
**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

Programa de posgrado en Ciencia e Ingeniería
de la Computación

# Graph-based Siamese Network applied to the Authorship Verification task

# T E S I S

QUE PARA OPTAR POR EL GRADO DE:

**MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN**

P R E S E N T A:

**DANIEL EMBARCADERO RUIZ**

T U T O R A:

**DRA HELENA M. GÓMEZ ADORNO**
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS, UNAM

Ciudad Universitaria, CD. MX. Enero, 2022

Jurado asignado:

| | |
|---|---|
| Presidente: | Dr. Ángel Fernando Kuri Morales |
| Vocal: | Dra. Helena M. Gómez Adorno |
| Secretario: | Dr. Gibrán Fuentes Pineda |
| Suplente: | Dra. Gemma Bel Enguix |
| Suplente: | Dr. Grigori Sidorov |

Directora de tesis:

**Dra. Helena M. Gómez Adorno**

**Acknowledgments**

This research was partially funded by CONACYT PNPC scholarship with No. CVU 1004062, DGAPA-UNAM PAPIIT grant number TA100520, DGAPA-UNAM PAPIIT grant number TA400121 and CONACYT CB A1-S-27780.

The authors also thank CONACYT for the computer resources provided through the INAOE Supercomputing Laboratory's Deep Learning Platform for Language Technologies.

# Agradecimientos

*Ciudad Universitaria, IIMAS, 2022*

Me gustaría agradecer a mi hermano Alberto Embarcadero, por enseñarme las bases sobre las que pude desarrollar todo este trabajo y siempre ser mi respaldo cuando surgen contratiempos; a mi tutora por los inumerables consejos que me ha dado y a mis padres por apoyarme en todos los proyectos que emprendo.

# Contents

# List of Figures

# List of Tables

**Título**

Graph-based Siamese Network applied to the Authorship Verification task

**Daniel Embarcadero Ruiz**
Posgrado en Ciencia e Ingeniería de la Computación
Universidad Nacional Autónoma de México

## Resumen

En este trabajo proponemos un nuevo enfoque para resolver la tarea de verificación de autoría en un escenario donde los textos a evaluar pertenecen a temas distintos y pueden ser de autores nunca vistos durante el entrenamiento. La tarea de verificación de autoría consiste en determinar si dos textos fueron escritos por el mismo autor.

Proponemos representar los documentos como grafos y utilizar una arquitectura de red neuronal directamente sobre estos grafos para extraer las características relevantes. Presentamos tres estrategias para representar los documentos como grafos, basadas en las relaciones de las etiquetas POS y la coocurrencia de las palabras. En nuestro modelo empleamos una arquitectura Siamesa compuesta de capas convolucionales sobre grafos, capas *pooling* y capas de clasificación. Presentamos algunas variaciones de esta arquitectura y evaluamos el desempeño de cada una de estas.

Para evaluar nuestra propuesta utilizamos el conjunto de datos compilado para la tarea compartida de Verificación de Autoría en PAN@CLEF 2021; dicho conjunto de datos se presenta en dos versiones: un conjunto de datos pequeño y uno grande. Nuestra propuesta obtuvo en promedio (AUC ROC, F1, Brier score, F0.5u y C@1) resultados entre 90% y 92.83% para los modelos entrenados en el conjunto de datos pequeño y grande, respectivamente. Estos resultados hacen que nuestro modelo sea comparable con los métodos conocidos por el estado del arte y superior a los baselines tradicionales en esta tarea.

**Title**

Graph-based Siamese Network applied to the Authorship Verification task

**Daniel Embarcadero Ruiz**
Posgrado en Ciencia e Ingeniería de la Computación
Universidad Nacional Autónoma de México

## Summary

In this work, we propose a novel approach to solve the Authorship Verification task on a cross-topic and open-set scenario. Authorship verification is the task of determining whether or not two texts were written by the same author.

We model the documents in a graph representation and then a graph neural network extracts relevant features from these graph representations. We present three strategies to represent the texts as graphs, based on the relation of the POS labels and co-occurrence of the words. We propose a Siamese Network architecture composed of graph convolutional networks along with pooling and classification layers for the automatic verification process. We present different variants of the architecture and discuss the performance of each one.

To evaluate our approach we used a collection of fanfiction texts provided by the PAN@CLEF 2021 shared task in two settings: a small corpus and a large corpus. Our graph-based approach achieved average scores (AUC ROC, F1, Brier score, F0.5u, and C@1) between 90% and 92.83% when training on the small and large corpus, respectively. We show that these results are comparable to those of the state of the art and greater than traditional baselines in this task.

# Chapter 1

# Introduction

Authorship analysis aims to identify characteristics of an author's writing style given a text sample, and ultimately be able to identify the author himself/herself. The idea behind this research area is that some features of the documents allow distinguishing texts written by different authors [18].

There are different tasks in authorship analysis, for example, authorship attribution, author profiling, author clustering, and plagiarism detection [42]. The authorship verification task aims to determine if two given texts were written by the same author.

Graph theory allows us to model problems that are difficult to analyze in a traditional structured manner; with this, we can describe relations between objects or individuals that cannot be interpreted as vectors or fixed grids [27]. A graph representation of a text is powerful because it can describe the topological and statistical relations between relevant elements of texts like sections, paragraphs, sentences and words [41]. In recent years, there had been a lot of progress when using deep learning techniques to learn relevant patterns directly from graph structured data [49]. These techniques are commonly recognized under the name of graph neural networks.

## 1.1 Problem definition

The problem to solve is the authorship verification task. In this task, we receive two texts and aim to determine if these were written by the same author or not. We focus our research on an open-set scenario, that is, the model will be evaluated on a dataset that has texts of authors never seen in the training dataset.

## 1.2 Objective

The general objective of this work is to develop a model capable to solve authorship verification task using a graph representation from text and graph neural networks over that representation. We want to define a graph representation from text capable to capture the author style.

