youtube.com/channel/UCCeMOR5DA$_Q$B1R4i14R11Qg/videos

Notation and Acronyms

API	:	Application Programming Interface
UPFD	:	User Preference-aware Fake News Detection
SAFE	:	Similarity-Aware Multi-Modal Fake News Detection
BERT	:	Bidirectional Encoder Representations from Transformers
NLP	:	Natural Language Processing
CV	:	Computer Vision
ANN	:	Artificial Neural Network
CNN	:	Convolutional Neural Network
LSTM	:	Long short-term memory
PYG	:	Pytorch-geometric
DGL	:	Deep Graph Library
t-SNE	:	t-Distributed Stochastic Neighbour Embedding
PCA	:	Principal component analysis
GCN	:	Graph Convolutional Networks
GAT	:	Graph Attention Network
SIFT	:	Scale Invariant Feature Transform
BoW	:	Bag of Words
CBoW	:	Continuous Bag of Words
TF-IDF	:	Term frequency-inverse document frequency
FN	:	False Negative
TN	:	True Negative
FP	:	False Positive
TP	:	True Positive
GD	:	Gradient Descent
SGD	:	Stochastic Gradient Descent

Acknowledgements

I would like to express my gratitude to everyone who helped me in some way or another to write this thesis that lays before you.

I would like to particularly thank my supervisor Dr. Ryan McConville as well as Dr. Dan Saattrup Nielsen for their support and their guidance during our weekly meetings and during the different exchanges we had by email. Your great domain knowledge whether in graph machine learning and multi-modal learning were really valuable for me.

To conclude, I would like to thank the people dearest to me, my family, my girlfriend and my friends, who have always been extremely supportive and believed in me.

Chapter 1

Introduction

1.1 Motivations

With the arrival of the internet, fake news have multiplied, especially on social networks and forums, disseminating false or misleading information to manipulate or deceive the public. Misinformation had a lot of repercussions; for example, some individuals and organisations spread fake news in social media for political gains – a report showed that fake news had an influence on the 2016 US presidential elections [1]. Fake news can also have economical repercussions; in 2007, unfounded allegations were circulating through Vietnamese newspapers, mentioning that eating grapefruit could increase the chances of getting cancer [2] – this fake news had an impact on the grapefruit price. A more recent fake news has been stated by Donald Trump himself, where he was suggesting everyone to inject disinfectant to fight Covid-19 [3]. So, fake news became a critical issue, at a point where demanding approaches to detect fake news have been called for, using artificial intelligence. Until then, the most straightforward approach was fact-checking, which was labour-intensive: the experts could not keep up with the amount of misinformation and the speed at which they spread.

1.2 Existing work

ClaimBuster [18], published in 2017, is the first-ever End-to-end Fact-checking system which leaded to the emergence of new approaches using deep learning models, showing very promising results for fake news detection. For example, the SAFE [50] model used the news article content to predict fake news, using a Text-CNN [21] architecture to extract textual features for each news article; for images, a Text-CNN architecture has also been used on top of a pre-trained image2sentence model. The SAFE model showed that the use of images present in the news article combined with the textual information is more insightful and increases the performance of the fake news detection model. However, this method only focuses on modelling news content and ignores user exogenous context and user endogenous preferences, but often, user preferences are correlated with their online news consumption behaviours. This observation has been explored in the UPFD framework [11], using textual information from the news context, endogenous preference of a user using his/her social network information and the user exogenous context by bringing to light all the users engaged with that news.

1.3 Challenges

1.3.1 Multi-modal information

As explained before, our intention is to use different types of data to predict fake news – building a machine learning model for fake news prediction is already a tough challenge, building a model that takes in consideration different modalities is even harder [45]. The common belief that the more data we have, the more performing our model will be is not always true - an uni-modal model can outperform

[1] https://www.independent.co.uk/life-style/gadgets-and-tech/news/tumblr-russian-hacking-us-presidential-election-fake-news-internet-research-agency-propaganda-bots-a8274321.html

[2] https://www.sggpnews.org.vn/national/vietnamese-grapefruit-not-linked-to-cancer-18391.html

[3] https://www.bbc.com/news/world-us-canada-52407177

a multi-modal model because it is often prone to over-fitting - i.e., building a model that makes very good predictions on the training set but is not able to predict data he has not been trained on - as each modality is generalising at a different rate.

1.3.2 Structure of social media data

Modern deep learning models are designed for Euclidean data, i.e., data that can be represented as simple sequences and grids. However, representing our world is much more complicated because we are interacting with each other and hence, relations are created. Thus, graphs are a general language for describing and analysing entities with relations and interactions – social networks can be represented as a graph data of different modalities, where nodes can represent users, tweets, URLs and edges can represent replies, retweets or likes; building such an interconnected graph is a challenge itself. In addition, there are different types of graphs, the graph can be directed or uni-directed, heterogeneous or homogeneous. Our problem can be represented both as homogeneous graphs and heterogeneous graphs, which will be presented in further details.

1.3.3 Big Data

Another challenge of this project is the size and complexity of the data. As we said before, social networks data is complex because of its structure, however, it is also the representation of our society, i.e., a collection of 7+ billion individuals, bots, etc.. Moreover, we also want to extract features from images and videos which are computationally expensive to process.

1.3.4 Fake News Data

Extracting information related to fake news on social media is not easy and is not static – truth can change over time, and we clearly saw that with Covid-19 when some people were saying that masks were not preventing spread of Covid-19 [4]. Also, Twitter's policy regarding fake news is strict, thus, a lot of tweets related to Fake News can be deleted or even the users can delete their accounts for personal reasons. So, fake news data can be incomplete and can change over time.

1.4 Proposed System

This being said, the project aims to push forward the research of fake news detection and proposes a pipeline that could outperform the results of the UPFD framework, by collecting more data, shaping the graphs in another way than the graphs dataset proposed in the UPFD framework. Also, we are going to create heterogeneous graphs to add image information from the news article. Once the two datasets generated, i.e. a heterogeneous and homogeneous graphs dataset, we build a graph machine learning model that can classify news using graphs as inputs. To compare our performance with other state-of-art models, we use a popular dataset use for many fact-checking applications called FakeNewsNet [36], which consists of labelled news article collected on PolitiFact and GossipCop with context about how these news have been propagated on social media. The model will need pre-processing steps such as features extraction of text and images inputs, using NLP and Computer Vision tools. Then, we build a model using graph neural networks. Indeed, using GNNs is a good choice for social media data because it can be represented as a graph, where nodes represent tweets, users, retweets, and edges represent relations between the nodes. Using graphs instead of traditional data structure can be helpful when we deal with complex domains which have a rich relational structure - it explicitly models the relationships of our data.

1.5 Contributions

- I use a Twitter Developer Account provided by the University to collect the social context data from Twitter, using the Twitter API from the beginning of June to mid-August.
- I implement two end-to-end graph neural network system for fake news detection with two different graphs dataset structure, one involving heterogeneous graphs and the other one homogeneous graphs;

[4] https://health-desk.org/articles/what-do-we-know-about-claims-that-masks-do-not-work

- The study is involving a multi-modal approach, exploiting text and images from news, but also social context, i.e., how the news have been propagated on Twitter (tweets, retweets, user profiles ...).
- I conduct experiments on real-world datasets to demonstrate the performance of our model against state-of-the-art.
- I build a pipeline [5] for fake news detection which outperforms the UPFD framework.

[5] https://github.com/ArmandKassai/FakeNewsPipeline.git

Chapter 2

Technical Background

2.1 Introduction

This chapter tries to provide the background required to understand the technical content of the project and also explains the different existing works related to fake news detection. We will first provide some definitions related to machine learning before diving to some more advanced topics such as Natural Language Processing, Computer Vision and Graph Neural Networks.

2.2 Machine Learning Background

2.2.1 Introduction

Machine Learning is a field of study of artificial intelligence built on mathematical and statistical methods to build computer systems that are able to learn and adapt without following explicit instructions. Even though traditional handcrafted rules were providing good results, adopting a machine learning approach leaded to far better results [9]. In machine learning, we build a model that learns how to make decisions using a dataset. A dataset is used to tune the parameters of a machine learning model. A dataset is composed of features vectors and a target vector - in our case, our features can be news information and social context and the target vector is the labels of the news, i.e., fake or real news.

2.2.2 Classification

The type of machine learning problem we are facing in this project is a classification problem. A classification problem is a problem which aims to assign each input vector to one of a finite number of discrete categories, e.g., fake and real news. Thus, we use a learning algorithm to learn a classifier from the training data set and we use a test data set to see how effective is the model on unseen data - dividing the dataset into a training set and a test set is essential to avoid over-fitting and thus, get a model that is able to generalise well on unseen data. Indeed, if we were using only a training set, we would learn a classifier that would be especially effective on the dataset he has been trained on but we would not know if the model would be good at classifying data that it never faced before.

2.2.3 Evaluation of the performance

Once the parameters of our model are tuned with the training set, we evaluate the results obtained by our model on the test set, using different metrics.

Accuracy

The accuracy is probably the most obvious metric to use, it corresponds to the sum of labels which have been correctly predicted divided by the the number of all our predictions. To be more precise, the labels which have been correctly predicted are either True Positive or True Negative, i.e., respectively labels that are positive (real news) being correctly classified and labels that are negative (fake news) being correctly classified. The mathematical formulation of the accuracy is the following:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision, Recall and Specificity

The problem with the accuracy is that we do not have information of what type of label our model is predicting. For instance, for the purpose of this project, the objective is to distinguish the news which are fake. However, if the dataset is unbalanced, i.e. if we have way more real news than fake news in the dataset, we can have good accuracy because our model predicts all the news which are real news, but he is not able to predict any fake news, which is supposed to be the main objective of our model. Thus, we usually use other metrics with the accuracy such as the precision, which corresponds to the number of True Positives divided by the number of True Positives and False Positives.

$$precision = \frac{TP}{TP + FP}$$

The recall is another metric used to get the number of True Positives divided by the True Positives and False Negatives:

$$recall = \frac{TP}{TP + FN}$$

If our task gives a particular importance of the predictions of negative labels, we can use the specificity metric:

$$specificity = \frac{TN}{TN + FP}$$

F1-Score

To conclude on the different metrics that are often used when we build a classification model, the most efficient metric is probably the F1-score, which tries to give a harmonic mean of the precision and the recall:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

2.3 An introduction to Deep Learning

2.3.1 Introduction

Deep Learning is a sub-field of Machine Learning, which consists of using a set of learning methods to model data with complex architectures combining different non-linear transformations, also called neural networks. Deep Learning has shown great success in the recent years, specifically when it comes to complex applications with unstructured and big data. In particular, the main research topics of Deep Learning today are sound and image processing, speech recognition, computer vision and natural language processing [16].

2.3.2 Neural Networks

To understand deeper how Deep Learning is working, some details on neural networks should be presented. As we said, we use an artificial neural network to model the data. In fact, an artificial neural network is an end-to-end non linear application, meaning that given an entry x, it returns an output $y = f(x, \theta)$, where θ are some parameters that are tuned during the learning phase, as we explained in the section 2.2.1. An artificial neural network is made up of vectors of neurons called layers. The **Deep** of Deep Learning comes from the fact that we assemble multiple layers together to get a Deep Neural Network. The most basic ANN is the artificial neuron, which means that our network is composed of only one neuron.

An artificial neuron is simply a linear transformation of an input vector x:

$$y = w_b + \sum_{i=1}^{d} x_i w_i$$

where $w \in \mathbb{R}^{d+1}$ is the vector of weights w_i and bias w_b that must be learned during the learning phase. The bias helps the learning of these values by encoding a prior belief about the value the neuron should produce. However, real world problems cannot be resolved using only linear transformations, thus, we

usually add on top of that linear equation a function called an activate function σ to give a non-linear behaviour to the function:

$$y = \sigma(w_b + \sum_{i=1}^{d} x_i w_i)$$

A commonly used activate function is the rectified linear unit (ReLU):

$$\sigma(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

The structure of the artificial neuron can be seen as following:

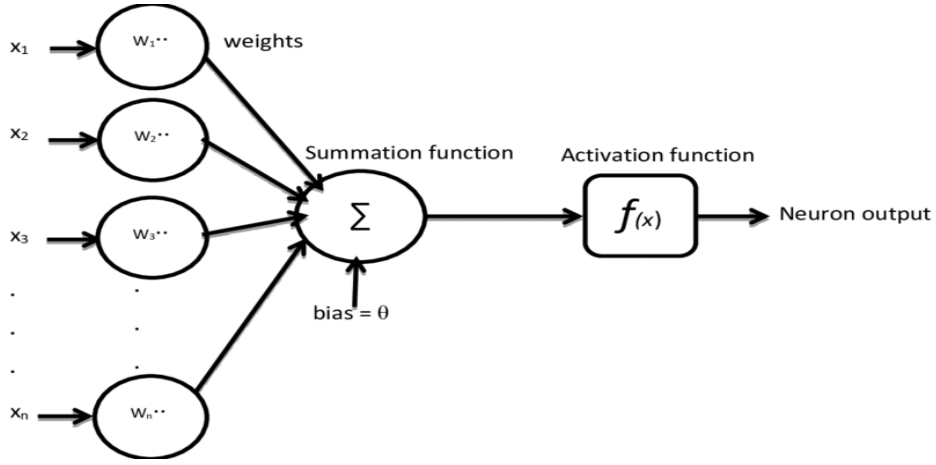

Figure 2.1: Structure of the artificial neuron [49]

Thus, to build powerful systems that can get good predictions, we have multiple neurons in a layer and we stack multiple layers together to get a complex model:

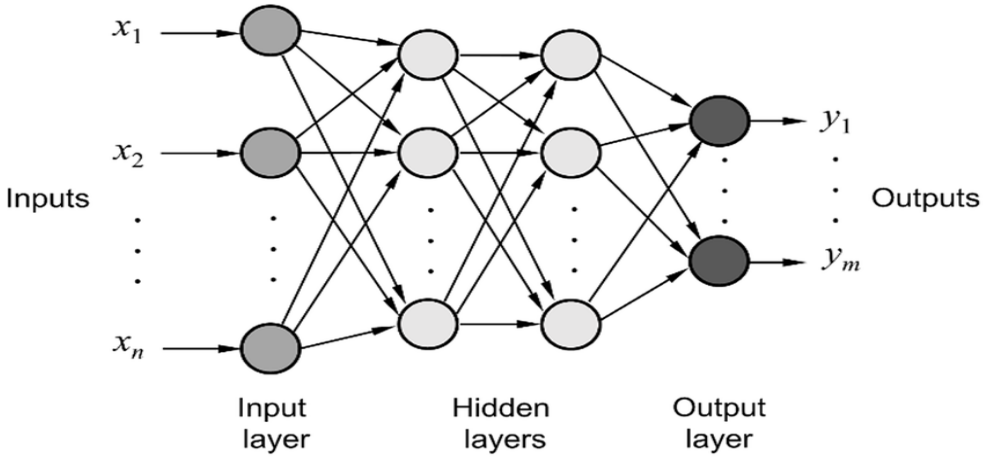

Figure 2.2: Structure of a multi-layer neural network [32]

A multi-layer neural network is composed of three different types of layers: an input layer, a hidden layer and an output layer [16]:

- The input layer represents the features which are fed into the network. Features can be of different types such as discrete or continuous. Each neuron in the input layer represents a distinct feature fed to the neural network.
- A hidden layer transforms the inputs into the desired outputs by applying non-linear transformations as explained for the artificial neuron. The number of neurons in the hidden layer (hidden units) determine the complexity of functions a network can model - the greater number of hidden units

means the more complex function a network can model. Each output of a hidden layer n is the input of the hidden layer $n+1$. Calculating all the outputs layers based on the input features is called the forward pass.

- Finally, the output layer is the layer that returns the predictions. The output layer's architecture depends on the type of task we are trying to solve; for a binary classification task, i.e. a task when we try to predict a 0 or a 1, we can use the sigmoid activation function. It returns the probability $\mathbb{P}(Y|X)$ that the class is either 0 or 1 given the input features.

When each neuron in one layer is connected to every other neurons in an adjacent layer like in the figure 2.2, we say that the layers are fully connected.

2.3.3 Loss function

Until now, we explained how we build a deep neural network and how we produce an output y using a multi-layer neural network. However, to produce good results, we need to tune the weights of each neuron of the network correctly. To do so, we use a loss function that compares the predictions of the network against the real labels of our dataset. The loss function quantifies how good the predictions are. One popular loss function for classification is the cross-entropy, which measures the difference between a target distribution (p) and an estimated distribution (q):

$$H(p,q) = -\sum_{x} p(x) log(q(x))$$

In our case, q is the estimated distribution of our network predictions built by our output layer and p corresponds to the target distribution of our ground truth. The network produces some predictions y^* and we compare the values obtained with the truth labels using a loss function L to find an optimal parameter set W^*:

$$W^* = \arg\min_{W} \sum_{n=1}^{N} L(f(x_n; W), g) = \arg\min_{W} J(w)$$

where W is the aggregation of several multi-dimensional matrices. To minimise the loss function, we apply an optimisation technique called the Gradient Descent, which computes the gradient of the loss with respect to the current W.