The particular objectives of this work are:

- To propose a deep learning architecture working over the graph representations using graph neural networks.

- To make experiments over the proposed representations and architectures to find the best configurations for the task.

## 1.3 Contribution

The main contribution of this work is a novel Siamese network architecture composed of two graph convolutional neural networks, pooling, and classification layers to approach the authorship verification task. We present three strategies (*short*, *med*, and *full*) for representing texts as graphs based on the relation of the POS labels and co-occurrence of the words. Each strategy varies in the complexity of the graph and the computational cost for processing it.

In this work we present several experiments to test our proposed approach. First, we present experiments varying the graph convolutional layers and the number of layers in pooling and classification, for each graph representations used. Later, we propose to combine more than one graph-based representation for the feature extraction in an ensemble architecture. For this ensemble architecture, we present the evaluation of different training strategies. Additionally, we added a stylistic-based feature extraction component to the Siamese Network architecture to improve the performance of the models. Finally, we present a technique for a threshold adjustment that improves the performance of our architectures on the authorship verification task.

## 1.4 Thesis organization

The rest of this work is organized as follows. Chapter 2 presents an overview of the literature on four research areas relevant to our work: Authorship Verification, Graph-based representation of texts, Graph Neural Networks and Siamese Neural Networks. In Chapter 3 we describe our proposed method to solve the Authorship Verification task; the details of the process to model text on graphs is presented in Section 3.1. Chapter 4 includes the description of our metrics, the dataset used, baselines to compare our approach to, and the experimental settings using in our research. Chapter 5 reports and analyzes the results obtained by the different experiments with the proposed architectures and baseline methods; particularly, in Section 5.5 we show how our model is compared with the state of the art methods. Finally, Chapter 6 presents our conclusions and points out to future research lines.

# Chapter 2

# Related work

In this chapter, we present the relevant related work in authorship verification task, graph-based text modeling, Graph Neural Networks, and Siamese Neural Networks. These four topics are used as part of our final model.

## 2.1 Authorship Verification

Authorship analysis aims to identify characteristics of an author's writing style given a text sample, and ultimately be able to identify the author himself. The idea behind this research area is that some features of the documents allow distinguishing texts written by different authors [18]. The formal study of authorship analysis started in the 19th century, it was first tackled with linguistic approaches and eventually by statistical and computational methods [31].

We can distinguish different tasks in the authorship analysis context [42, 44]:

1. *Authorship attribution*, sometimes named authorship identification. This is the task of identifying the author of a given text from a group of possible authors.

2. *Author profiling*, consists in obtaining relevant information about the author, like age, sex, social group, etc.

3. *Author clustering* consists in grouping documents written by the same author so that each cluster corresponds to a different author.

4. *Plagiarism detection*, consists in measuring the similarity of two texts to decide how possible is that one is a copy from the other.

5. ***Authorship verification***. This task aims to determine if two given texts were written by the same author.

6. *Detection of stylistic inconsistencies*, consists in measuring the stylistic changes inside a single document. These inconsistencies may happen in collaborative writing.

These tasks continue growing in attention for their practical applications, for example, in a variety of computer crime investigations ranging from homicide to identity

theft and many types of financial crimes [9] or in the context to identify the author of a source code [15].

Authorship attribution is the most frequently studied task, it can be modeled as a classification problem over a closed set of possible labels (the authors); a natural generalization of this problem is to consider the possibility that the real author does not belong to the known set of authors, giving us a classification problem over an open set. When we consider the open set scenario with just one author in the known set, we get the authorship verification task [45].

In this work, we will study the Authorship Verification task with a computational approach. A relevant known problem when applying machine learning or deep learning techniques to this task is that in practical applications we usually just have a very limited amount of texts for training and these training texts can be of different length, topic, genre, etc. with respect of the test texts [43].

### 2.1.1 Classification approaches

Considering that the authorship verification task can be modeled as the authorship attribution task with an open-set scenario and having only texts from one known author as a reference, we will explain the common classification approaches proposed to solve the authorship attribution task. This general approach has been applied also to other authorship analysis tasks like author profiling and plagiarism detection.

In general, the most popular approach for authorship analysis is to extract features from the texts and use them to train a classification algorithm, which can be based either on supervised learning or similarity measures. We can distinguish the extracted features according to the computational requirements for extracting them [42]:

1. Lexical: Word length, sentence length, bag of words, vocabulary richness, misspelled words, etc...

2. Character-level: character n-grams, character types, count of special characters, compression methods, etc...

3. Syntactic: POS, chunks, sentence and phrase structure, etc...

4. Semantic: Semantic dependencies, synonyms, etc...

5. Specifics: Features dependent of the analysed language; for example, the language structure of HTML.

In the first works about authorship analysis, lexical features were used because of the simplicity, but they have the problem of being usually topic-dependent. Semantic features provide us with a deep representation of the text, but in return are computationally expensive and depend on the accuracy of semantic analysis tools. The syntactic and character features generally achieve good results, the computational cost is usually acceptable.

It is usual to search over all these possible features sets to find the most relevant to obtain the best performance, but these feature selections sometimes are data-dependent and lead to a poor generalization capacity [44].

When considering an authorship analysis task as a classification problem, we can distinguish the training data (the texts for which we know the author) and the test data (the texts for which we do not know the author). There are two approaches with respect to the way we use the training data: the profile-based approach uses all the available texts from the same author as a unique document from where we will learn the author style; the other is the instance-based approach that considers each document as an individual representation of the author's writing style.

Some supervised classification algorithms used in authorship verification tasks are support vector machines, decision trees, discriminant analysis, neural networks, and genetic algorithms [44]. Another option is to calculate the similarity between texts and use it to predict if both texts are written by the same author or not; this is a more natural approach if we take authorship verification as an open set classification problem over a lot of possible authors [24].

### 2.1.2 Unmasking and impostors methods

A specifically designed solution for the authorship verification task is the unmasking method, proposed by [25]. This method tries to measure how deep is the difference between two texts by comparing them several times, each one using less relevant features.

This method has two main stages: first, each pair of texts is transformed into a vector, and last, a classifier is trained over the vectors defined in the first step. To define the vector, we need to obtain the degradation curve for each pair of texts following the next steps:

1. Consider the $n$ most common words of the concatenation of both texts as the vocabulary. Each text is divided into paragraphs of the same length (500 words) and each paragraph is vectorized counting the frequency of the words in the vocabulary.

2. Using 10-fold cross-validation, a support vector machine classifier is trained and tested over the paragraph vectors. Each paragraph is considered a single instance and the classifier aims to predict to which texts each paragraph originally belongs.

3. The support vector machine assigns a coefficient to each word used as a feature; these coefficients represent how relevant this feature is to the classification. The $k$ words with higher coefficients and the $k$ words with lower coefficients are removed from the vocabulary.

4. These steps are repeated $i$ times, each one with the paragraph vectors just considering the remaining vocabulary. For each iteration, the accuracy of the support vector machine classifier is recorded.

From the previous steps, we get the list of the accuracy obtained in each iteration that is named degradation curve. To obtain the final vector representation of the pair of texts, the authors propose to use this accuracy alongside other features like the first order and second order derivatives. The proposed features are:

- Accuracy after $i$ elimination rounds.

- Accuracy difference between round $i$ and $i + 1$.

- Accuracy difference between round $i$ and $i + 2$.
- $i$ th highest accuracy drop in one iteration.
- $i$ th highest accuracy drop in two iterations.

Finally, a support vector machine classifier is trained with these vectors, trying to predict if the pair is from the same or different authors.

This method achieved an accuracy of 95.7% when tested in datasets of long texts of about 500K words. When trying to apply it to datasets of shorter texts or complaining texts of a different genre and topics the performance decrease considerably. For example, Kestemont et al. [19] evaluates the performance of the unmasking method over a dataset of theatrical and prose texts of about 10,000 words obtaining accuracy of just 77%.

Bevendorff et al. [3] proposed an alternative unmasking method that obtains accuracy between 75% and 80% in short texts of about 4,000 words. The performance of the original unmasking method hinges on the availability of sufficiently many paragraphs per text, each one of enough length. If the paragraphs are too short the training data becomes too sparse and no descriptive curves can be generated. The authors exploit the bag-of-words nature of the unmasking features and create the paragraphs by oversampling words in a bootstrap aggregating manner. They treat each text as a random pool of words from which we can draw without replacement to fill up a chunk. Once the pool is exhausted, they replenish it and draw again until we have generated a sufficient number of paragraphs.

Another relevant strategy to track the authorship verification task is the impostors' method [24, 26]. This method proposes to use a set of impostor documents collected from an outsource (usually the web) in a way that these impostor documents have a similar topic with the known and unknown document. A set of features are also defined, some of the features included in this set are function words, word n-grams, and character n-grams.

The method starts with a known document $X$, an unknown document $Y$, a set of impostor documents $S$, and a set of features $F$. The output will be a label telling us whether if both documents are from the same author or a different author. The process to define this label, as described by Seidman [40], is:

1. Set $score = 0$.
2. Repeat $k$ times:
   (a) Randomly choose a fraction $rate\%$ of features from $F$.
   (b) Randomly choose $n$ impostors from $S$: $I_1$, ..., $I_n$.
   (c) Update the score based on the similarity obtained between the documents: $score = score + \frac{1}{k}$ if $Sim(X, Y) * Sim(Y, X) > Sim(X, I_i) * Sim(Y, I_i)$, for each $i \in \{1, ..., n\}$.
3. Return same-author label if $score > \Delta$; else return diff-author.

Where $rate\%$ is a fixed fraction between 0 and 1, $Sim(A, B)$ is the similarity of document $A$ with respect to $B$ and $\Delta$ is the threshold for the binary classification.

### 2.1.3 Recent approaches

Before 2015, the authorship verification task was mainly evaluated using datasets on training and testing documents that shared the same topic and same genre. It has been observed that in a cross-topic scenario the performance of the traditional approaches decreases [45].

Stamatatos [43] showed that the character n-grams are more reliable than other lexical features for the authorship analysis task when the genre and topic of the documents vary. Sapkota et al. [39] claimed that some kind of n-grams work better than others as features for classification. Particularly some trigrams that include punctuation marks work better in a cross-topic scenario.

The use of a heterogeneous classifier that combines independent classifiers each one with a different approach has achieved good results, usually better than the ones obtained using a single classifier [44, 45]. PAN is a series of scientific events and shared tasks on digital text forensics and stylometry (https://pan.webis.de/, accessed on 21 July 2020). Since 2011 it has focused on authorship analysis tasks and the methods presented in their workshops are the state of the art in this research area [35].

In recent years, neural network architectures have been proposed to solve the task. For example, Bagnall [2] proposed a recurrent neural network architecture over characters to solve the authorship verification task in the PAN 2015 dataset. This approach achieved considerably better results than other approaches over the same dataset but it is computationally more expensive.

Jafariakinabad et al. [17] introduced a syntactic recurrent neural network to encode the syntactic patterns of a document in a hierarchical structure. The model first learns the syntactic representation of sentences from the sequence of POS tags. Their experimental results on the PAN 2012 dataset for the authorship attribution task show that the syntactic recurrent neural network (working over POS tag embeddings) outperforms the lexical model (working over word embeddings) with an identical architecture by approximately 14% in terms of accuracy.