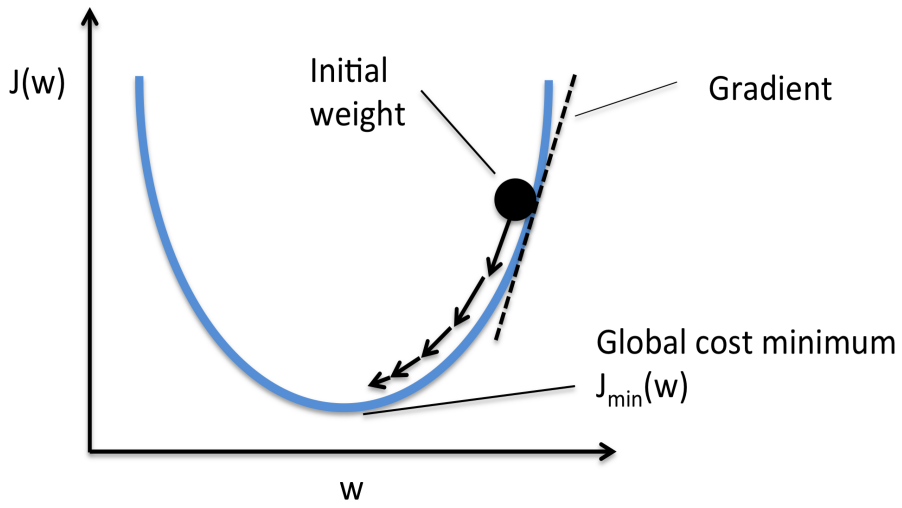

Figure 2.3: An explanation of the Gradient Descent Algorithm [2]

Looking to the Fig. 2.3, we start at a random initial weight and we move towards the direction which minimises the loss by computing the gradient and stepping in the opposite direction of the gradient. The size of the step is defined by the learning rate η. At each step, we update the weights as follow:

$$W^{\text{new}} = W^{\text{current}} - \eta \frac{\partial J}{\partial W}$$

After obtaining our new W, which is the aggregation of several multi-dimensional matrices $W^{(1)}, ..., W^{(n)}$, we use the chain rule recursively to back-propagate our gradient calculation layer by layer from right to left to update each weight W_i. This step is called the Back Propagation. Once the back propagation step done, we start again the forward pass step. The goal is to repeat this process until the convergence of the loss function.

2.3.4 Transfer Learning

Transfer Learning is a sub-field of machine learning and more precisely deep learning which aims to apply the knowledge gained from one task for your own application. This is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.

Thus, to achieve transfer learning, we first need to use a pre-trained model created by someone else, often tech giant or group of researchers. The pre-trained model is often trained on a very large dataset to solve a particular problem such as image classification and word embeddings. Once the model is chosen and answers a particular task, we often have to fine-tune the parameters of our pre-trained model to achieve higher accuracy and generate the output in the correct format. To do so, we often drop the top layers and add layers that are suited to our problem. A particular example is the AlexNet [24] pre-trained model, which can classify around 1000 different objects; if our task is to classify fake and real news, we should change the output layer to have a layer with only two neurons.

We introduce Transfer Learning in this project because the use of pre-trained models often achieve state-of-art performance and hence, is a great option for many of our down-stream tasks such as image embeddings and text embeddings, that are going to be introduced in many details in the following sections.

2.4 Introduction to Computer Vision

Computer Vision is a scientific field where we build a computer system that can understand digital images or videos. In other words, we are helping computers to see. It is a subfield of artificial intelligence and machine learning. Computer Vision can be used in many applications such as medical imaging, motion capture, 3D model building [35]. For this project, we are using Computer Vision methods to get image embeddings - this is an important down-stream task for many applications involving images as input.

2.4.1 Image embeddings

When it comes to data related problems, for most practical applications, the original input variables are dirty and need some pre-processing; we typically transform the inputs into a new space of variables which is more informative and thus increases the chance to solve our classification problem. This pre-processing stage is called feature extraction, which involves removing uncorrelated features or transforming them to increase the accuracy of the learning algorithm and reduce the training time.

In the case of images pre-processing, we want to extract features from images, i.e., builds a vector that represents the information given by the image - an image embeddings vector is a vector of numbers which represent images. Getting image embeddings is essential so that we can extract information from images. To do so, we first represent images as grid of colours where each pixel consists of one value of the grid. There are exactly 3 grid of colours, one representing the red colour, one representing the green and the last one the blue colour. By combining these three grids, we can represent colour pictures using numerical values.

There are different techniques used to extract image embeddings. There are some traditional techniques such as the SIFT [5] descriptor. SIFT is a descriptor that extracts features from the images using computer vision. The SIFT algorithm works well on images that represent the same item. However, in our case, we want to extract features from images that are different. We can also build image embeddings using neural networks. The following parts will introduce the CNNs, a type of neural network layer that is used for images and a widely used pre-trained model called VGG-19 [40], that is used for different tasks such as visual embeddings and image classification.

2.4.2 CNNs

Convolutional neural networks, also known as CNNs, are a specific type of neural networks used for image processing. The traditional architecture of a CNN is composed of a convolutional layer, a pooling layer and a fully connected layer [4]. The architecture is described in the Fig. 2.4:

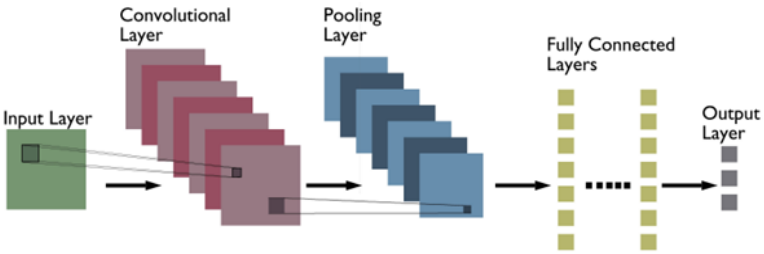

Figure 2.4: Traditional CNN Architecture [4]

We are going to explore the different layers of the traditional CNN architecture. We pass the input images represented as grids in a convolutional layer, which uses filters to perform convolutional operations to scan the input layer with respect to its dimensions, as described in the Fig. 2.5:

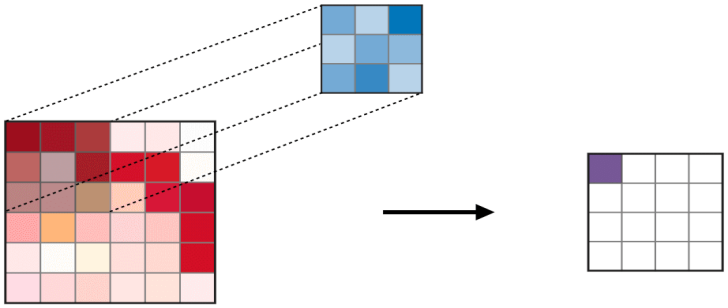

Figure 2.5: Convolutional Operation [1]

The outputs are then put in a pooling layer - it is a down-sampling operation applied after the convolutional layer, doing some spatial invariance. There are different types of pooling layers: a max pooling layer selects the maximum value of the current view while an average pooling layer averages the values of the current view. The Fig. 2.6 describes a max pooling operation:

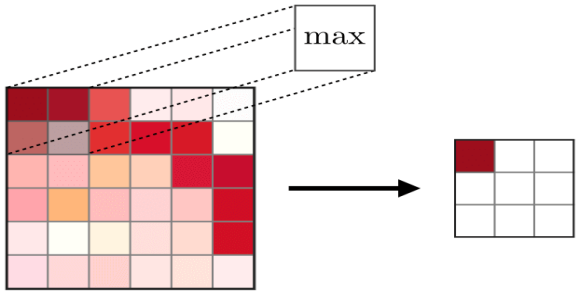

Figure 2.6: Max pooling operation [1]

Finally, the fully connected layer operates on a flattened input where each input is connected to all neurons.

Convolutional neural network layers are very popular in the literature and are widely used to solve computer vision problems. In our case, we introduced convolutional neural networks layers to introduce VGG-19, a pre-trained model that can extract image embeddings using CNNs layers.

2.4.3 VGG-19

As introduced in the section 2.3.4, pre-trained models are extremely popular as they achieve good performance on a wide variety of tasks. There are many popular pre-trained models used in computer vision, such as AlexNet [24] and VGG [40]. In this project, we use the VGG-19 pre-trained model, a variant of VGG which is composed of 19 layers:

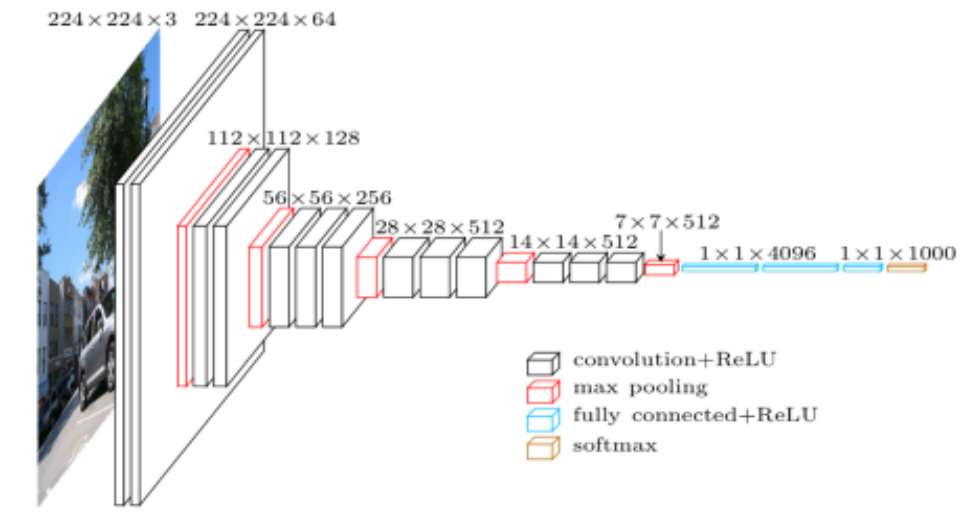

Figure 2.7: Architecture of VGG [40]

VGG is a convolutional neural network model for image recognition proposed by the Visual Geometry Group in the University of Oxford [40]. To extract features from the VGG-19, we remove the layers after the fully connected layer which outputs a vector of size 4096.

2.5 Natural Language Processing for text features extraction

2.5.1 Introduction

Natural language processing is the intersection of computer science, linguistics and machine learning. Concretely, NLP is the field of research that aims to build computers systems that can understand and generate human language. Applications of NLP techniques include voice assistants like Amazon's Alexa and Apple's Siri [39], but also use cases like machine translation and text-filtering. In our case, we are interested in text feature extraction, which is a down-stream task of many NLP pipelines. Text feature extraction is the process of extracting useful information from text document. Traditional methods used for text feature extraction are filtration, fusion, mapping, or clustering techniques [30]. Text feature extraction is an important step of the project, as our dataset is mainly composed of text information. We will first give a broad explanation of text pre-processing techniques [20] and then, we will review some of the state-of-art methods to extract features from text used in many fact-checking models.

2.5.2 Review of text features extraction techniques

One-hot-encoding

One-hot-encoding is a traditional and simple technique used to represent a word in a form that can be understood by the computer. If we assume we have a text document of vocabulary size V, when we encode our words using one-hot-encoding, we create for each word a vector of length V, where all values in the vector are zero except the value corresponding to the index of the word in V. One big issue of one-hot-encoding is that this sparse representation does not allow us to compare words and thus, the computer system does not understand much information about the words.

Bag of words

A BoW is a way of extracting features from text. A bag-of-words is a representation of text that describes the occurrence of words in a document. A bag-of-word does not care about ordering and the structure of the words, but only checks how many times a word occurs in the document. The intuition of bag-of-words is that documents with similar words have similar content. However, like one-hot-encoding, extracting features using bag-of word could be improved by comparing words.

Term frequency-inverse document frequency

The inverse document frequency is a measure of the importance of the term in an entire corpus. If we use TF-IDF, we try to give more weight to the less frequent terms. Indeed, very frequent words like *the* and *it* carry little information. TF-IDF emphasises words that occur in fewer documents by incorporating inverse document frequency (IDF):

$$idf(t, D) = log(\frac{|D|}{\text{number of documents in D that t occurs in}})$$

where t is a word and D is corpus with documents inside it. Also, we calculate the term frequency (TF):

$$tf = log(count(t, d) + 1)$$

where d is a document of D. Finally, we compute the TF-IDF:

$$tf \cdot idf(t, D) = tf(t, d) \cdot idf(t, D)$$

Word2Vec

Word2vec are models developed by the Google Team, which extract features from text using artificial neural networks. Two different architectures have been proposed:

- the continuous bag of words (CBoW)
- the skip gram model

The CBoW model tries to predict a word based on the words which are closed to it whereas the skip gram model does the opposite, i.e., predict the words in the context of the sentence based on the input word. The CBoW model usually learns faster than the skip-gram model but is less efficient. In both models, the architecture involves 2 layers and can be used to extract features from text. The architecture of the model is described in the Fig. 2.8:

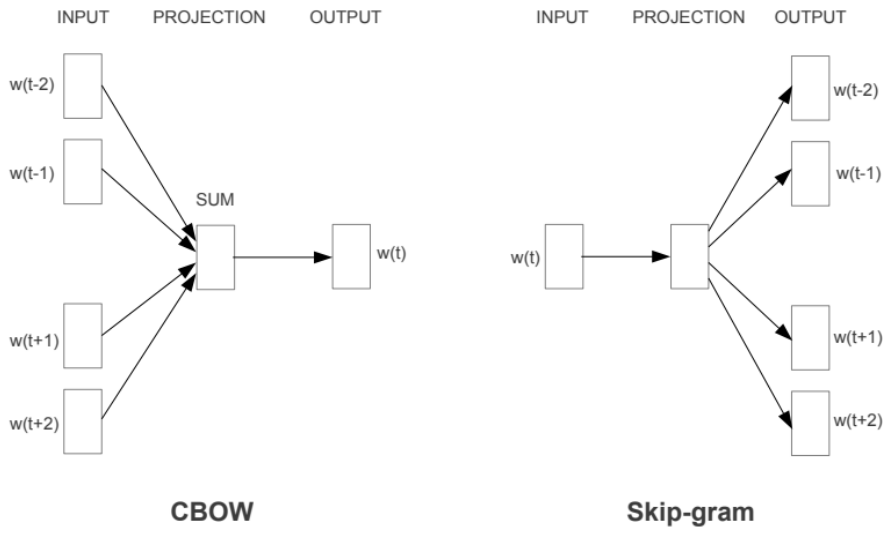

Figure 2.8: Architecture of the CBoW and Skip-gram models [29]

There is also an output layer that can be used for classification tasks, using a softmax activation function.

However, the model does not need any labels as the ground truth is being directly deduced from the data and more particularly from the proximity of the words within the training corpus. We often call these type of learnings **self-supervised learning**.

Now that we reviewed the main techniques used to extract features from text, we are going to study in details some advanced pre-processing techniques used to extract features from text that have been designed in state-of-art fact-checking models.

2.5.3 Text-CNN

In the SAFE model [50], a model used for fake news detection, the Text-CNN [21] architecture has been used for text feature extraction. If CNNs models have been originally imagined for image feature extraction, they have shown very good results for some NLP tasks such as semantic parsing, sentence modelling or other traditional tasks [21]. The architecture of the Text-CNN is provided in the Fig. 2.9, which contains as explained above, a convolutional layer and a max pooling layer.

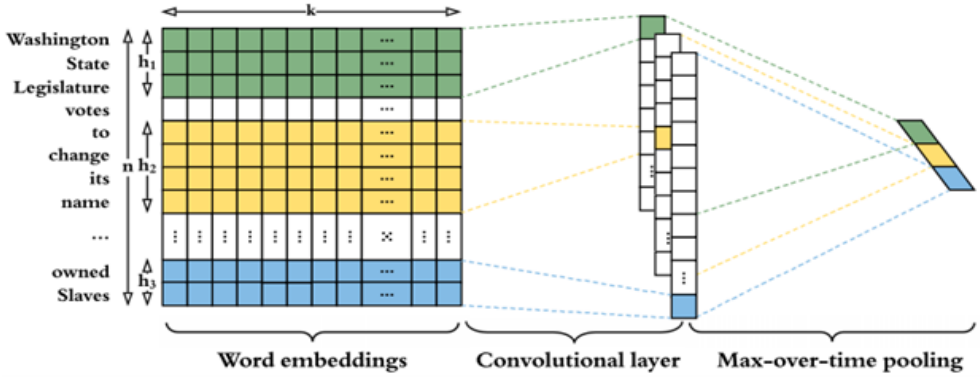

Figure 2.9: Text-CNN Architecture [21]

If we consider one document D, containing n words, where each word is embedded as $x_t^l \in \mathbb{R}^k$, $l = 1, 2, ..., n$. We use a convolutional layer to produce a feature map denoted as $C_t = \{c_t^i\}_{i=1}^{n-h+1}$ from a sequence of local inputs $\{x_t^{i:(i+h-1)}\}_{i=1}^{n-h+1}$, via a filter w_t. So, each local input can be considered as a group of h continuous words as explained in the paper [21]. We can also write is as:

$$c_t^i = \sigma(w_t \cdot x_t^{i:(i+h-1)} + b_t)$$

$$x_{i:i(i+h-1)} = x_i \oplus x_{i+1} \oplus ... \oplus x_{i+h-1}$$

where w_t, $x_t^{i:(i+h-1)} \in \mathbb{R}^{h*k}$, $b_t \in \mathbb{R}$ is a bias, $\oplus$ is the concatenation operator and σ is the ReLU activation function. The max-pooling layer is applied on the feature map obtained by the CNN layer for dimension reduction: $\hat{c}_t = \max\{c_t^i\}_{i=1}^{n-h+1}$

We obtain the representation of the news text using a fully connected layer:

$$t = W_t \hat{c}_t + b_t$$

where $\hat{c}_t \in \mathbb{R}^g$, g is the window size chosen, $W_t \in \mathbb{R}^{d*g}$ and $b_t \in \mathbb{R}^d$ are the parameters to learn.

2.5.4 Pre-trained models using Transformer

Attention Layers

Before we dive in the popular pre-trained models for natural language processing, it is important to introduce one layer architecture that allowed the arrival of state-of-art pre-trained models.

For a variety of NLP tasks, LSTM layers were considered as the state-of-the-art architecture for sequential text processing, as it stores context information along a sequence of words to help make the

decision for each token. However, there are some drawbacks with the LSTMs such as scalability and long-range dependencies. Therefore, a new type of architecture has been proposed by google researchers called an attention layer, which takes relevant context information from anywhere in a sequence to process the current token:

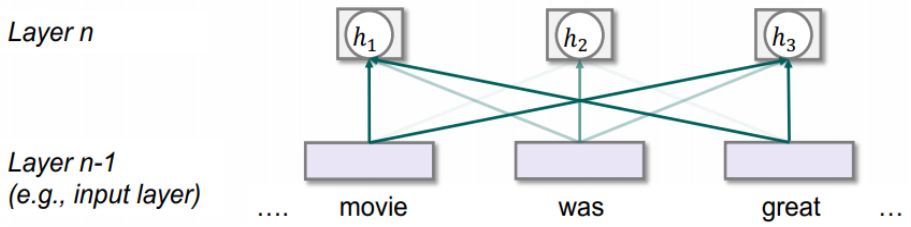

Figure 2.10: Illustration of a self-attention layer

Attention tries to weight the connections between a sequence of values in a layer $n-1$ and a sequence of nodes in a layer n. The strengths of the weights are determined by comparing the values at each position in layer $n-1$ with each other. To do so, we use the dot product to compare the vectors of two nodes in the layer $n-1$:

$$score(x_i, x_j) = W^q x_i \cdot W^k x_j$$

where W^q and W^k are matrices of weights that can be learned using Back Propagation. We scale and apply the softmax function to turn the score into attention weights that sum to 1:

$$\alpha(i,j) = softmax(\frac{W^q x_i \cdot W^k x_j}{\sqrt{d}})$$

$\alpha(i,j)$ indicates the proportion of attention we put on x_j in layer $n-1$ when determining the hidden state h_i in layer n for token i and is the number of dimensions of the embeddings in layer n-1. To compute the hidden state h_i for layer n, we calculate the weighted sum over the input sequence:

$$h_i = \sum_{j=1}^{N} \alpha(i,j) W^\nu x_j$$

where W^ν is a matrix of learnable weights.

The major advantages of a self attention layer over a LSTM layer are that it allows the model to select relevant parts of the context when processing each token and each hidden state h_i is computed independently, so parallel computing is possible. However, the self-attention mechanism does not deal with word ordering, which is an important information to understand a sentence. To provide this information, we add positional embeddings to the input embeddings, i.e., vectors that encode the position of each token in the sequence.

BERT model

Bidirectional Encoder Representations from Transformers (BERT) [10] is a pre-trained model developped by researchers at Google which provides state-of-art results in many NLP tasks such as Question Answering or featured-base training, i.e., producing word embeddings which are used for other NLP tasks, which is the application we are interested in. BERT makes use of Transformer [43], a deep learning architecture that stacks multiple self-attention layers along with fully-connected layers. Transformer is an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task:

- The **Encoder** is composed of 6 layers, each of them composed of 2 sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. Some normalisation layers are also used in each sub-layers.
- The **decoder** is also composed of a stack of N = 6 identical layers. However, the decoder is composed of three sub-layers - the third layer performs multi-head attention over the output of the encoder stack.

We can summarise the architecture of the BERT model in the Fig.2.11:

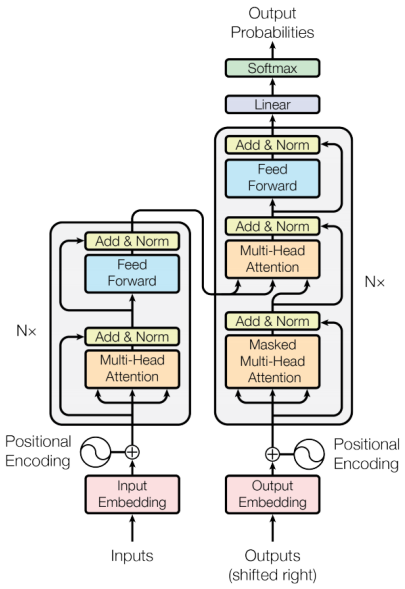

Figure 2.11: BERT Transformer Architecture

Since we use BERT as a down-stream task to generate sentence embeddings from our text news articles, tweets, retweets and user profiles description, only the encoder is needed for our application. The purpose of this down-stream task is to produce text features inputs for our machine learning model.

2.6 Social networks analysis

2.6.1 Introduction

SNA is the process of investigating social structures in terms of complex networks, thanks to graph theory. Social networks can be seen as a complex network composed of nodes and edges, where nodes can characterise individuals or objects and edges express the relations between these. We can observe in the Fig. 2.12 how we can represent a US presidential election network [41]:

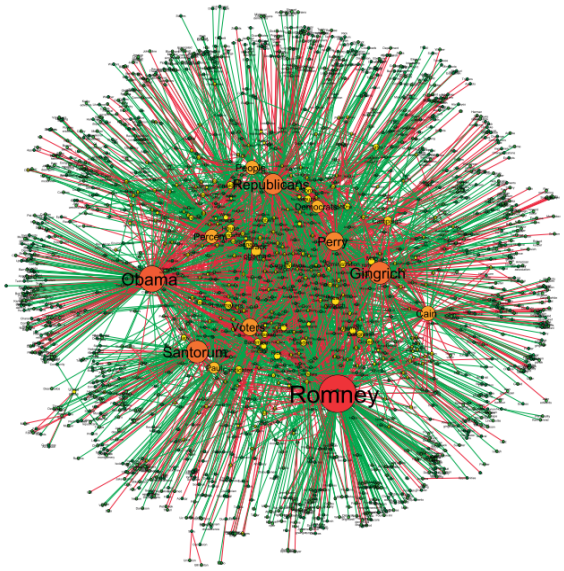

Figure 2.12: US presidential election network data [41]

In this example, edges do not have direction, but there are some situations where we have to precise the direction of the node, e.g. if I am calling my friend, I am the person who calls him and he is the one who receives the call, so the relation is unidirectional. An edge can also have a weight, which describes the number of times that an edge appears between two nodes. If I am calling 3 times my friend, the edge will have a weight equal to 3. One important notion when creating a graph is the centrality, which quantify how influential a node is in the network. There are different ways to measure the centrality of the node such as the degree, the closeness and the betweenness [47]:

- A node's degree is the number of edges linked to a node.
- Closeness measures how good is a node connected to all the other nodes in the network; it is the average number of hops needed to reach the other nodes of the network. A hop is defined as the path of an edge from one node to another.
- Betweenness measures the importance of a node's connections in allowing nodes to reach other nodes. It represents the number of shortest paths the node is included in divided by the total number of shortest paths.

SNA can be applied in many applications such as sociology, psychology, health, biology and even economics. In our case, we are going to study how social networks can propagate misinformation and thus can be used to detect fake news.

2.7 Misinformation in social media

Misinformation is false or inaccurate information that is often intended to deceive. When the information is deliberately deceptive, it is disinformation, a subset of misinformation. Misinformation is often used to bring fear to people, make a buzz or manipulate people. Social media largely contributes to the spread of misinformation, because users often share information and discuss about it without checking the legitimacy of the information they read. Some studies showed that fake news were spreading way more faster and deeper than accurate information on Twitter [3]. Fighting the spread of misinformation on social media is difficult for many reasons:

- The diversity of information sources makes it hard for the reader to verify the reliability of the information.
- One of the biggest strengths and the most dangerous aspects of social media is that people tend to follow and support information from link-minded individuals and influencers which lead to the formation of echo chambers and filter bubbles.

2.7.1 FakeNewsNet dataset

Popular datasets

There are different existing datasets used for fake news detection. We will give a quick description of some of them:

- LIAR [46] is a publicly available dataset for fake news detection. A decade-long of 12.8K manually labelled short statements were collected in various contexts from poltificat, which provides detailed analysis report and links to source documents for each case.
- With the COVID-19 [31] pandemic, fake news and rumours are frequent on social media. To tackle this, a annotated dataset of 10,700 social media posts and articles of real and fake news on COVID-19 has been created to build machine learning models for fake news detection related to COVID-19.
- FakeNewsNet [36] is a fake news dataset collected from two fact-checking websites: GossipCop and PolitiFact containing news contents with labels annotated by professional journalists and experts, along with social context information.

Structure of Fake News Net dataset

The dataset used in this project is the FakeNewsNet dataset. The FakeNewsNet dataset is composed of different types of information and thus, is complex and of different modalities. First, it provides the news content which contains information such as the title, the article, the published date, the url of the article, the url of images and videos related to the news. We could build a fake news model only based on the information provided by the news article, but the FakeNewsNet dataset also provides information on how the news have been propagated on the social media. Thus, we can collect the tweets from the people who talked about the news, the retweets and the user profiles. We can also collect the user time-lime tweets, i.e., the 100 last tweets from a user that tweeted something on the fake news, the users followers and the users followings. An example of the information provided by one tweet is described in Appendix A.

The statistics of the dataset is described in the following table:

Table 2.1: Statistics of FakeNewsNet dataset [36]

	Category	Features	PolitiFact		GossipCop	
			Fake	Real	Fake	Real
News Content	*Linguistic*	# News articles	432	624	5,323	16,817
		# News articles with text	420	528	4,947	16,694
	Visual	# News articles with images	336	447	1,650	16,767
Social Context	*User*	# Users posting tweets	95,553	249,887	265,155	80,137
		# Users involved in likes	113,473	401,363	348,852	145,078
		# Users involved in retweets	106,195	346,459	239,483	118,894
		# Users involved in replies	40,585	18,6675	106,325	50,799
	Post	# Tweets posting news	164,892	399,237	519,581	876,967
	Response	# Tweets with replies	11,975	41,852	39,717	11,912
		# Tweets with likes	31692	93,839	96,906	41,889
		# Tweets with retweets	23,489	67,035	56,552	24,955
	Network	# Followers	405,509,460	1,012,218,640	630,231,413	293,001,487
		# Followees	449,463,557	1,071,492,603	619,207,586	308,428,225
		Average # followers	1299.98	982.67	1020.99	933.64
		Average # followees	1440.89	1040.21	1003.14	982.80
Spatiotemporal Information	*Spatial*	# User profiles with locations	217,379	719,331	429,547	220,264
		# Tweets with locations	3,337	12,692	12,286	2,451
	Temporal	# Timestamps for news pieces	296	167	3,558	9,119
		# Timestamps for response	171,301	669,641	381,600	200,531

2.8 Machine Learning with graphs

2.8.1 Graphs data structure

A graph is a data structure consisting of two objects: nodes and edges. A graph is a general structure for describing data that can be modelled by relations or interactions. The world we are living in can be modelled as a graph where we are represented as nodes and edges represent the relations we have with others. We can represent a lot of different type of data as graphs, such as molecules [27], underground networks, networks of neurons, citation networks or social networks. In our case, that is the last application we are interested in, as we have some information about how the fake news are propagating on Twitter; thus, we could represent our data as a dataset composed of graphs. However, traditional machine learning and deep learning applications are designed for simple sequence and grids data structure. If we want to represent our data as different graphs, we need to use tools that can learn from networks, which are extremely complex as there is no fixed ordering between nodes and the nodes have often multi-modal features, like our problem.

2.8.2 Application of GNNs

Graph Neural Networks are a type of Neural Network which directly operates on the graph structure. GNNs can be used for both supervised and unsupervised learning and solve both regression and classification problems. A popular application is graph classification, where we try to predict the label at a graph level. In this case, we have an inductive structure, i.e. a dataset composed of multiple graphs where we try to build a model that can predict the labels of the graphs. We can also build a model for

node classification, where we try to predict the labels of some nodes in a graph. In this approach, we can either have a dataset composed of multiple graphs or one big graph. For the project, we used an inductive structure and we are solving a graph classification problem.

2.8.3 Introduction to GNNs

If we assume we have a graph G, the idea behind graph neural networks is to transform the information at the neighbours of each graph and combine it together. Thus, nodes aggregate information from their neighbours and are the inputs of a graph convolutional network [23]:

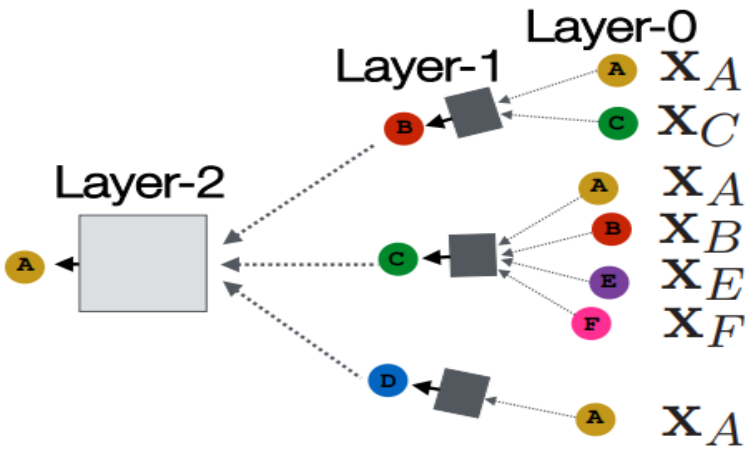

Figure 2.13: Neighbourhood aggregation [25]

In the figure 2.13, the node A is next to three other nodes B, C, D which are themselves next to other nodes. Thus, at the layer 0, we only have the nodes and their respective node feature. The black boxes represent aggregation functions. The mathematical formulation of the hidden representation of the node ν at the layer $l+1$ can be written:

$$h_\nu^{(l+1)} = \sigma(W_l \sum_{u \in N(\nu)} \frac{h_u^{(l)}}{|N(\nu)|} + B_l h_\nu^{(l)})$$

where $l \in \{0, ..., L-1\}$. W_l and B_l are the trainable weight matrices, σ is the activate function, $h_\nu^{(l)}$ is the embedding of the node ν at layer l and $|N(\nu)|$ is the average of neighbour's previous layer embeddings. We can feed these embeddings into any loss function and run an optimisation technique such as the SGD to train the weight parameters. For a classification task, we can use the cross-entropy loss:

$$L = \sum_{\nu \in V} y_\nu log(\sigma(z_\nu^T \theta)) + (1 - y_\nu) log(1 - \sigma(z_\nu^T \theta))$$

where θ is the vector of weights, y_ν is the node class labels and $z_\nu = h_\nu^{(l)}$ is the node embedding.

2.8.4 GraphSAGE

Until now, we have aggregated the neighbour messages by taking the weighted average. However, a better approach has been proposed by researchers from Stanford University, called GraphSAGE [17], which propose different neighbour aggregations. Thus, we can rewrite the hidden representation of the node ν at the layer $l+1$:

$$h_\nu^{(l+1)} = \sigma([W_l \cdot AGG(\{h_u^{(l)}, \forall u \in N(\nu)\}), B_l h_\nu^{(l)}])$$

The different variants for the neighbour aggregations are as follows:

- Mean: Take a weighted average of neighbours

$$AGG = \frac{h_u^{(l)}}{|N(\nu)|}$$

- Pool: Transform neighbour vectors and apply symmetric vector function

$$AGG = \gamma(\{MLP(h_u^{(l)}), \forall u \in N(\nu)\}) \text{ where } \gamma \text{ is the element-wise mean/max}$$

- LSTM: Apply LSTM to reshuffled of neighbours

$$AGG = LSTM([h_u^{(l)}, \forall u \in \pi(N(\nu))])$$

Chapter 3

Project Execution

3.1 Introduction

This chapter presents the different steps needed to build a system that can predict fake news. The whole architecture of the project is first described and then all steps are analysed in details. The project aims to propose two new structure of graphs, one involving homogeneous graphs and one involving heterogeneous graphs. The type of data used also differs from the one used by the UPFD framework. For the heterogeneous graphs, we also add images features to the news nodes and check how it impacts the performance of the model.