Weerasinghe and Greenstadt [47] propose a model with notable performance using a stylometric approach. They extracted stylometric features from each text pair and used the absolute differences between the feature vectors as input to a classifier. They evaluated one model using a logistic regression classifier and another using a neural network approach. The features that they propose are character and POS n-grams, special characters, frequency of function words, number of characters, number of words, word-lengths, vocabulary richness, POS-tag chunks, and noun and verbal phrases construction. The authors improve their model for the Authorship Verification task on an open-set scenario [48]. They propose to use stop-words and POS tag hybrid tri-grams, POS ratios, and unique spellings in addition to the previously proposed stylistic features. These stylistic approaches obtain the second and the third best places in the Authorship Verification task of 2020 and 2021, respectively.

## 2.2 Graph-based representation of texts

A common and standard approach to model text documents is bag-of-words. This model is suitable for capturing word frequency; however structural and semantic information is ignored. Graph representation is a mathematical construct and can model

relationships and structural information effectively. A text can be appropriately represented as a graph using feature term as vertices and significant relations between the feature terms as edges. [41].

There are several graph-based representations used to model documents. The general approach consists of identifying relevant elements in the text - i.e. words, sentences, paragraphs, etc. - and considering them as nodes in the graph. Then meaningful relations between these elements are considered as edges. Traditionally, the elements used as nodes in the graph are words, sentences, paragraphs, documents, and concepts. To define the edges, usually syntactic, semantic relations, and statistical counts are used [41].

Some relevant graph-based representations proposed for a variety of authorship analysis tasks are: co-ocurrence graph, co-occurrence based on POS, semantic graph and hierarchical keyword graph [41]. Pinto et al. [36] present another graph-based approach for document understanding. In this work, the authors propose to represent texts as Integrated Syntactic Graphs (ISG). This approach considers different levels of linguistic analysis, such as lexical, morphological, syntactical, and semantic, to build a graph representation of a given document. The authors also introduce a technique for extracting useful text patterns based on shortest paths.

### 2.2.1 Graph based representations applied to Authorship analysis

Castillo et al. [7] present an overview of different graph-based representations proposed to solve authorship analysis tasks. The core of this manuscript is to highlight the importance of enriched/non-enriched co-occurrence graphs as an alternative to traditional feature representation models like vector representation. They propose a directed graph based on star topology for the Authorship Attribution task, an enriched co-occurrence directed graph for the authorship verification task, and a non-directed graph representation based on a co-occurrence graph for the Author Profiling task.

Particularly in [8], the authors applied the proposed graph-based approach on the English language partitions of the CLEF PAN 2014 and 2015 Authorship Verification datasets, producing competitive results that outperform the baseline. They propose to use four centrality measures: closeness, betweenness, degree, and eigenvector to detect relevant patterns of an author's writing style. In particular, they found that words with a high closeness which is part of some chunk phrases and words with high betweenness that are included in bigrams and trigrams, contribute in a more effective way to verify document authorship.

Gómez-Adorno et al. [16] generalize the graph-based method described in [36], initially proposed for the extraction of text patterns obtained by shortest path traversal over Integrated Syntactic Graphs (ISG). They build the ISG representation using linguistic features of various levels of language description, which provides relevant information about the texts. In particular, they show that their method discovers patterns that can be used in authorship attribution and authorship verification. The authors include the results of the experiments carried out for authorship attribution and authorship verification.

## 2.3 Graph Neural Networks

The recent success of neural networks has boosted research on pattern recognition and data mining. The success of deep learning in many domains is partially attributed to the rapidly developing computational resources, the availability of big training data, and the effectiveness of deep learning to extract latent representations from Euclidean data [49].

Deep learning effectively captures hidden patterns of Euclidean data. There is an increasing number of applications where data can be naturally represented in the form of graphs. Graph neural networks can help us to generalize the deep learning approach to such kind of data. For example:

- E-commence: A graph-based learning system can exploit the interactions between users and products to make recommendations.
- Chemistry: Molecules are modeled as graphs, and their bioactivity needs to be identified for drug discovery.
- Citation Network: Papers are linked to each other via citations and they need to be categorized into different groups.

The complexity of graph data has imposed significant challenges on existing machine learning algorithms. Graphs can be irregular, may have a variable size of unordered nodes, and each one of these nodes may have a different number of neighbors.

Wu et al. [49] categorize Graph Neural Networks (GNNs) into:

- Recurrent Graph Neural Networks (RecGNNs) mostly are pioneer works of Graph Neural Networks. RecGNNs aim to learn node representations with recurrent neural architectures. They assume a node in a graph constantly exchanges information/messages with its neighbors until a stable equilibrium is reached.
- Convolutional Graph Neural Networks (ConvGNNs) generalize the convolution operation from grid data to graph data. The main idea is to generate a node representation by aggregating its features and neighbors' features.
- Graph autoencoders (GAEs) are unsupervised learning frameworks that encode nodes/graphs into a latent vector space and reconstruct graph data from the encoded information. GAEs are used to learn network embeddings and graph generative distributions. For network embedding, GAEs learn latent node representations through reconstructing graph structural information such as the graph adjacency matrix.
- Spatial-temporal Graph Neural Networks (STGNNs) aims to learn hidden patterns from spatial-temporal graphs, which become increasingly important in a variety of applications such as traffic speed forecasting, driver maneuver anticipation, and human action recognition.

### 2.3.1 Graph Neural Networks applied to Authorship analysis

We found just a couple of works that applied Graph Neural Networks techniques to an Authorship analysis task.

For instance, Cruz [11] evaluate the robustness of three different approaches for the Authorship Attribution task when the length of the text is varied from 2,500 to 20,000 tokens. The first two tested approaches are a global strategy and a local strategy based on complex network measurements. The third approach tested is a graph embedding technique based on a skip-gram model called Graph2Vec. They found that the local strategy with linear discriminant analysis achieves 97% of accuracy and conclude that it is better to describe short texts than Graph2Vec with low dimension. However, the performance on local and global strategies decreases while the length of each text is increasing, in contrast to the graph embedding technique, which has better results with the large length of texts. A relevant note about these results is that they remove all punctuation marks, remove language stop-words and use lemmatization.