3.1.1 Problem formulation

As we explained before, the dataset we are using for this project is the FakeNewsNet dataset, composed of news content and social context, which is going to be represented as graphs. For our task, we can define the problem as follows: given a training data $D = (G_1, y_1), \ldots, (G_N, y_N)$, where $G_i \in G$ is a graph describing the propagation pattern for the news i, with N the number of examples and $y_i \in Y = \{0, 1\}$ is the label of a graph G_i (0 is a fake news and 1 a real news), the goal is to learn a mapping $g : G \to Y$ that labels each graph.

3.1.2 Project Management

Before we dive in the main topics of the project, the different tools used for the project are as follows:

- First, I use git for version control, which is useful to keep a track of the experimental branches.
- I receive a Twitter Developer Account from the University of Bristol to collect more data than the one I could get for free by applying on the Twitter developer platform.
- I use a remote server provided by the University of Bristol for data processing and model training. This server speeds up the computing time of codes and helps me handle some memory issues I was facing with my own device.
- The code is written in Python, where I use different libraries for data pre-processing and machine learning models building.
- Different libraries are available such as Pytorch-geometric [13], Deep Graph Library [44] and DeepSnap [26]. The two libraries have been experienced and show pros and cons. Pytorch-geometric is probably the most research-oriented library, providing a lot of freedom to build any advanced graph machine learning model and is well documented for homogeneous graph.
- For heterogeneous graph data structure, I would recommend the use of Deep Graph Library as it provides more resources to handle this type of structure.

3.2 Structure of the project

The project involves different steps that are often common in many data science projects:

- First, we have to collect the raw data, which is big and of different types.
- Once the data is collected, we need to store it to transform the data in a way that it can be understood by a deep learning model. This step is one of the most important step of a data science project, called data pre-processing [12], it can be itself divided in many sub-fields: data cleaning, exploratory data analysis, feature engineering... One particular important step of the project is to transform our text and image data to get sentence embeddings and image features. This has been done using two complex pre-trained models, i.e., BERT for the NLP part and VGG19 for the CV part.
- Once the data pre-processed, we need to transform it to get a graph dataset structure. It can also be included in the pre-processing step of the data science pipeline. Thus, we need to specify all the different types of nodes and create matrices that point out which nodes are connected to which.
- Once the data is in the correct format, we can start building our graph neural network model. The graph neural network takes as inputs graphs and is passed in graph neural network layers and traditional neural network layers. Two different machine learning models have been built, one handling homogeneous graphs and one handling heterogeneous graphs.
- Then, we train the model using a train set; we use an optimisation method to update the weights by minimising a loss function. Also, we optimise the learning hyper-parameters of the model using a validation set. To test the performance of the model, we keep a test set that has not been used for training the model and we measure how well the model predicts the labels of the test.

The different bullet points can be summarised in the Figure 3.1:

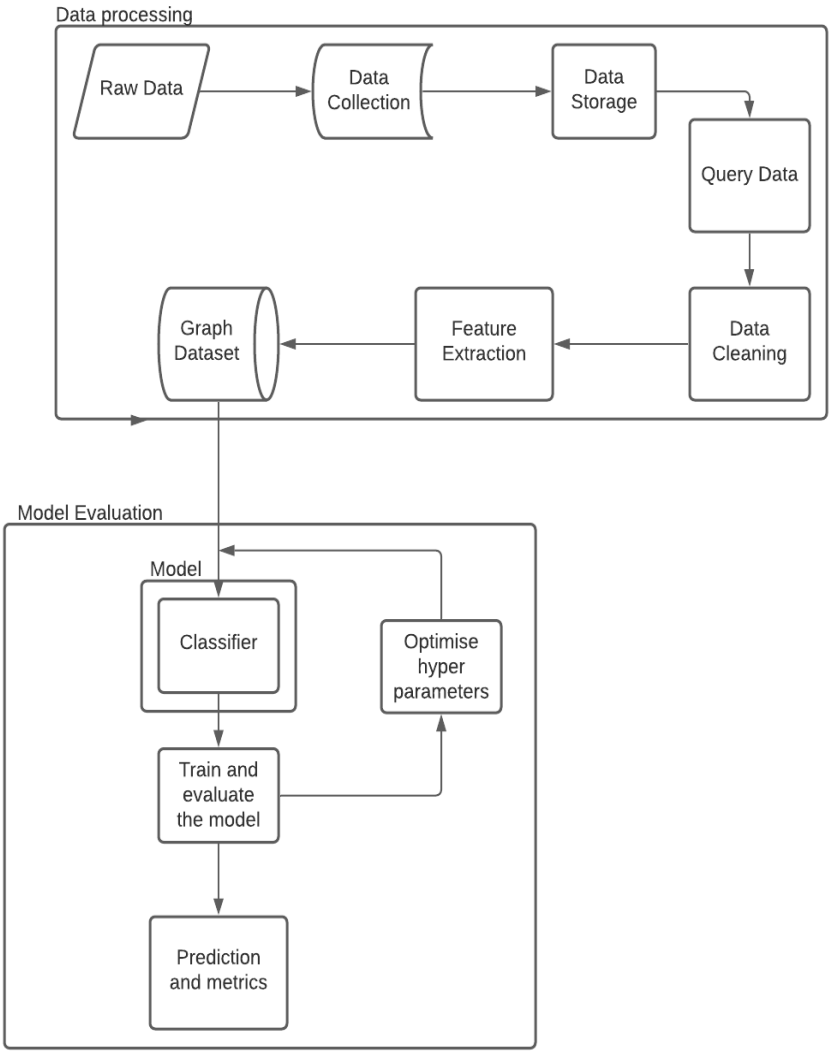

Figure 3.1: A diagram illustrating the system architecture

Looking to the diagram, we can divide the structure of the project in three different parts:

- Data processing
- Model construction
- Model Evaluation

3.3 Collecting the dataset

The FakeNewsNet dataset can be collected using the FakeNewsNet repository [36, 38, 37] available on the GitHub repository. We collect fake news from the Politifact and GossipCop, two fact-checking websites that rate the accuracy of respectively political and gossip news. Using the FakeNewsNet repository, the downloaded files are stored in the format presented in the appendix B.

A part of the code provided by the FakeNewsNet repository [36] is used to build the dataset. The tweets and news content are collected using the FakeNewsNet API, however, to collect other resources such as the retweets and the user profiles, personal code has been created to collect the data. The reason is that when using the FakeNewsNet repository code, it was trying to download data that has been deleted by Twitter, resulting in some errors/warnings. The reason behind that is probably that the information

that has been deleted was related to fake news content and thus, because of Twitter policies, they deleted the tweets/retweets.

To collect the retweets and the user profiles, we use the request library. For each tweet collected using the FakeNewsNet repository, I collect the tweets ids which correspond to the filename of each tweet, e.g., in the appendix A, we can see the filename 1170671160715233351. The tweet id is needed to collect the data, and thanks to the request library, we can have access to all the other information related to this tweet using the Twitter API.

The structure of the dataset is dirty and needs pre-processing; we will first create a code that can query the data easily. In the section 2.6.3, We saw that one tweet has a lot of unnecessary features so we have to clean each type of data collected such as the news content, the tweets content, the retweets and the user profiles content. We are interested in the text of the different files, thus, we have to remove all the other unnecessary information such as the user id, the number of followers, URLs, etc..

3.4 Querying the data

To clean the data, we first need to build a code to query each type of data. Thus, a python file called query_data.py has been created to query all the different types of data collected to build our dataset.

3.4.1 Query the news

To collect a particular news article, we first get a list composed of all the filenames of a source news such as Politifact for a particular label, e.g. the fake labels. Then, like illustrated in the listing 3.1, we can query a particular news and remove some features that are not necessary and keep the important features for feature extraction.

```
def get_news_files(path, news_source, label):
    path = path + news_source + '//' + label
    os.chdir(path)
    list_news_files = os.listdir(path)
    return list_news_files

def get_news_content(path, news_source, label, filename):
    path = path + '//' + news_source + '//' + label + '//' \
           + filename + '//'
    os.chdir(path)

    if os.path.isfile('news␣content.json') == True:
        f = open(f'news␣content.json',)
        data = json.load(f)
        keys = ['url', 'images', 'top_img', 'keywords', 'canonical_link',
                'meta_data', 'movies', 'publish_date', 'summary']

        for key in keys:
            data.pop(key)

        news_content_df = pd.DataFrame.from_dict([data])
        return news_content_df
```

Listing 3.1: Python code to query news

3.4.2 Query the tweets, retweets and user profiles

To query the tweets, the retweets and the user profiles, similar codes are designed. The only difference between querying the social context information and the news is that for a news file, there are multiple tweets, retweets and user profiles files related to the particular news file that we need to query. Thus, we get the list of all the tweets files and do a **for** loop to query all the tweets related to the news file, as illustrated in the listing 3.2:

```
def get_tweets(path, news_source, label, filename):
    path = path + '//' + news_source + '//' + label + '//' \
           + filename + '//tweets'
    os.chdir(path)
    list_tweets = os.listdir(path)
    tweet = pd.read_json(list_tweets[0], typ = 'series')
    tweet = tweet[['created_at', 'id', 'text']]
    df = pd.DataFrame([tweet])
    for tweet_id in list_tweets[1:]:
        tweet = pd.read_json(tweet_id, typ = 'series')
        df = df.append(tweet, ignore_index = True)
    return df[['id', 'text']]
```

Listing 3.2: Python code to query tweets

We are using the same process to collect both retweets and user profiles.

3.4.3 Querying the images

In the news content file, they give the urls of the images related to the news article. Thus, we first need to collect all the images of the news article by scraping the images using the related urls of each image using the request library. Once each image is collected and stored in its related news directory, we design a function that query all the images, for image features extraction. The benefit of querying all the images together is that we can process all the images together instead of processing each image one by one.

3.5 Feature Extraction

3.5.1 Text Feature Extraction

To extract features from text, as we introduced in the 2.5.4, one popular pre-trained model is the BERT model, developed by researchers from Google [10]. The BERT model has been trained on more than 70 languages and has been trained using the Wikipedia's data (2.5B words) and publicly-available books (800M words).

This being said, the BERT model has a complex and deep architecture, composed of 12 transformer layers, each containing 12-self attention layers with feed-forward layers - it represents 110 million parameters in total. Thus, extracting features using a BERT model can be computationally expensive if the dataset is big. In our case, we have a lot of data consisting of text that we need to process, thus, using GPUs to process the data is essential. GPUs can be used for deep learning because most deep learning architectures are making use of matrices operation and GPUs are very efficient to calculate matrices operation while being also good at fetching large amounts of memory [28].

For our application, we use the BERT-base-uncased model [10], which is the traditional BERT model, returning 768-dimensional contextualised word embedding. Different libraries are available to load a pre-trained model; the transformers [48] library has been used, which provides state-of-the-art Natural Language Processing pre-trained models for Pytorch and TensorFlow. The BERT model takes as inputs text, that we first tokenize using Bert Tokenizer. It tokenizes all the sentences and maps the tokens to their word IDs. The listing 3.3 illustrates how we get sentence embeddings using BERT pre-trained model:

```
def get_inputs(sentences, tokenizer, device, max_length):
    # Tokenize sentences and map the tokens to their segment IDs.
    input_ids = []
    attention_masks = []

    # For every sentence
    for sent in sentences:
        encoded_dict = tokenizer.encode_plus(
                            sent,   # Sentence to encode.
                            add_special_tokens = True,
```

```
                                max_length = max_length,
                                padding = 'max_length',
                                return_attention_mask = True,
                                return_tensors = 'pt',
                                truncation = True
                           ).to(device)

        # Add the encoded sentence to the list.
        input_ids.append(encoded_dict['input_ids'])


    # Convert the lists into tensors.
    input_ids = torch.cat(input_ids, dim=0)
    segment_ids = torch.ones_like(input_ids)

    return input_ids, segment_ids

def get_model(device):
    # Load pre-trained model (weights)
    model = BertModel.from_pretrained('bert-base-uncased',
                                      output_hidden_states = True,
                                      )
    model.to(device)

    model.eval()
    return model

def get_embeddings(model, input_ids, segment_ids):

    with torch.no_grad():
        outputs = model(input_ids, segment_ids)
        hidden_states = outputs[2]

    token_vecs = hidden_states[-2]
    # Calculate the average of all 22 token vectors.
    sentences_embedding = torch.mean(token_vecs, dim=1)
    return sentences_embedding
```

Listing 3.3: Code to extract sentences embeddings using BERT

The get_inputs function returns the inputs ids for the pre-trained model and the sentences ids. The get_model function loads the pre-trained model and puts it in the evaluation mode, meaning that we are in the feed-forward operation. Finally, to get our embeddings, we use the get_embeddings function. The hidden_states is a four dimensions object, composed of:

- The layer number (13 layers)
- The batch number
- The token number
- The hidden unit / feature number (768 features)

Here, we have 13 layers because we also have the input embeddings included. To get sentences embedding, we average the second to last hidden layer of each token producing a single 768 length vector. Thus, the resulted sentences_embedding object is a tensor containing sentences embedding, each sentence of length 768. As an example, we query tweets from the file gossipcop-93262280, which is a news file composed of 572 tweets. We use the BERT model to extract features from all the tweets. The obtained sentences_embedding object is an object of dimension (572, 768).

We can visualise the embeddings using T-SNE algorithm [42], a non-linear method allowing to represent a set of points of a large-dimensional space in a two or three-dimensional space. The T-SNE algorithm calculates the conditional probability that a point x_i picks x_j as its neighbour:

$$p_{j|i} = \frac{e^{-\frac{||x_i - x_j||^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{||x_i - x_j||^2}{2\sigma_i^2}}}$$

where σ_i^2 is the variance of a Gaussian centred on x_i. Then, we create a similar metric in the low-dimensional space:

$$q_{j|i} = \frac{e^{-\frac{||y_i - y_j||^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{||y_i - y_j||^2}{2\sigma_i^2}}}$$

If the points y_i and y_j map correctly the similarity between the high-dimensional data points x_i and x_j such as a sentence embedding of length 768, the conditional probabilities $q_{j|i}$ and $p_{j|i}$ will be equal. To measure the mismatch, we use the Kullback-Leibler divergence [9]:

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log(\frac{p_{j|i}}{q_{j|i}})$$

where P_i represents the conditional probability over all other data points given data point x_i and similarity for Q_i. To minimise this cost function, we use the gradient descent.

To get even better results, we often combine PCA and t-SNE, another widely used technique for dimensionality reduction. Principal Component Analysis can be defined as the orthogonal projection of the data onto a lower dimensional linear subspace, known as the principal subspace, such that the variance of the projected data is maximised.

Thus, we use t-SNE and PCA together to visualise how well sentences are embedded by transforming our embeddings into a lower dimensional space so that we can visualise it:

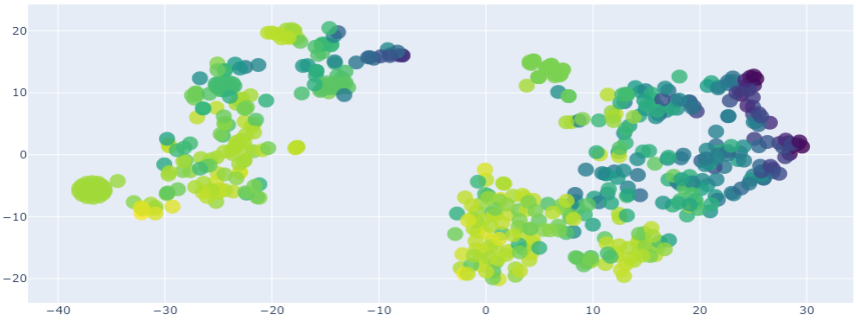

Figure 3.2: Visualisation of the embedded tweets

The colours describe the different text tweets. The points which have similar colours are points which are supposed to be close, i.e., points with close sentences embeddings. We can check visually how close are the meanings of two sentences by comparing the content of two purple points, presented in the Fig. 3.3 and 3.4:
We can see that both tweets are close, except that the first person that tweeted the tweet is jealous of Taylor Swift while the second person is jealous Harry Styles, two celebrities.

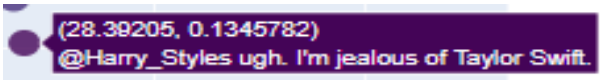

Figure 3.3: Tweet content of the first point

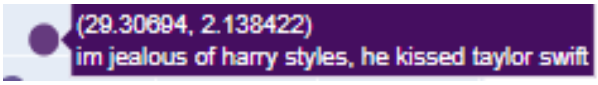

Figure 3.4: Tweet content of the second point

3.5.2 Image Feature Extraction

Similarly to text feature extraction, we achieve feature extraction from images using the VGG-19. We load the model using the torchvision library. The torchvision [1] library is part of the pytorch projects and consists of popular datasets, model architectures, and common image transformations for computer vision. The VGG-19 pre-trained model takes as input images and returns features of length 4096 by removing the top layers. The code is provided in the image_feature_extraction.py python file.

3.6 Building the Graph Dataset

3.6.1 The data used

The FakeNewsNet dataset is big and provides a lot of social context information such as the user tweets time-lime, the user followings and followers. For this project, we only use the news content, the tweets, the retweets and the user profiles to build the dataset. The principal reason is that the Twitter API has some restrictions regarding the quantity of data that can be downloaded in once. For instance, for the tweets and retweets, we can respectively collect 450 and 300 objects at once before we exceed the rate limit. When the rate limit is exceeded, we have to wait 15 minutes before we can start collecting data again. That is why collecting all the data took a lot of time, about 3 weeks to collect the Politifact dataset and more than 2 months to collect the GossipCop dataset. Also, with the amount of time we had for this project, we could only collect the tweets, retweets and user profiles because the rate limit of the user time-lime tweets, the user followers and followings downloads is 15 objects so we have to wait 15 minutes a lot more.

3.6.2 Different types of graph structure

The dataset we want to build is a dataset composed of multiple graphs, each one representing the news information (text and images) and its social context. Different approaches can be imagined to build the graph. The graph can be heterogeneous or homogeneous and directed or uni-directed:

1. A homogeneous graph is a graph only composed of the same type of edges and nodes. A typical homogeneous graph can be a social network where nodes represent users and edges represent friendship relation.

2. A heterogeneous graph is a graph composed of multiple edge types and node types, contrary to the homogeneous graph. In our context, a heterogeneous graph can be a graph where nodes represent news, tweets, retweets, users and the edges represent the relationship between each node type (e.g. a user is the author of a tweet).

3. A directed graph is a graph composed of nodes and edges where edges give information of the direction of the relation, e.g., I am following Taylor Swift but she is not following me on a social media.

4. An undirected graph, unlike the directed graph, does not give any direction information for the edges. Thus, we use undirected graphs when the relation type of the edges is undirected, such as a friendship relation on Facebook.

[1] https://github.com/pytorch/vision

We build two different types of datasets, one involving undirected 4 homogeneous 1 graphs - like the one presented in the UPFD framework - and the other one composed of undirected 4 heterogeneous 2 graphs. The reason that leads to build a homogeneous graphs dataset is that it already proves to be very effective so we will be able to judge our pipeline performance and compare it with the results of the UPFD paper. Also, we build a heterogeneous graph machine learning model to add image features and check if adding images increase the accuracy of the predictions.

3.6.3 Build a homogeneous graph

FakeNewsNet represented as a homogeneous graph

To build a dataset composed of homogeneous graphs, we consider that all the nodes have the same type. Thus, all the nodes must have the same features. The first node of the graph, also called the root node, is the news node, containing the text of the news. Then, we connect all the tweets related to the news. Finally, we also link the user profile and the retweets affiliated to each tweet. For each type of node, the node feature contains the text of the object:

- for the news, the text corresponds to the article.
- for the tweet, the text corresponds to the content inside the tweet.
- for the retweet, the text corresponds to the cited tweet and additional text written by the user which retweeted the tweet.
- for the user profiles, the text corresponds to the description of the user. The description on Twitter is often a quick profile presentation given by the user.

Supporting technologies

To build a homogeneous graph, different approaches are possible. We can either directly build a graph dataset using PYG [13] or DGL [44] libraries - which provide resources to build graph data structure that can serve as input for a graph machine learning model - or we can build our graphs using a network library like NetworkX or iGraph [2], which are famous libraries used to create, manipulate and study complex networks. Then, both pytorch-geometric [13] and DGL [44] provide functions to transform a NetworkX/iGraph object into a pytorch-geometric/DGL data object.

We use the second approach to build our homogeneous graphs, using the NetworkX library to create graphs and using Pytorch-geometric library and notably the from_networkx function from the library to transform a NetworkX graph object into a PYG data object.

Adding image features in homogeneous graphs

One of the issues of using a homogeneous graph is that all the nodes must have the same feature size. However, real-word applications often deal with objects of different modalities. We assume that our nodes are of the same type, so the graph does not make any difference between a node that is a tweet and a node that is news article. But in reality, we would like to add another feature to the news node which is the image features. One possibility is to add a padding to every node, i.e. adding zeros so that the size of our features is the same for each node.

Without the images, we have nodes of size 768, as the sentence embeddings is a vector of size 768. To add image features, we concatenate the sentence embeddings with the image features [8], so we have a node feature of length $768+4096 = 4864$. So for all the other nodes except the news node, we concatenate the sentence embeddings with a list of 4096 zeros. However, using padding is not a good idea in this case, as it may decrease a lot the performance of the model.

3.6.4 Build a heterogeneous graph

FakeNewsNet represented as a heterogeneous graph

As we explained before, one purpose of this project is to propose a new way of classifying fake news data using heterogeneous graphs. Using heterogeneous graphs, we can add the images features to the news

[2] https://github.com/igraph/igraph

nodes; in a heterogeneous graph, we specify all the different nodes and edges, thus, each node type can have his own feature size.

Supporting technologies

To build a heterogeneous graph, we can either use the DGL or the PYG library. Unlike the homogeneous graphs, we cannot use NetworkX or iGraph to build the heterogeneous graphs because DGL and PYG do not provide support for transforming a heterogeneous graph into a PYG/DGL Data object.

Adding image features in heterogeneous graphs

The advantage of heterogeneous graphs over homogeneous graphs is that we can have different node types and thus node features. Therefore, we can easily add image features to our news nodes.

3.7 Implementing a graph neural network model

Now that we have graphs, we can start building a model that is suited for graphs:

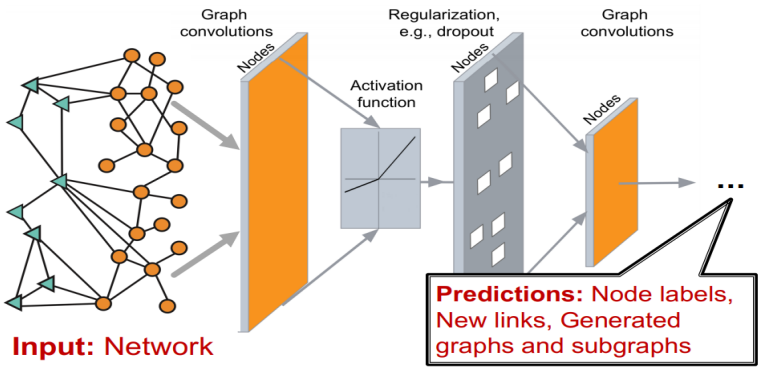

Figure 3.5: Structure of a graph neural network [25]

Building a graph deep learning model is not very different than building a model using traditional data structure, except that the data is a network and the layers are graph layers.

3.7.1 Building graph neural networks with homogeneous graphs

To build a homogeneous graph neural network, we use pytorch-geometric. The architecture we use to build our model is a combination of different layers as described in the listing 3.4:

```
class Model(torch.nn.Module):
    def __init__(self, args, concat=False):
        super(Model, self).__init__()
        self.args = args
        self.num_features = args.num_features
        self.nhid = args.nhid
        self.num_classes = args.num_classes
        self.dropout_ratio = args.dropout_ratio
        self.model = args.model
        self.concat = concat

        if self.model == 'gcn':
            self.conv1 = GCNConv(self.num_features, self.nhid)
        elif self.model == 'sage':
            self.conv1 = SAGEConv(self.num_features, self.nhid)
        elif self.model == 'gat':
            self.conv1 = GATConv(self.num_features, self.nhid)
```

```
        if self.concat:
            self.lin0 = torch.nn.Linear(self.num_features, self.nhid)
            self.lin1 = torch.nn.Linear(self.nhid * 2, self.nhid)

        self.lin2 = torch.nn.Linear(self.nhid, self.num_classes)

    def forward(self, data):

        x, edge_index, batch = data.x, data.edge_index, data.batch

        edge_attr = None

        x = F.relu(self.conv1(x, edge_index, edge_attr))
        x = gmp(x, batch)

        if self.concat:
            news = torch.stack(                                    \
            [data.x[(data.batch == idx).nonzero().squeeze()[0]] \
            for idx in range(data.num_graphs)])
            news = F.relu(self.lin0(news))
            x = torch.cat([x, news], dim=1)
            x = F.relu(self.lin1(x))

        x = F.log_softmax(self.lin2(x), dim=-1)

        return x
```

Listing 3.4: Graph Neural Network model for homogeneous data

First, we use a graph neural network layer as introduced in the section 2.8. Depending on the parameters we specify using the argparse library, allowing line-command options to change the parameters of the model, we can either use GCN, SAGEGraph or GAT layers. Then, we use the ReLU activate function and add a global max pooling layer. Finally, we get the news nodes and apply a feed-forward neural network on the news embeddings. We concatenate the news and the outputs obtained by the global max pooling layer and apply two others feed-forward neural layers. The architecture is described in the figure 3.6:

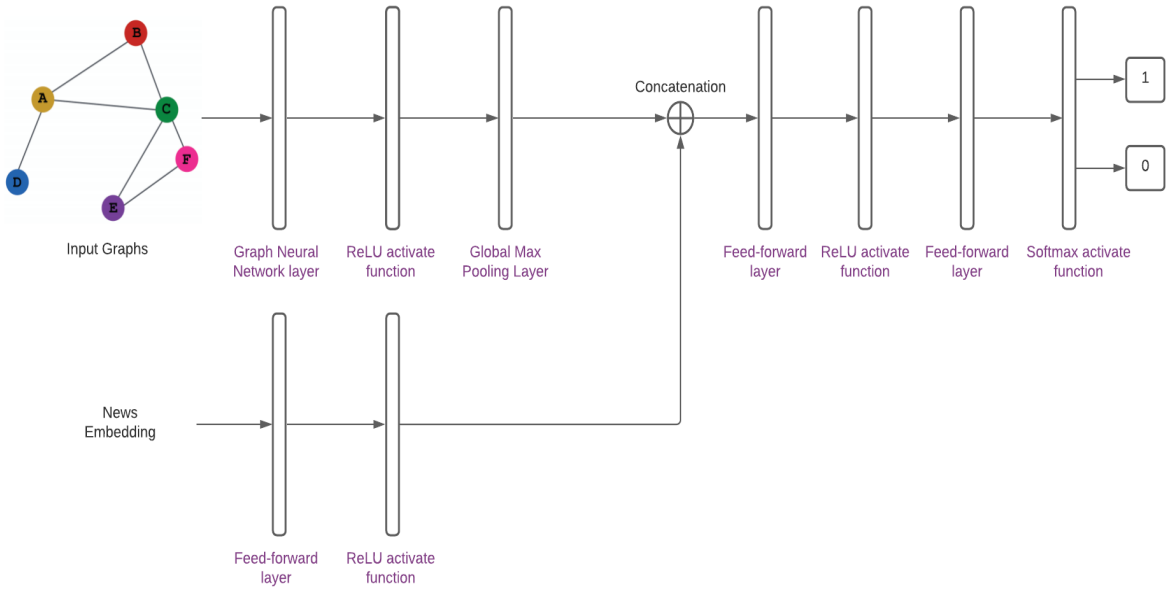

Figure 3.6: Structure of a GNN model for homogeneous graphs

This graph neural network model is a multi-modal model that uses social context and news text information to learn how to differentiate fake from real news. In other words, we obtain a binary classifier, taking in inputs graphs and returning the predicted label.

3.7.2 Building graph neural networks with heterogeneous graphs

We build heterogeneous graphs using the Deep Graph Library. The main difference between building a homogeneous graph and a heterogeneous graph is that we specify all the different relations when building the graph. In our case, the different type of relations we have are the following:

- The relation between news nodes and tweet nodes has been named **mentionened by** because the news information has been mentioned on social media thanks to tweets.
- The relation between tweet nodes and retweets nodes has been named **spread** because retweets help the information to be spread on social media.
- The relation between tweet nodes and user nodes has been named **tweeted by** because the users are the authors of the tweets.

We specified three types of relations, but in reality, we have six relations. Because we want to have undirected graphs, we need to create relation types in both ways, e.g., news $\leftrightarrow$ tweets. It is important to specify all these relations because unlike homogeneous graphs, heterogeneous graphs are graphs that contain different types of nodes and edges. As we explained earlier, different types of nodes and edges often have different types of attributes that capture the characteristics of each node and edge type. In homogeneous graphs, the message passing and aggregation can be computed through all the nodes. In the context of heterogeneous graphs, the message passing is splitted into two parts:

- Message computation and aggregation for each relation
- Reduction that merges the aggregation results from all relations for each node type.

Thus, we build a model which takes in consideration all the different type of relations and we build a graph neural network for each type of relation, presented in the appendix C.

3.7.3 Data Splitting

The different sets

Now that we constructed the model, we have to split the model in different sets. This is an essential step of the data science pipeline [34]. The data must be splitted into three parts: a training set, a validation set and a test set. The training set is the data used during the learning process of our neural network model. It is used to fit the weights of our classifier. The goal of the learning process is to produce a trained model that generalises well to new unknown data.

In Deep Learning, there is a hyper-parameter called **epochs**, which defines the number of times the model will be trained on the training set [16]. At the epoch n, the model classifies the output of each input in the training set - when the predictions are computed, we calculate the loss function and then adjust the weights learned from the epoch $n-1$ using gradient descent.

The validation set, is the set of data that is used to validate our model during the training. We just mentioned that during each epoch, the model was trained on the training data and the loss function was calculated based on the predictions we made, when we have a validation set, we also classify the inputs from the validation set. It is important to mention that the weights are not updated on the loss calculated on our validation data.

One of the reasons that motivates the use of a validation set is to assure that the model is not over-fitting. We validate the model on the validation set during the training phase by calculating the predictions of the model on the validation set - therefore, we can verify that the predictions are as good on the validation set as the training set. If the predictions on the validation set are good, we are sure that the model is able to generalise and make accurate classifications on data it was not trained on.

Finally, the test set is the set of data that is used to test the model, after all the weights have been

correctly updated. The test set is used to make sure that the model is able to generalise before deploying the model to production.

Using batches

We can also use batches to divide our training set into small sets. The number of batches we are going to use for our model is also a hyper-parameter. In fact, rather than specifying the number of batches, we define the size of the batch, i.e, the number of samples we are using to update the parameters of our model. The size of the batch has a repercussion on the the type of gradient descent we are using. There are three different types of gradient descent depending on the size of the batch:

- **Batch Gradient Descent**: The batch size is the size of the training set.
- **Stochastic Gradient Descent** [19]: The batch size is equal to one sample.
- **Mini-Batch Gradient Descent** [33]: The batch size is a small part of the training set, superior to one and less than the training set size.

3.8 Training the model

In the training part, we try to learn the weights that minimise our loss function and hence, get better predictions on the training set. The code of the listing 3.5 is the code used to train the model.

```
min_loss = 1e10
val_loss_values = []
best_epoch = 0

t = time.time()
model.train()
for epoch in tqdm(range(args.epochs)):
    loss_train = 0.0
    out_log = []
    for i, (batched_graph, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        out = model(batched_graph)
        loss = F.nll_loss(out, labels)
        loss.backward()
        optimizer.step()
        loss_train += loss.item()
        out_log.append([F.softmax(out, dim=1), labels])
    acc_train, _, _, _, recall_train, _ = eval_deep(out_log,
                                                    train_loader,
                                                    hetero = True)

    [acc_val, _, _, _, recall_val, _], loss_val = compute_test(valid_loader)

    print(f'loss_train: {loss_train:.4f}, acc_train: {acc_train:.4f},'
                          f'recall_train: {recall_train:.4f},'
                          f'loss_val: {loss_val:.4f},'
                          f'recall_val: {recall_val:.4f}')
```

Listing 3.5: Code used to train the model

The **for** loop is the loop used to train the model at each new epoch. When we have trained the model at an epoch i, we compute the outputs and we calculate the loss function and some metrics to evaluate the performance of the training. We also evaluate the performance of our model on the validation set, as explained in the section 3.7.3. To minimise the loss function, we calculate the gradient of the loss function and update the weights. Using gradient descent, we move towards a local minima of the loss function.

3.9 Evaluation of the model

Now that we trained our model, we need to evaluate its performance on the test set. To do so, we pass our test set into our model and it returns the predictions of the news. If the prediction is 0, the news is fake, while if the news is labelled 1, the news is real. To evaluate the performance of the model, we compute the metrics described in the section 2.2.3.

```
[acc, f1_macro, f1_micro, precision, recall, ap], \
test_loss = compute_test(test_loader, verbose=False)
    print(f'Test set results: acc: {acc:.4f}, \
    f1_macro: {f1_macro:.4f}, f1_micro: {f1_micro:.4f}, '
                  f'precision: {precision:.4f}, recall: {recall:.4f}')
```

Listing 3.6: Evaluation of the model

The compute_test function is a function designed to get the predictions from a Pytorch [3] Data Loader object and computes the different metrics based on the real values and our predictions.