In [29], the authors consider a task from traditional literary criticism: annotating a structured, composite document with information about its sources, and approach it as an Authorship Attribution task. They take the Documentary Hypothesis, a prominent theory regarding the composition of the first five books of the Hebrew bible, extract stylistic features designed to avoid bias or overfitting and train several classification models over the source labels assigned in each part of the texts. Their main result is that the graph convolutional network architecture outperforms structurally uninformed models.

## 2.4 Siamese Neural Networks

Siamese Neural Networks (SNN) was first presented by Bromley et al. [6] to solve the problem of signature verification. Their formulation defines two identical sub-networks, each one acting on an input pattern to extract features. The key idea of their formulation is that the two sub-networks share their weights, that means that both sub-networks must extract the features exactly in the same way. They use the cosine of the angle between the two feature vectors obtained by the sub-networks to assign a distance between the compared instances. The idea is that the siamese network learn how to extract feature vectors from the instances in a way these vectors are close if the instances are similar and these vectors are far if not.

SNNs are in general computationally expensive but perform better as compared to other techniques when learning similarity [33].

Koch et al. [23] proposed a Siamese convolutional network to solve face verification, they emphasize the application of their architecture to manage the one-shot recognition problem. Also, instead of using a contrastive loss as the objective loss, they use a fully connected layer followed by a sigmoid function on top of the Siamese network to get a prediction and optimize a cross-entropy loss with $l_2$ normalization.

### 2.4.1 Siamese Neural Networks applied to Authorship analysis

Recently, SNNs have been proposed to solve the authorship verification task. For instance, two approaches used SNNs to solve the PAN Authorship 2020 Verification

Task.

Boenninghoff et al. [4] used a Siamese architecture of LSTMs with attention coefficients to extract first sentence embeddings and after that a document embedding. This Siamese network aims to learn a pseudo-metric that maps a document of variable length onto a fixed-sized feature vector. At the top, they also incorporate a probabilistic layer to perform Bayes factor scoring in the learned metric space. In their model, they incorporate some text preprocessing strategies like topic masking, using a sliding window to generate the sentences, and data augmentation. Their proposed method achieved excellent overall performance scores, outperforming all other systems that participated in the PAN 2020 Authorship Verification Task, in both the small and "large" datasets.

Araujo-Pino et al. [1] used a Siamese neural network approach that receives as input the character n-gram representation (with n varying from 1 to 3) of the document pairs to be compared.

## 2.5 Summary

We can conclude that the Author Verification task had been studied as a relevant task since the start of the Authorship Analysis. The most used strategies to track this task had been to extract stylometric features and apply a classification algorithm over them. Also, there are relevant ad-hoc methods such as the Unmasking and the Impostors methods.

With respect to the graph-based representation of texts, we can conclude that the graph structure is a natural way to take advantage of the structural information contained in the text. These approaches had shown to be flexible and capable to capture lexical, morphological, syntactic, and semantic information that usually are underused in the classic vector representations such as bag of words.

The recent success of neural networks, partially attributed to the rapid development of computational resources and the availability of a big amount of training data, have motivated the research of new techniques capable to be applied directly to graph-structured data. In the literature, we found a lot of different Graph Neural Networks approaches proposed to tasks such as node classification, graph classification, node clustering, edge prediction, etc. applied to E-commerce, chemistry, and social network analysis.

The Siamese Neural Network had been proposed to track several verification-related tasks. We found applications in the image domain that emphasize the capability of these architectures to generalize verification over classes never seen before. Recently, we found two relevant approaches used for the Authorship Verification task. Both approaches model the texts in a sequential manner.

The graph-based representation of texts had show good results in the Authorship Analysis task. We notice that the known graph-based approaches usually introduce particular techniques to extract relevant vector features, that are dependent on the task. It is known that the Neural Network approach is a way to allow a model to learn relevant patterns from the data instead of proposing hand-crafted methods to extract features.

Even with the development of Graph Neural Networks techniques, we found just a couple of works that applied these techniques to an Authorship analysis task and none

of the works we know applied this technique to the Authorship Verification task. We consider that using Graph Neural Networks to extract relevant patterns automatically as part of a Siamese Neural Network architecture is a natural and powerful way to take advantage of a graph-based representation of texts.

# Chapter 3

# Graph-based Siamese Network for Authorship Verification

In this chapter, we present a novel approach to solve the Authorship Verification task. Our proposed approach the texts into graphs and let graph neural networks extract relevant features from these representations. We use all these ideas in a Siamese Network architecture because it has shown good performance in other verification tasks.

## 3.1 Modeling Texts as Graphs

A core component of our method corresponds to the modeling of the texts as graphs. Our graph representation attempts to capture the relationships between words and POS labels in the text. To make clear our process we use the next text as an example:

`Momo, also known as The Grey Gentlemen or The Men in Grey,`

First, each text is preprocessed as follows:

1. Substitute no ASCII characters to ASCII equivalent. We employed the unidecode package (https://github.com/avian2/unidecode, accessed on 21 July 2021).
2. Tokenize and obtain the POS label.
3. Lowercasing.

We do not remove punctuation of any type, the non-ASCII character substitution was made to reduce the variability of the used punctuation.