[3] https://github.com/pytorch/pytorch

Chapter 4

Project Results

4.1 Introduction

This chapter is designed to present the results of the project and explains the different issues which occurred during the project. We will also present the decisions took during the project that leaded to these results.

4.2 Parameters of the model

Before the results of the model are presented, we give an overview of the different hyper-parameters used to obtain the results. In machine learning, a hyper-parameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are derived via training. The hyper-parameters are not updated during the learning process - choosing the appropriate hyper-parameters to get a good model is the model selection step. In our pipeline, there are a lot of hyper-parameters that can potentially increase or decrease the performance of the model. First, a summary of the model architecture is given in 4.1 to recall the different models where hyper-parameters have been selected:

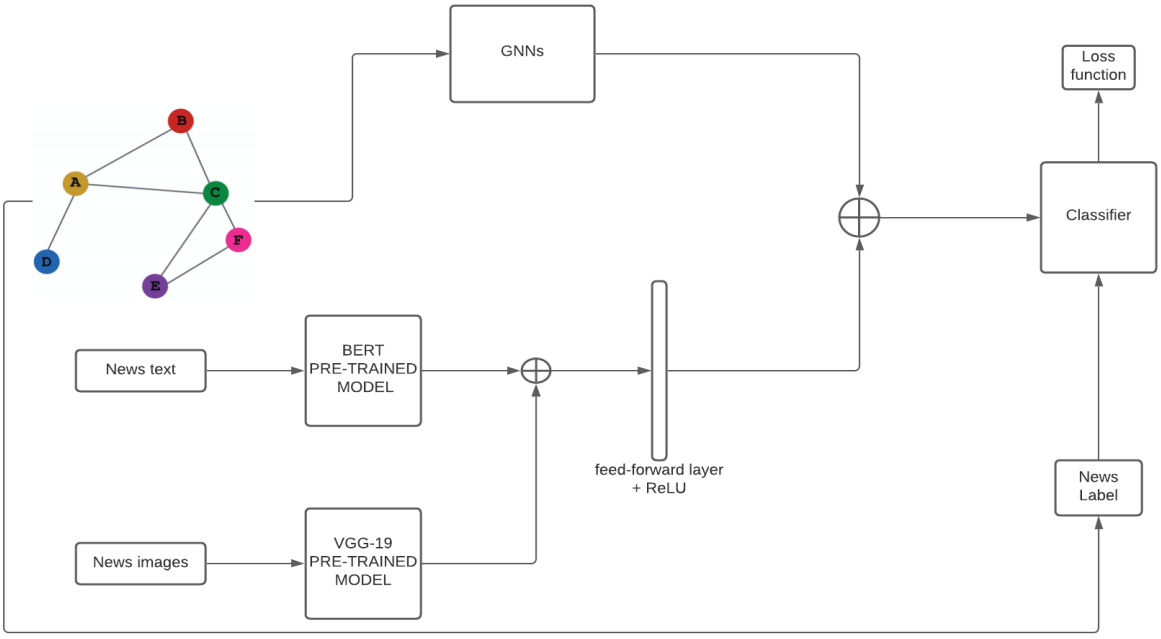

Figure 4.1: Architecture of the global Model

4.2.1 Text pre-processing

Although the embeddings visualised using T-SNE were showing good results, we did not explain how we can generate such results. There are two parameters that can possibly impact the performance of our embeddings:

- The number of maximum tokens we fix for the sentences.
- The type of BERT model we choose

Type of BERT pre-trained model

We choose the bert-base-uncased pre-trained model as our pre-trained model to embed the sentences. However, we could have chosen the bert-large-uncased pre-trained model, which is an improved version of the bert-base-uncased model, composed of 16 self-attention layers [43] against 12 in the base version. Using a bert-large-uncased model returns a sentence embeddings of dimension 1024. In consequence, the sentence embeddings returned using a bert-large-uncased model often describes better the sentences than a classic bert-base-uncased model. However, the time to compute the embeddings is higher with the large model. the classic version of the BERT model is used because the number of sentences to pre-process is high and can be computationally expensive.

Number of tokens

The number of tokens is a hyper-parameter that we have to precise when computing the sentences embeddings. Our news, tweets, retweets and user profile descriptions have different number of words. The issue is that all the news do not have the same number of tokens and when we pass our sentences into the BERTtokenizer, we have to specify a limit number of tokens. In consequence, for the sentences where the number of tokens is behind the limit specified, we use a padding. If the sentences have a number of tokens higher than the limit, we have to remove a part of the sentence. To determine the appropriate value, we can check the number of words inside each sentence. We take a certain number of tweets and news, an enough number to represent well the dataset:

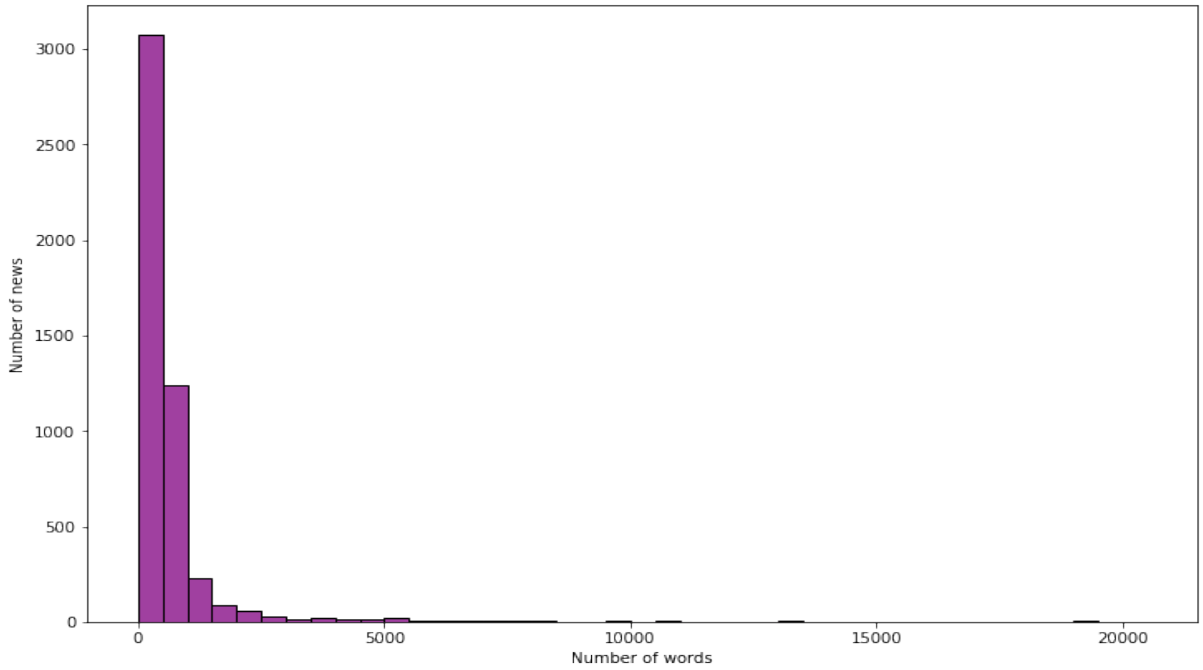

Figure 4.2: Distribution of the number of words for the news

For the news, we see that a majority of the news article have less than 1000 words. We could define the number of max tokens to 1000, however, the bert model cannot support a vector of size higher than 512. Thus, we can only have a document with 512 words. For the news article which exceed 512 words, we delete the additional words.

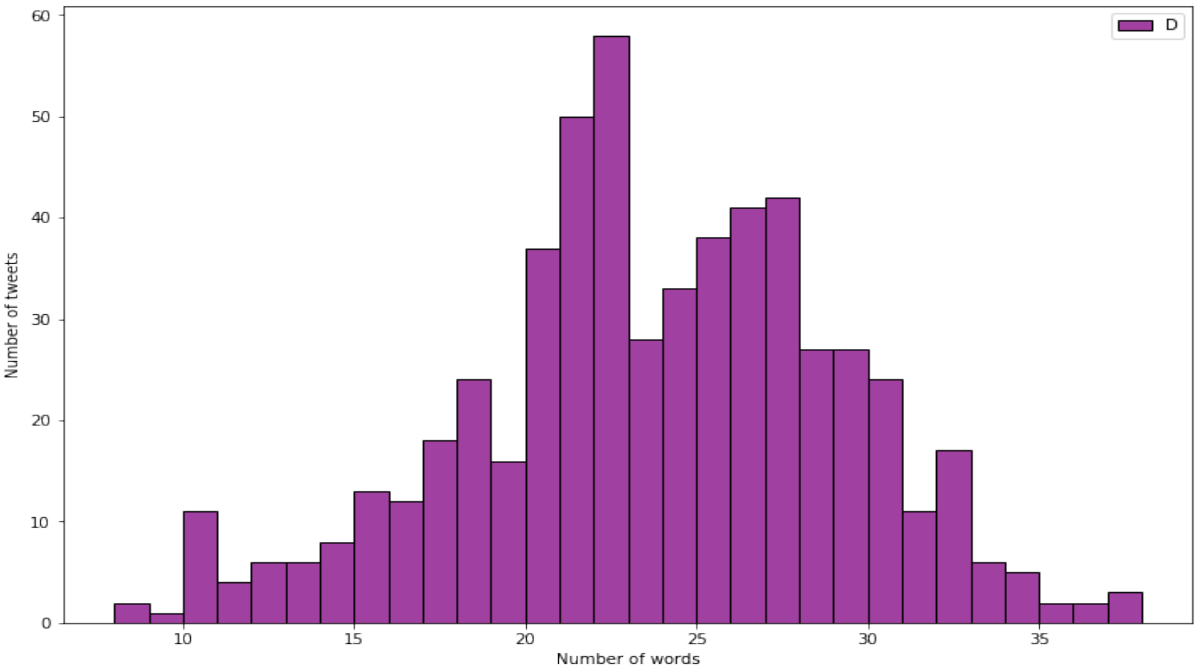

Figure 4.3: Distribution of the number of words for the tweets

For the tweets, we see that a majority of the tweets article have less than 25 words. Thus we can define the limit of tokens for the tweets to 30. That is not surprising, we recall that there is a limit of 280 characters today and before November 2017, the number of characters was limited to 140.

4.2.2 Hyper-parameters for the dataset

There are also some hyper-parameters which are important and have an impact on the performance of the model. Those presented here have already been mentioned in the section 3.7.3, such as the batch size:

- Finding the optimal size of the batch is not easy. The bigger the batch size, the lesser is the noise in the gradients so we get a better estimate of the weights, which lead to a better local minimum of the loss function [33]. However, bigger is the size of the batch, higher is the time needed to compute the gradients. Also, it is important to mention that usually, the loss function is not a convex function, so finding a local minimum does not assure that we find the global minimum. Thus, having an accurate gradient estimate does not guarantee that we found the global optimum. The advantage of keeping the batch size small is that it makes the gradient estimate noisy which might allow to bypass a local minimum during convergence. The different batch size used for the models are described in the appendix D.

- Finally, the proportion of data we set for the training set is also important and has an impact on the performance of the model [34]. If the training set is too low there is a chance that we do not have enough data to represent well the reality and hence, the neural network learns patterns only from a small number of news. If we increase the size of the training set, we increase the computing time of the training. Also, we have to make sure that we keep enough data for the test set so that the evaluation we make is robust. 70% of the dataset is dedicated to the training set, 10% of the dataset is for the validation set and 20% for the test set.

4.2.3 Hyper-parameters for the graph neural network model

For the graph neural network, there are a lot of hyper-parameters to take in consideration. This is one of the problems with Deep Learning, the number of different hyper-parameters to define increases with the complexity of the model.

Also, the type of graph neural network is an important parameter to consider. We compare the performance of our model with three different architectures: GCN, SAGEConv and GATConv. The architecture which is the more suited for our problem and finds the best predictions is the SAGEConv layer. This being said, there are way more hyper-parameters we can consider, such as the number of layers to use, the number of outputs we want our graph model to return at each layer, the type of activation function we choose.

Also, there are other type of layers that can be added such as dropout layer, which is layer used to reduce over-fitting by randomly setting some inputs units value to 0. The ratio of inputs that we set to 0 is also a hyper-parameter. The value we chose for the ratio is described in the appendix D.

Optimising all these hyper-parameters can be done using optimisation techniques such as Bayesian optimisation [14], but it is computationally expensive.

4.2.4 Hyper-parameters for the training

Now that we defined the parameters of our model, we still have to define some hyper-parameters that are going to influence the learning phase of our model:

- The type of optimisation technique is an important hyper-parameter in all Deep Learning applications. Until now, we introduced the Batch Gradient Descent and its variants, the Stochastic Graident Descent and the Mini-Batch Gradient Descent. However, the real optimisation technique chosen for this problem is the Adam optimisation [22] algorithm. It is an extension of the stochastic gradient descent that combines advantages of two other extensions of stochastic gradient descent called AdaGrad and RMSProp. Adam improves the performance of the learning on problems dealing with sparse gradients such as NLP and CV and also does well on non-stationary problems.

- The loss function is also a hyper-paramater that has an impact on the performance of the learning. The loss function we use is the negative log likelihood loss. This is a widely used loss function with the softmax function, called the Cross-Entropy Loss function when combined together. We use the negative log because we want to maximise the probability of choosing the correct category. Thus, maximising the probability of choosing the correct probability means minimising the negative log likelihood.

- The learning rate is a tuning parameter in the optimisation algorithm, i.e., the Adam algorithm. The learning rate determines the step size at each iteration while moving toward a minimum of a loss function. As we know, the descent direction is determined by computing the gradient of the loss function and the learning rate determines how big a step is taken in that direction. Unfortunately, like many hyper-parameters, there is a trade-off to make when defining the learning rate. If the learning rate is too high, the learning can jump over a minima and if the learning rate is too low, the learning will take a long time to converge and is likely to be stuck in a local minima [7]. The value used for the models is in the appendix D.

- Another hyper-parameter to consider is the weight decay. The weight decay is a regularisation technique used to limit over-fitting in a neural network [15]. A regularisation refers to the process of adding information to a problem, often to avoid over-fitting. This information can be considered as a penalty for the complexity of the model. Indeed, when we are facing under-fitting, it means that the model is not complex enough to find good predictions while facing an over-fitting scenario means that the model is too complex. Thus, the penalty provided by a regularisation technique enables to reduce the value of the weights. The new loss function takes the following formula:

$$L_{Regularised} = L + \lambda \sum_i w_i^2$$

  The parameter lambda is the parameter that gives importance to penalty. It is a positive coefficient. The value of lambda is often close to 0. Our value is listed in the appendix D

- The number of epochs has also an impact on training phase. If the number of epochs is too low, it means there is a risk that we do not train enough the neural network [6] to get the optimal weights to get a low loss function - when this is the case, we say that we are under-fitting our model. When we set a high number of epochs, it means we train a lot our neural network, and thus, there is a

risk that we train too much the neural network. Thus, there is a chance that our weights are well fitted to get good predictions on our training data, but the model is not generalising well, i.e., is not performing well on unseen data. In this case, we are over-fitting our model. The number of epochs we chose is explained in the appendix D.

4.3 Data Collection results

Even if the principal objective of the project was to build a model that classifies news, it is important to emphasise the size of the data that has been collected.

The collection of the data set has been done using both the FakeNewsNet repository and personal python code. Although we presented all the statistics of the dataset in the table 2.1, we did not retrieve all the data we could have collected using the FakeNewsNet repository. The reasons for that are multiple:

- Some news have been deleted and cannot be downloaded anymore.
- The time to collect the data is too long.

Moreover, the data collected is the raw data and need some data pre-processing, as we already explained. The data files can include missing data and corrupted data. Therefore, the number of graphs we get at the end is lesser than the number of news we collected. Also, we do not have the same amount of graphs for the heterogeneous and homogeneous datasets. In the heterogeneous graphs dataset, because we specify all the relations, if there is a graph where there is not a certain type of node, e.g., retweets, we decided to remove the graph from the dataset. We can compare the number of graphs we have with the UPFD framework, which also uses FakeNewsNet dataset for fake news detection.

UPFD graphs data statistics:

- For the politifact dataset: the model has been trained on 314 graphs, including 157 fake news.
- For the gossipcop dataset: the model has been trained on 5464 graphs, including 2732 fake news.

Project graphs data statistics for the homogeneous graphs:

- For the politifact dataset: we collected 323 real news and 349 fake news, so we have a total of 672 graphs in our dataset.
- For the gossipcop dataset, we collected 1713 fake news and 2163 real news, so we have a total of 3876 graphs in the dataset.

Project graphs data statistics for the heterogeneous graphs:

- For the politificat dataset, we collected 240 real news and 295 fake news, so we have a total of 535 graphs in the dataset.
- For the gossipcop dataset, we collected 1279 real news and 1709 fake news, so we have a total of 2988 graphs in the dataset.

To compare with the UPFD, we have more graphs for the politifact graphs dataset than the UPFD model but we have less graphs in the gossipcop dataset. Less graphs for the gossipcop are used for two reasons. First, it was taking a lot of time to collect all the graphs, and secondly, processing all these graphs could be hard computationally.

4.4 Model's performance

4.4.1 Loss function evolution

The model evaluation is the step where we check the performance of our model and we verify that our results are satisfying. To evaluate our model, we first check how the loss function evolves during the training of the neural networks. The goal of the loss function is to fit well our model during the learning phase. The loss of the model is usually lower on the training set than the validation set, this gap is often called the generalisation gap; the lower the gap, the higher is the generalisation. An optimal learning curve should describe the following points:

- The training loss decreases to a point of stability
- The validation loss decreases to a point of stability
- The generalisation gap is minimal

Loss curve for the homogeneous graph model

For the homogeneous graph model, the loss curve of the politifact dataset is described in the Fig. 4.4:

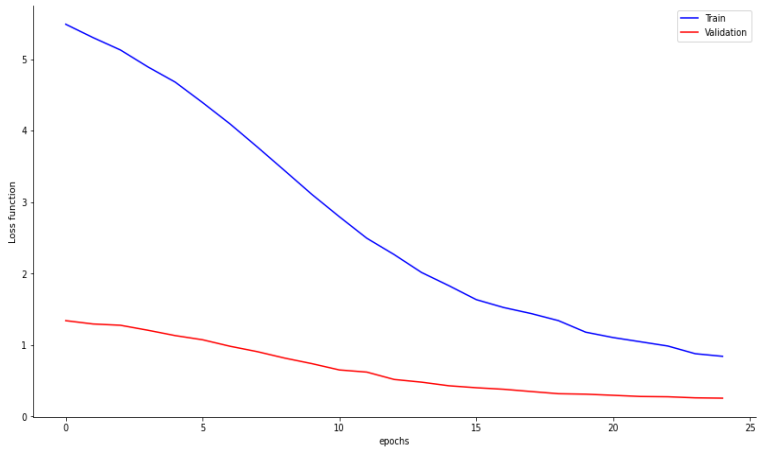

Figure 4.4: Loss curve of the homogeneous machine learning model for the politifact dataset

We can see that the loss function decreases as expected, both for the training and validation loss functions. One interesting point in this graph is the evolution of the validation loss, which is unusual; the validation loss is lower in the first epochs than the training phase. The main reasons for this observation are the followings:

- We apply regularisation during the training but not during the validation.
- The training loss is measured during each epoch while validation loss is measured after each epoch.
- Finally, there is also a possibility that the validation set is easier to classify than the training set.

For the gossipcop dataset, the loss curve is described in the Fig. 4.5:

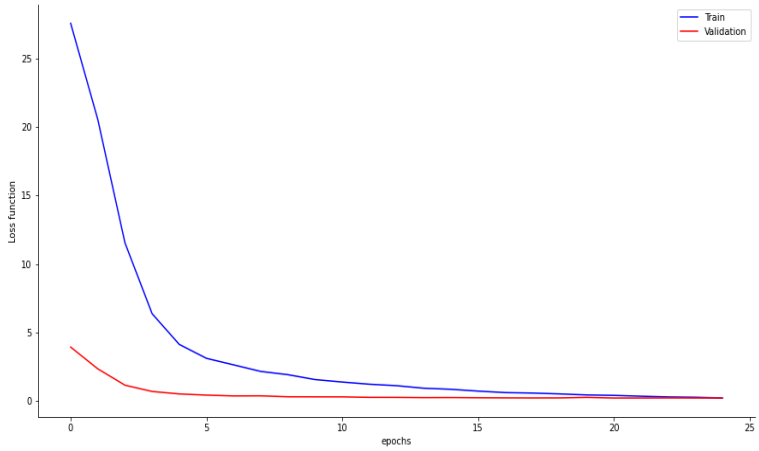

Figure 4.5: Loss curve of the heterogeneous machine learning model for the gossipcop dataset

We can draw the same conclusions on the gossipcop dataset as the politifact dataset.

Loss curve for the heterogeneous graph model

For the heterogeneous graph model, the loss curve is described in the Fig. 4.6:

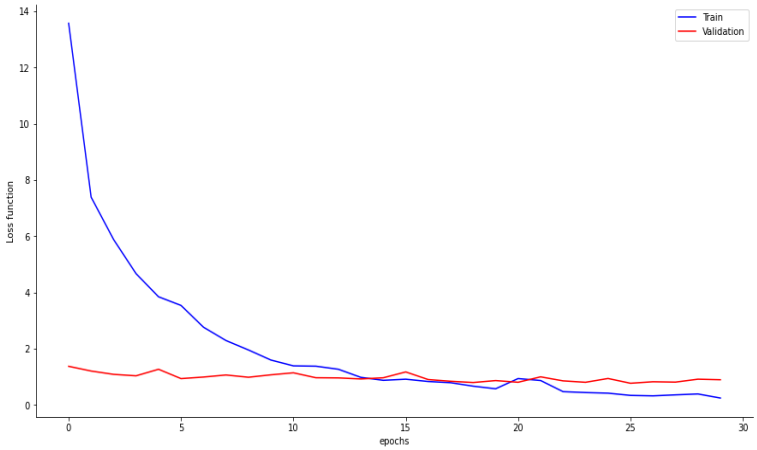

Figure 4.6: Loss curve of the heterogeneous machine learning model for the politifact dataset

Here, the validation loss is lower in the first epochs than the training phase and the validation loss is already stable.

The loss curve for the gossipcop dataset is given in the Fig. 4.7:

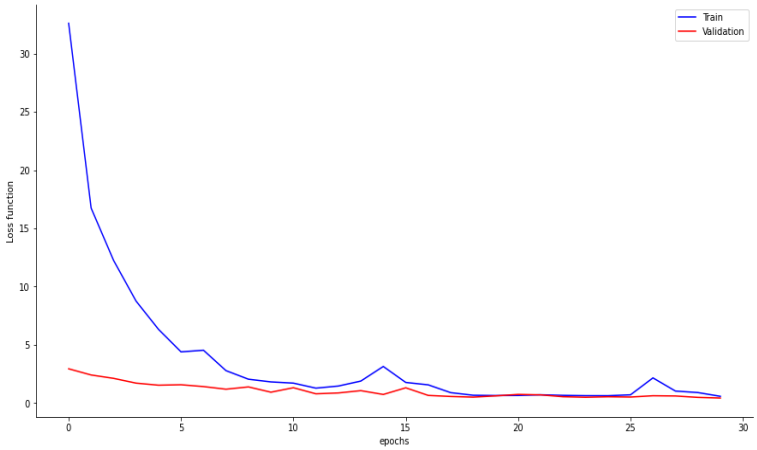

Figure 4.7: Loss curve of the heterogeneous machine learning model for the gossipcop dataset

The loss curve is similar as the other loss functions found.

4.4.2 Accuracy metric evolution

The accuracy curve is also useful to be plotted to understand the performance of our neural network during the training. Having a good accuracy curve assures that our model is able to make good predictions, i.e. determine fake and real news. The accuracy used is the one defined in the section 2.2.3, i.e.:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy curve for the homogeneous graph model

Like with the loss functions, we also plot the accuracy curves of the politifact and gossipcop datasets. The Fig. 4.8 describes the politifact accuracy curve:

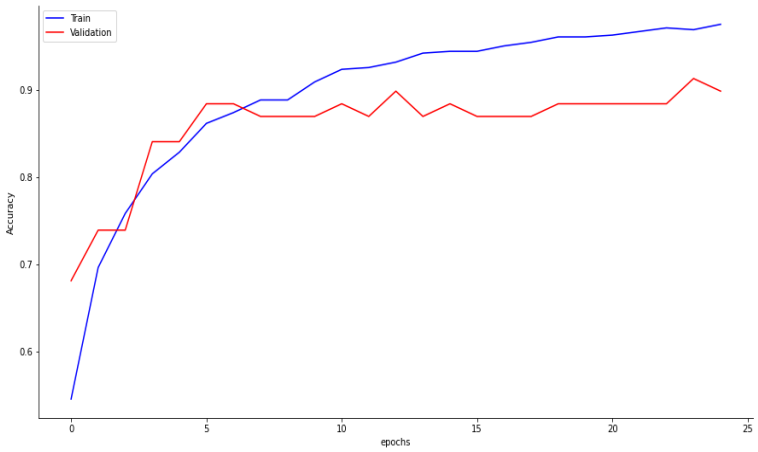

Figure 4.8: Accuracy curve for the homogeneous machine learning model for the politifact dataset

The accuracy increases as the number of epochs increases, for both training and validation set. The generalisation gap is not very high, which means that our model generalises well on unseen data.

In the gossipcop accuracy curve given in the Fig. 4.9, we see that our results for the accuracy are excellent, as both the training and the validation accuracy increases and stabilises around 0.97, which means that 97% of our predictions are well predicted. Also, we can remark that the generalisation gap is extremely low, which is promising for the results we can expect on the test set.

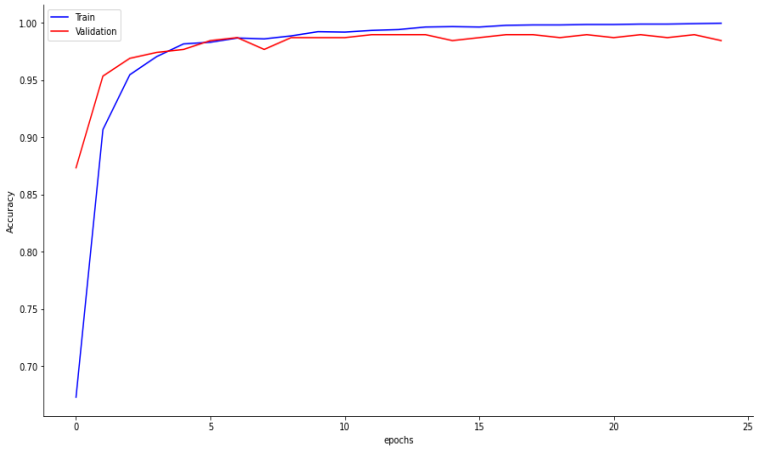

Figure 4.9: Accuracy curve of the homogeneous machine learning model for the gossipcop dataset

Accuracy curve for the heterogeneous graph model for the politifact dataset

For the heterogeneous graph model, the accuracy curve is presented in the Fig. 4.10:

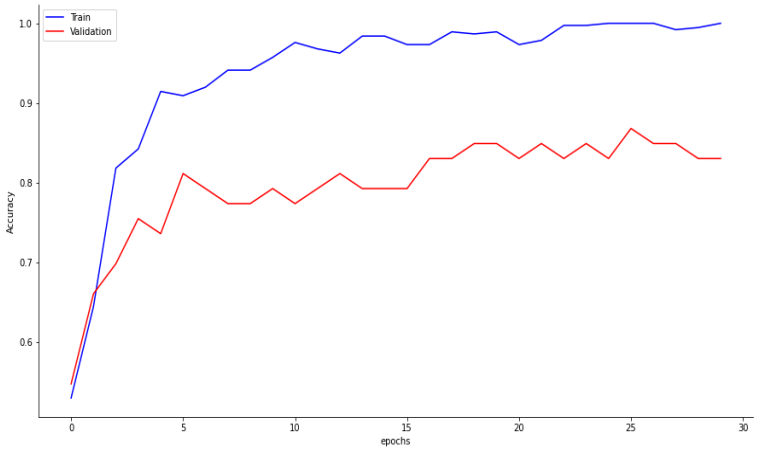

Figure 4.10: Accuracy curve of the heterogeneous machine learning model for the politifact dataset

We can observe that the training accuracy is increasing with the validation accuracy. However, the generalisation gap is a bit high even with the different regularisation techniques used such as the dropout layer and the weight decay. One approach that could decrease the gap is adding more data for the training.

For the gossipcop dataset, the accuracy curve 4.11 plotted shows good results, like with the homogeneous graph. We have also a low generalisation gap, due to the higher number of training samples than with the politifact dataset.

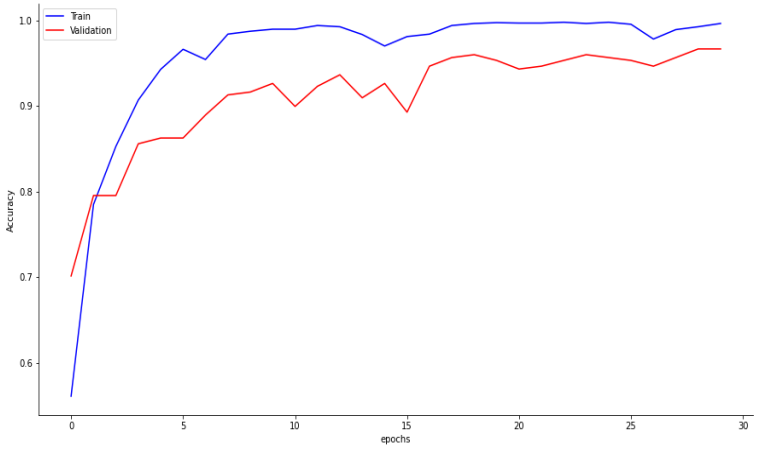

Figure 4.11: Accuracy curve of the heterogeneous machine learning model for the gossipcop dataset

Results on the test set

The results on the set are the most important results; these results guarantee that our model can classify news and is ready to go to production. The model has been evaluated on different metrics: the accuracy, the f1 score, the precision and the recall. We will first present the different metrics calculated on the test set for each dataset and each model. We will then compare the performance of the model with the results found by the UPFD paper, in the section 5.1.1.

The metrics evaluated on the test set for the homogeneous graph machine learning model are described in the table 4.1:

PolitiFact				**Gossipcop**			
F1-score	Accuracy	Recall	Precision	F1-score	Accuracy	Recall	Precision
92.44	92.81	90.29	93.03	99.60	99.61	99.79	99.57

Table 4.1: Project performance of the homogeneous graph model on the test set for the two datasets

We found excellent results on both datasets, with notably a significant F1-score, which is often the most important metric to consider when evaluating a model.

The metrics evaluated on the test set for the heterogeneous graph machine learning model are described in the table 4.2:

PolitiFact				**Gossipcop**			
F1-score	Accuracy	Recall	Precision	F1-score	Accuracy	Recall	Precision
91.31	91.67	85.37	97.04	97.56	97.66	97.51	97.29

Table 4.2: Project performance of the heterogeneous graph model on the test set for the two datasets

We found also good results on both datasets with the heterogeneous graph machine learning model. We have a very solid F1-score and a good accuracy, but we see that the recall is a bit lower on the politifact dataset, which means that there are some news that we consider as fake while there are true.

We can see also that for the Gossipcop dataset, we have for both datasets better results. This is mainly due to the fact that we have way more gossip news, so our neural network is trained on more examples and has a better learning.

Chapter 5

Critical Evaluation

5.1 Analysis of the results

5.1.1 Comparison with the baseline

Overall, the pipeline returns very good predictions that outperform the UPFD framework. The comparison of the performance with the UPFD paper on the politifact and the gossipcop dataset is reported in the table 5.1:

	PolitiFact		*Gossipcop*	
	ACC	F1	ACC	F1
UPFD	84.62	84.65	97.23	97.22
Heterogeneous model	91.67	91.31	97.66	97.56
Homogeneous model	92.81	92.44	99.61	99.60

Table 5.1: Comparison of the the models performance

For the politifact dataset, the pipeline's results is about 6% better than the UPFD framework using the heterogeneous graph machine learning model and about 7% better in the case of the homogeneous graph machine learning model. Several reasons can explain this difference. First, we collect more data and hence, our neural network has been trained on more graphs than the UPFD framework. Also, despite the model is the same, we tried to optimise as much as possible the hyper-parameters to build the best classifier. Finally, the social context used in both applications differs. In the UPFD framework, the users information is collected using the historical tweets of the user while we used the user's description. In the UPFD framework, the root node of the graph is the news information and all the others nodes represent users information while in this project, the root node is the news and then, the others nodes can be either tweets, retweets or user profiles information.

For the gossipcop dataset, the pipeline's results is about 0.34% better using the heterogeneous graphs and about 2% better using the homogeneous graphs. This is a great result because with lesser data than the UPFD framework, we achieve better or similar performance; for the homogeneous dataset, we have 3876 graphs and for the heterogeneous dataset, we have 2988 graphs in the dataset, while in the UPFD paper, they collected 5464 gossip news. This shows that our model is performing very well.

The results are satisfying and hence, we could say that the proposed system outperforms the UPFD model. However, this is complicated to compare which model is the best, as there are not trained with the same graphs. Nevertheless, we can say that the pipeline described in this project allows to build a solid classifier that can differentiate fake and real news.

5.1.2 Heterogeneous Graphs versus Homogeneous Graphs

Having validated our results with respect to the UPFD framework, we still have to compare the results of the homogeneous model with the heterogeneous model. The homogeneous model provides better results for both datasets; the principal reason is that we have lesser data with the heterogeneous graphs. In the case of heterogeneous graphs, because we have added image features, our input dimension is very high and is prone to over-fitting. That is why we had to add regularisation methods for the heterogeneous graphs like the dropout layers and the weight decay, to reduce the complexity of the model. Adding more data could increase the performance of the model, as it allows to use more complex models and hence, reduces the variance of the model.