To get the POS labels, we considered the PENN-Treebank POS labels [30] obtained by the NLTK package and after that, we add two additional labels: `$PUNCT` to mark all punctuation and `$OTHER` to mark any other word that the NLTK model failed to identify. In total, we consider 38 labels. After this process we obtain a parsed text like this:

```
[('momo', 'NNP'), (',', '$PUNCT'), ('also', 'RB'), ('known', 'VBN'),
 ('as', 'IN'), ('the', 'DT'), ('grey', 'NNP'), ('gentlemen', 'NNPS'),
 ('or', 'CC'), ('the', 'DT'), ('men', 'NNPS'), ('in', 'IN'),
 ('grey', 'NNP'), (',', '$PUNCT')]
```

We define a directed graph as an ordered pair $G = (V, E)$, where $V$ is a set of vertices composed by $(word, pos)$ tuples and $E$ is a set of weighted edges. The edge set $E \subseteq \{(n_1, n_2, w) \mid n_1, n_2 \in V \ , \ n_1 \neq n_2 \ , \text{ and } w \in \mathbb{R}\}$, where $w$ is the edge weight. The construction is based in a co-occurrence graph, as explained in [41] with a different edge weighting; we used the Networkx (https://networkx.org/, accessed on 21 July 2021) package to construct the graphs.

Let $P$ be the parsed text as a list of tuples, $\ell(P)$ the number of elements in the list, and $P[i]$ the i-th element in the list. The co-occurrence graph is constructed as follows. We define a set of POS labels called `REDUCE_LABELS`; for each $P[i] = (word, pos)$ in $P$, we can define:

$$M[i] = \begin{cases} (word, pos) & \text{if pos } \notin \texttt{REDUCE\_LABELS} \\ (pos, pos) & \text{if pos } \in \texttt{REDUCE\_LABELS} \end{cases}$$

Where $M$ is the list defined by the tuples masked as explained. For each pair of tuples $T_1, T_2 \in M$ let be $f(T_1, T_2)$ the number of times $T_1$ is followed by $T_2$ in $M$ and let be $T = l(P) - 1 = l(M) - 1$; note that $T$ is the total number of times a pair of tuples co-occur in $M$.

Now we can define the nodes and edges of our graph:

$$V = \{T \mid T \in M\}$$

Note that $M$ is a list with order and $V$ is just the set of all tuples in $M$. We want to define an edge between any two nodes (tuples) that appear together in the list $M$:

$$E = \{(T_1, T_2, \frac{f(T_1, T_2)}{T}) \mid T_1, T_2 \in M \wedge f(T_1, T_2) > 0\}$$

With this construction we identify all tuples from parsed text having a specific label in `REDUCE_LABELS` as a single node, so the graph structure changes and with it the information abstracted from the text.

In our experiments, we evaluated graphs generated with different `REDUCE_LABELS` sets. From now we will denominate *short graph* to the graph generated using the set of all possible POS labels as `REDUCE_LABELS`, *full graph* to the graph generated using `REDUCE_LABELS` $= \emptyset$ and we will denominate *med graph* to the graph generated using the following set of `REDUCE_LABELS`:

```
REDUCE_LABELS = ['JJ', 'JJR', 'JJS',                # Adjectives
                 'NN', 'NNS', 'NNP', 'NNPS',        # Nouns
                 'RB', 'RBR', 'RBS',                # Adverbs
                 'VB', 'VBD', 'VBG',                # Verbs
                 'VBN','VBP', 'VBZ',                # Verbs
                 'CD',                              # Cardinal numbers
                 'FW',                              # Foreign words
                 'LS',                              # List item marker
                 'SYM',                             # Symbols
                 '$OTHER'                           # Others]
```

The size of the graph depends of the `REDUCE_LABELS` set used in the construction. Continuing with our example, the *short graph*, the *med graph* and the *full graph* are

shown in 3.1(a), 3.1(b) and 3.1(c) respectively; for simplicity we do not draw the edge weights.

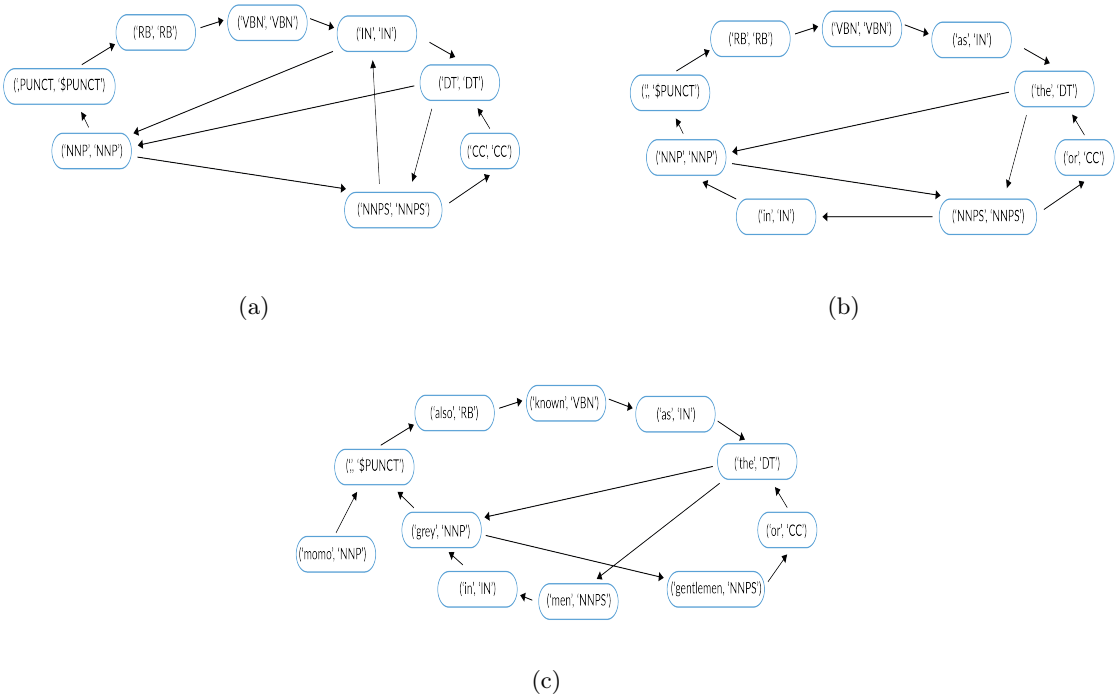



**Figure 3.1:** (a) *Short graph*, (b) *Med graph*, and (c) *Full graph*.

To input our graph to a neural network, we need to encode each node into a vector. To that end, we use a one-hot encoding representation with respect to the 38 possible POS labels used. We do not consider the words in the nodes at this point, only the POS tag information of the node is passed to the neural network. In the final graph, each node is represented by a vector of dimension 38 that encodes their POS label.