5.2 Limitations

Despite the pipeline provides solid results on fake news classification, there are some limitations that can be raised:

- One of the limitations of this model is the data. Indeed, the data is multi-modal, hard to collect and process. Despite data is more and more accessible, it is still important to take in consideration than the less data is needed, the more cost-efficient is the model. Here, we have to retrieve both news information and social context related to the news. Moreover, Twitter API is limiting the access to their information and therefore, the data is hard to collect.
- We are using a Deep Learning model, i.e. an end-to-end model. One of the major limitations of Neural Networks model is that we cannot explain wellthe results. It is important to understand how the news are classified and what information is used and valued by the model.
- The diversity of the data is also an issue. We used two datasets in this project, Politifact which is dealing with political news and GossipCop which shares gossip news. In consequence, we can only say that this model is able to predict gossip and political news. There are different type of fake news spread on social media; there are other fields where there are fake news such as in health, industry or sports.
- Tuning the hyper-parameters of such a deep learning model is complicate. There are some optimisation techniques used for hyper-parameters optimisation such as Bayesian optimisation, but this is computationally expensive to find all the hyper-parameters. Thus, the hyper-parameters have been tuned manually.

5.3 Further work

This project could be explored in further details and many approaches could lead to better results and a more solid pipeline.

5.3.1 Adding more modalities

One of the important points of our project is that it is multi-modal. The data can be either social context, text and images. Using different types of information can increase the performance of the model as different data types can bring different information and thus help making a better prediction. In the FakeNewsNet dataset, it is possible to extract videos present in a news article. There are also a lot of tabular data present in the tweets, retweets and user profiles files. For instance, we know if a user is verified user, i.e., a user that has been authenticated by Twitter to certificate that this is account is the true and only account of the user. Verified accounts are often accounts of celebrities or publicly known person.

5.3.2 Different sources of fake news

As explained in the section 5.2, one of the limitations of the model is that our model is trained only with gossip and political news, hence, one further work that is important to explore is checking the performance of the model with other types of fake news. One particular area where fake news are numerous is health, in particular news related to Covid-19. The purpose of this exploration is to build a model that is flexible and can classify news regardless of the topic.

5.3.3 Text-Image concatenation

When we added images features to our text features, we used a simple concatenation method. However, there are other techniques that exist for multi-modalities fusion that can maybe result in a better representation of our features.

5.3.4 Text and Image pre-processing

In this project, we use the BERT and VGG-19 pre-trained models to extract features from text and images. Using these architectures provides good results, however, we also introduced the Text-CNN architecture [21], which can extract features from both text and images. It could be a good option for our heterogeneous model, as we combine features from both text and images. Indeed, the neural network has better chance to understand features that have been created from a similar architecture [50].

5.3.5 Hyper-parameters tuning

We explained before why we could not dedicate a whole part to hyper-parameter optimisation, as it can be computationally expensive, however, it could be interesting to explore the different optimisation techniques that could lead to better results for the predictions.

5.4 Assessment with regards to objectives

The main objective of the project was to build a pipeline for fake news detection, starting with collecting fake news, processing, modelling and evaluating the performance of the model. Thus, we propose a system that outperforms the results found by state-of-the-art models, such as the UPFD [11] framework, which is the system proposing the best performance to the best of my knowledge - this objective has been correctly respected and achieved.

Moreover, we wanted our system to extract information from social context, textual and image information from news article. To the best of my knowledge, there is no fact-checking model that classify news using all these different modalities. So, we succeeded to build a heterogeneous graph machine learning model that can classify news using all these different modalities which outperforms the UPFD framework.

Finally, one of the aims of the project was to build a system that could also use videos as a source of information for fake news detection. However, because the collection of the datasets took a lot of time and processing videos is costly in computation time, it was hard to include this part in the allotted time.

Chapter 6

Conclusion

This project consists in building a pipeline for fake news detection. The pipeline starts with data colection of the FakeNewsNet data using the Twitter API [36, 38, 37], which is composed of two datasets: Politifact and GossipCop. Two graphs dataset have been built using the outputs of the pre-trained models, one involving only the text features and the other one combining both text and images features. This leads to a creation of a homogeneous graph dataset in one case and a heterogeneous graph dataset in the second case. The main achievements of the project are the following:

- A Twitter Developer Account provided by the University has been used to collect the social context data from Twitter, using the Twitter API from the beginning, of June to mid-August.
- The data has been then pre-processed using two pre-trained models called BERT and VGG-19, which respectively process text and images data.
- The two models we propose outperform the performance of the state-of-art fact-checking models such as the UPFD [11] and SAFE [50] models. In particular, the homogeneous graph machine learning model's F1-score is about 92.44% on the PolitiFact dataset and about 99.60% for the GossipCop dataset against respectively 84.62% and 97.23% for the UPFD framework.
- To the best of our knowledge, the heterogeneous model built is the first model that uses social context, text and **image** information for fake news detection.

Therefore, the pipeline is working very well and provides valid results on the FakeNewsNet dataset. Thus, the main objectives of the project are fulfilled. However, one of the objectives of the project was to propose a system that can predict fake news using text, images but also videos which was not possible with the allotted time because pre-processing videos is complex and requires a lot of computational time.

Some limitations can be pointed out regarding this pipeline. First, the need of big data with a variety of modalities is required, which is hard to collect using the Twitter API. Also, one issue of this pipeline is that we cannot explain the predictions given by the model, as Deep Learning models are black boxes. Finally, the pipeline designed here is performing well on political and gossip news, but there a lot of different fields where fake-news are propagated.

These limitations are also opportunities for further work - it could be interesting to use this pipeline for different source of fake news such as Covid-19 related news. Also, even if the heterogeneous model already outperforms current state-of-art models, the different experiments suggest that adding data could increase the performance of the model and thus reduce the over-fitting issue we had to handle using regularisation methods. If more data were added, videos integration in the heterogeneous graphs could be the next frontier.

Bibliography

[1] Convolutional neural networks cheatsheet. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks.

[2] Gradient descent and stochastic gradient descent. http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/.

[3] Public health and online misinformation: Challenges and recommendations. https://www.annualreviews.org/doi/10.1146/annurev-publhealth-040119-094127#_i39.

[4] Amirhossein Yazdani Abyaneh, Ali Hosein Gharari Foumani, and Vahid Pourahmadi. Deep neural networks meet csi-based authentication, 2018.

[5] K. Acharya, R. Babu, and Sathish Vadhiyar. A real-time implementation of sift using gpu. *Journal of Real-Time Image Processing*, 2014.

[6] Saahil Afaq and Smitha Rao. Significance of epochs on training a neural network. *International Journal of Scientific & Technology Research*, 2020.

[7] Senior Andrew, Heigold Georg, Ranzato Marc'Aurelio, and Yang Ke. An empirical study of learning rates in deep neural networks for speech recognition. pages 6724–6728, 2013.

[8] Nicolas Audebert, Catherine Herold, Kuider Slimani, and Cédric Vidal. Multimodal deep networks for text and image-based document classification, 2019.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[11] Yingtong Dou, Kai Shu, Congying Xia, Philip S. Yu, and Lichao Sun. User preference-aware fake news detection, 2021.

[12] Sanjay Dwivedi and Bhupesh Rawat. A review paper on data preprocessing: A critical phase in web usage mining process. 2015.

[13] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[14] Peter I. Frazier. A tutorial on bayesian optimization, 2018.

[15] Gnecco Giorgio and Sanguineti Marcello. The weight-decay technique in learning from data: An optimization point of view. *Computational Management Science*, 2009.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017.

[18] Naeemul Hassan, Anil Nayak, Vikas Sable, Chengkai Li, Mark Tremayne, Gensheng Zhang, Fatma Arslan, Josue Caraballo, Damian Jimenez, Siddhant Gawsane, Shohedul Hasan, Minumol Joseph, and Aaditya Kulkarni. Claimbuster: the first-ever end-to-end fact-checking system. *Proceedings of the VLDB Endowment*, 10:1945–1948, 08 2017.

[19] Shun ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, pages 185–196, 1993.

[20] Al-Ansari Khaled. Survey on word embedding techniques in natural language processing. 2020.

[21] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[23] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, page 84–90, May 2017.

[25] Jure Leskovec. Lecture notes in theory of graph neural networks, February 2021.

[26] Jure Leskovec and Rok Sosic. Snap: A general purpose network analysis and graph mining library, 2016.

[27] Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs, 2019.

[28] M. Madiajagan and S. Sridhar Raj. *Deep Learning and Parallel Computing Environment for Bioengineering Systems*. Academic Press, 2019.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[30] A. Nikhath and K. Subrahmanyam. Feature selection, optimization and clustering strategies of text documents. *International Journal of Electrical and Computer Engineering (IJECE)*, 9:1313, 04 2019.

[31] Parth Patwa, Shivam Sharma, Srinivas Pykl, Vineeth Guptha, Gitanjali Kumari, Md Shad Akhtar, Asif Ekbal, Amitava Das, and Tanmoy Chakraborty. Fighting an infodemic: Covid-19 fake news dataset. 2020.

[32] Abraham Pouliakis, Effrosyni Karakitsou, Niki Margari, Panagiotis Bountris, Maria Haritou, John Panayiotides, Dimitrios Koutsouris, and Petros Karakitsos. Artificial neural networks as decision support tools in cytopathology: Past present and future. *Biomed Eng Comput Biol*, 2016:1–18, 02 2016.

[33] Xin Qian and Diego Klabjan. The impact of the mini-batch size on the variance of gradients in stochastic gradient descent, 2020.

[34] Z. Reitermanová. Data splitting. 2010.

[35] Szeliski Richard. *Computer vision algorithms and applications*. Springer, 2011.

[36] Kai Shu, Deepak Mahudeswaran, Suhang Wang, Dongwon Lee, and Huan Liu. Fakenewsnet: A data repository with news content, social context and spatialtemporal information for studying fake news on social media, 2018.

[37] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

[38] Kai Shu, Suhang Wang, and Huan Liu. Exploiting tri-relationship for fake news detection. *arXiv preprint arXiv:1712.07709*, 2017.

[39] Natale Simone. To believe in siri: A critical analysis of ai voice assistants. pages 1–17, 2020.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[41] Saatviga Sudhahar, Giuseppe Veltri, and Nello Cristianini. Automated analysis of the us presidential elections using big data and network analysis. *Big Data and Society*, 2:1–28, 04 2015.

[42] van der Maaten Laurens and Hinton Geoffrey. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[44] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[45] Weiyao Wang, Du Tran, and Matt Feiszli. What makes training multi-modal classification networks hard?, 2019.

[46] William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection, 2017.

[47] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[48] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[49] Joseph Yacim and Douw Boshoff. Impact of artificial neural networks training algorithms on accurate prediction of property values. *Journal of Real Estate Research*, 40:375–418, 11 2018.

[50] Xinyi Zhou, Jindi Wu, and Reza Zafarani. Safe: Similarity-aware multi-modal fake news detection, 2020.

Appendix A

Example tweet data in json format


```
{
    "created_at": "Fri Oct 28 02:45:50 +0000 2011",
    "id": 117067116071523335, "id_str": "117067116071523335",
    "text": "X Has Fighting Words for Y Over Z",
    "truncated": false,
    "entities":
        {
        "urls": [{
            "url": "http://t.co/tafa18rer",
            "expanded_url": "http://bit.ly/n1nzdoIr",
            "display_url": "bit.ly/n1noazr",
            "indices": [112, 136]}]
        },
    "user": {
        "id": 172999513,
        "id_str": "172999513",
        "name": "username",
        "screen_name": "userscreenname",
        "location": "New York",
        "description": "Love Music",
        "url": null,
        "entities": {
            "description": {
                "urls": []
                }
            },
        "protected": false,
        "followers_count": 212,
        "friends_count": 364,
        "listed_count": 3,
        "created_at": "Sat Jul 27 04:20:06 +0000 2010",
        "geo_enabled": false,
        "verified": false,
        "statuses_count": 23345,
        "lang": null,
        },
    "is_quote_status": false,
    "retweet_count": 0,
    "favorite_count": 0,
    "lang": "en"
}
```


Listing A.1: Example Tweet data in json format

Appendix B

Architecture of the repository

```
gossipcop
|--- fake
|    |--- gossipcop-1
|    |    |--- news content.json
|    |    |--- tweets
|    |    |    |--- 886941526458347521.json
|    |    |    |--- 887096424105627648.json
|    |    |    |--- ....
|    |    |---retweets
|    |         |--- 887096424105627648.json
|    |         |--- 887096424105627648.json
|    |         |--- ....
|    |--- ....
|--- real
|    |--- gossipcop-1
|    |    |--- news content.json
|    |    |--- tweets
|    |    |--- retweets
|    |--- ....
politifact
|--- fake
|    |--- politifact-1
|    |    |--- news content.json
|    |    |--- tweets
|    |    |--- retweets
|    |--- ....
|--- real
|    |--- poliifact-2
|    |   |--- news content.json
|    |   |--- tweets
|    |   |--- retweets
|    |--- ....
|--- user_profiles
|   |--- 374136824.json
|   |--- 937649414600101889.json
|   |--- ....
|--- user_timeline_tweets
|   |--- 374136824.json
|   |--- 937649414600101889.json
|   |--- ....
|--- user_followers
|   |--- 374136824.json
|   |--- 937649414600101889.json
```

```
|   |--- ....
|--- user_following
    |--- 374136824.json
    |--- 937649414600101889.json
    |--- ....
```

Listing B.1: architecture of the repository of the downloaded dataset

Appendix C

Architecture of the heterogeneous model

```
HeteroClassifier(
  (rgcn): RGCN(
    (conv1): HeteroGraphConv(
      (mods): ModuleDict(
        (mentionned by): SAGEConv(
          (feat_drop): Dropout(p=0.0, inplace=False)
          (fc_pool): Linear(in_features=4864, out_features=4864, bias=True)
          (fc_self): Linear(in_features=768, out_features=128, bias=False)
          (fc_neigh): Linear(in_features=4864, out_features=128, bias=False)
        )
        (spread): SAGEConv(
          (feat_drop): Dropout(p=0.0, inplace=False)
          (fc_pool): Linear(in_features=768, out_features=768, bias=True)
          (fc_self): Linear(in_features=768, out_features=128, bias=False)
          (fc_neigh): Linear(in_features=768, out_features=128, bias=False)
        )
        (mentionned): SAGEConv(
          (feat_drop): Dropout(p=0.0, inplace=False)
          (fc_pool): Linear(in_features=768, out_features=768, bias=True)
          (fc_self): Linear(in_features=4864, out_features=128, bias=False)
          (fc_neigh): Linear(in_features=768, out_features=128, bias=False)
        )
        (spread by): SAGEConv(
          (feat_drop): Dropout(p=0.0, inplace=False)
          (fc_pool): Linear(in_features=768, out_features=768, bias=True)
          (fc_self): Linear(in_features=768, out_features=128, bias=False)
          (fc_neigh): Linear(in_features=768, out_features=128, bias=False)
        )
        (tweeted by): SAGEConv(
          (feat_drop): Dropout(p=0.0, inplace=False)
          (fc_pool): Linear(in_features=768, out_features=768, bias=True)
          (fc_self): Linear(in_features=768, out_features=128, bias=False)
          (fc_neigh): Linear(in_features=768, out_features=128, bias=False)
        )
        (tweeted): SAGEConv(
          (feat_drop): Dropout(p=0.0, inplace=False)
          (fc_pool): Linear(in_features=768, out_features=768, bias=True)
          (fc_self): Linear(in_features=768, out_features=128, bias=False)
          (fc_neigh): Linear(in_features=768, out_features=128, bias=False)
        )
      )
```

```
    )
  (lin1): Linear(in_features=128, out_features=128, bias=True)
  (lin2): Linear(in_features=4864, out_features=128, bias=True)
  (lin3): Linear(in_features=256, out_features=128, bias=True)
  (classify): Linear(in_features=128, out_features=2, bias=True)
)
```

Listing C.1: Architecture of the heterogeneous model

Appendix D

Hyper-parameters of the model

For the homogeneous model, the hyper-parameters chosen are presented in the table D.1

	SEED	BATCH SIZE	LEARNING RATE	WEIGHT DECAY	HIDDEN SIZE	EPOCHS	DROP RATIO
POLITIFACT	777	64	0.0001	0.0	128	25	0.0
GOSSIPCOP	777	64	0.0001	0.0	128	25	0.0

Table D.1: Hyper-parameters of the homogeneous model

For the heterogeneous model, the hyper-parameters chosen are presented in the table D.2

	SEED	BATCH SIZE	LEARNING RATE	WEIGHT DECAY	HIDDEN SIZE	EPOCHS	DROP RATIO
POLITIFACT	777	32	0.0001	0.001	32	30	0.5
GOSSIPCOP	777	64	0.0001	0.01	128	30	0.5

Table D.2: Hyper-parameters of the heterogeneous model