## 3.2 Graph-based Siamese Network (GBSN)

To approach the authorship verification task, we use a Siamese network architecture [6] including a component to transform texts as co-occurrence graphs. Our Graph-based Siamese network (Figure 3.2) is composed of two identical feature extraction components with shared weights, a reduction step, and a classification network.

Each feature extraction component receives a text, transforms it into a graph, and returns a vector representation of this graph; the objective is to extract relevant features that can identify the author's style from the graph representation of the texts. We can distinguish three parts in the feature extraction component: graph representation, node embedding layers, and global pooling.

In our architecture, a node embedding layer is composed of a graph convolutional layer, followed by a batch normalization layer, and a ReLU activation function. We will call *conv_type* to the parameter identifying the graph convolutional layer type

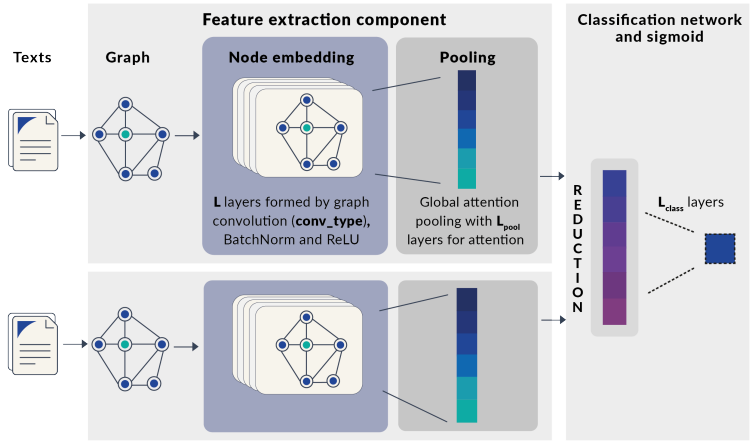



**Figure 3.2:** GBSN base architecture.

and $L$ to the parameter identifying the amount of node embedding layers used by the architecture.

The first node embedding layer takes as input a graph with an initial feature vector in each node; as we described in Section 3.1 each initial node vector has dimension 38 because this vector is a one-hot representation of the POS label of the node. The output of each node embedding layer is the same graph structure with new feature vectors in each node; the dimension of the vectors obtained can be defined in the same way we define the channels used in a traditional convolutional layer. Our architecture obtains vectors of dimension 64 in each convolutional layer.

For the *conv_type* parameter we use the convolutional layers proposed to learn features from nodes, that also take into account the edge weights in their formulation. All the selected layers were used as implemented in PyTorch-geometric (https://pytorch-geometric.readthedocs.io/en/latest/, accessed on 21 July 2021) with the default parameters when no other choice are explicitly described here:

1. GraphConv: A basic implementation of the graph neural network model described by Morris et al. [32].

2. LEConv: Originally proposed by Ranjan et al. [38] to select relevant clusters in a graph, the authors prove that it is more expressive than other layers like the Graph Convolutional Network layer as defined by Kipf and Welling [22] and affirm it can consider both local and global importance of nodes.

3. GCN2Conv: The Graph Convolutional Network via initial residual and identity mapping proposed in [10]. This layer was proposed to solve the over-smoothing problem by including a skip connection to the input layer. We implemented this layer with parameter $\alpha = 0.1$.

4. TAGConv: The Topology Adaptive Graph Convolutional network proposed by Du et al. [13]. This layer is defined in a way that in just one layer it can see the information of not just consecutive nodes but nodes at distance $K$. We performed our experiments with the default $K = 3$.

For the pooling layer, we use the global attention layer, originally proposed by Li et al. [28]. As it is shown in Figure 3.3, this layer takes the final output of the node feature extraction component as its input, that is, a graph with the vector embedding in each node. To obtain the final vector, it makes a weighted sum of each node vector with a coefficient obtained by doing attention over these same vectors, the formulation is:

$$r = \Sigma_{n \in V}\ softmax(h(x_n)) \cdot x_n$$

where $V$ is the set of all nodes in the graphs and $h$ is a fully connected neural network with a single scalar as output. This fully connected neural network has ReLU activation, 32 neurons in each hidden layer, and $L_{pool}$ total layers. The final output of each feature extraction component is a vector with dimension 64.

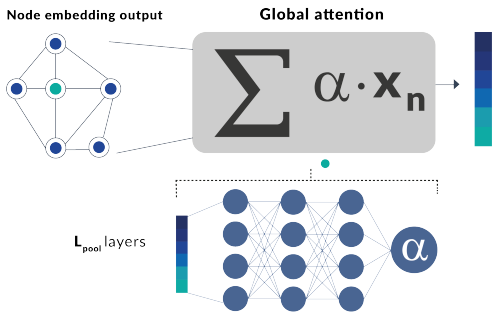



**Figure 3.3:** Global Attention Pooling layer.

For the reduction step, we simply compute the absolute value of the difference between the output of each feature extraction component for each document to be verified. The resulting vector is passed to a final classification network. The classification network is a fully connected network with ReLU activation, 64 neurons in each hidden layer, $L_{class}$ total layers, and a final sigmoid function.

Our model returns a single value in the interval $[0, 1]$ that can be interpreted as a measure of how much the two submitted texts are alike. An output close to 1 tells us that the model finds both texts to be from the same author. To evaluate our models, we use as default a threshold of 0.5, that is, if the output is higher than 0.5 the model says both texts are from the same author, if it is lower than 0.5 both texts are from different authors and if it is exactly 0.5 the model does not answer this problem.

## 3.3 GBSN Ensemble Architecture

We upgrade our base architecture putting side by side different feature extraction components in a new Graph-based Siamese Network Ensemble architecture. Each subnetwork in the Siamese architecture is now formed by several independent feature extraction components, each getting a vector from the text, as shown in Figure 3.4.

We proposed two types of feature extraction components: the first one is a graph-based component as defined in the base architecture; the second one is a stylistic component, based on stylistic features extracted from the texts traditionally. The stylistic component has two parts: stylistic feature representation and embedding network.

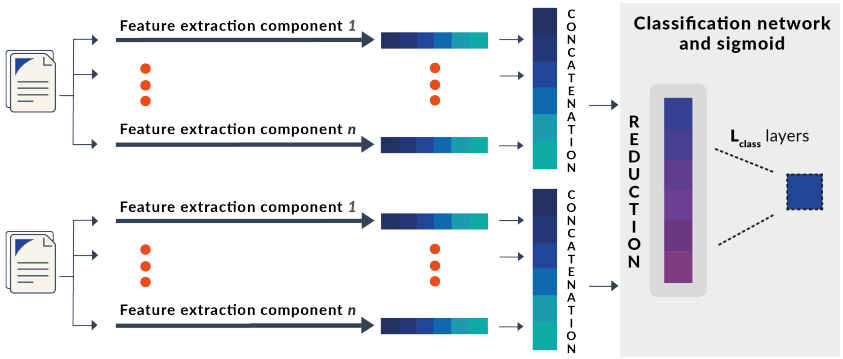



**Figure 3.4:** Graph-based Siamese Network Ensemble architecture.

To obtain the stylistic features from the texts, we use a subset of the features used by Weerasinghe and Greenstadt [47]. For simplicity we only use:

- Frequency of function words (179 function words in the NLTK list)
- Average number of characters per word
- Vocabulary richness
- Frequency of words of $n$ characters (from 1 to 10 characters)

The embedding network in the stylistic component is a fully connected network of just two layers with ReLU activation, 64 neurons in the hidden layer, and 191 neurons in both the input and output layers. The main function of this network is to transform the raw stylistic feature vector into a more meaningful vector before reducing it.

We concatenate the vectors obtained by each component and this one is used in the reduction step. In the reduction step, we compute the absolute value of the difference of the resulting vectors of each document to be verified as in the base architecture.

Finally, the GBSN Ensemble uses a fully connected layer as a classification network defined in the same way that in the base architecture. Note that the GBSN Ensemble has its own $L_{class}$ parameter, independent of any of the feature extraction components used.

## 3.4 Threshold adjustment

Our proposed model was initially trained to return an output between 0 and 1. Given a threshold $th$ and a margin $m$ we can transform the original output $out_o$ to a new one by following the formula:

$$l(out_o) = \begin{cases} \frac{out_o}{2 \cdot (th-m)} & if\ out_o < th - m \\ 0.5 & if\ th - m \leq out_o \leq th + m \\ \frac{out_o - 1}{2 \cdot (1-(th+m))} + 1 & if\ th + m < out_o \end{cases}$$

This formula linearly transforms the interval $[0, th - m)$ into $[0, 0.5)$, the interval $[th - m, th + m]$ into 0.5 and the interval $(th + m, 1]$ into $(0.5, 1]$.

This transformation allows the model to: adjust the best threshold for the binary classification, and predict some difficult problems as 'not answered', i.e., the model avoids the prediction of verification problems when it is not sure if the text corresponds to the same author or not.

## 3.5 Summary

In this chapter, we presented a novel graph-based approach for the Authorship Verification task. We describe our model in four sections: Section 3.1 described our method to model texts as graphs. Section 3.2 described the proposed base model, we also named the model parameters used for our experiments and described in detail the architecture. Section 3.3 described a more general model that allows us to combine more than one feature extraction component in the same architecture. Finally, Section 3.4 described our proposed method to optimize the threshold.

# Chapter 4

# Experimental settings

In this chapter, we describe the experimental settings followed in our experiments. We start describing the metrics used, next we describe the datasets used for training and test, and finally, we describe the baselines methods to compare with our approach.

## 4.1 Metrics

To measure the performance of the proposed models and baselines in the Authorship Verification task we use five different metrics [21]:

- AUC ROC: the ROC area-under-the-curve score.
- F1: The F1 performance measure (not taking into account non-answers).
- Brier score: The complement of the Brier score loss function. [5] as implemented in sklearn (`https://scikit-learn.org/stable/`, accessed on 16 August 2021).
- F0.5u: A newly-proposed F0.5 based measure that emphasizes correctly-answered same-author cases and rewards non-answers [3].
- C@1: A variant of the conventional accuracy measure, which rewards systems that leave difficult problems unanswered [34, 44].

Note that F0.5u and C@1 reward the model to explicitly left difficult problems unanswered rather than providing wrong answers.

The Brier score was meant to measure the probabilistic confidence of the predictions in a more fine-grained manner. This metric rewards models that produce bolder but correct scores (i.e., responses close to 0.0 or 1.0). Conversely, the metric would indirectly penalize less committal solutions, such as non-answers [21].

## 4.2 Datasets

We used the training datasets provided by PAN@CLEF for the Authorship Verification task 2021 [21]. We will refer to these datasets as "small" and "large" according to their size. Table 4.1 shows basic statistics of both datasets; the first column shows many pairs of texts (problems) each dataset has, the other columns show how many different texts, authors, and topics are included in each dataset.

**Table 4.1:** Number of problems, texts, authors and topics in the "small" and "large" datasets.

| | Problems | Texts | Authors | Topics |
|---|---|---|---|---|
| "Small" | 52,601 | 93,662 | 52,654 | 1,600 |
| "Large" | 275,565 | 494,226 | 278,162 | 1,600 |

Each problem is composed of two fanfiction texts; fanfiction is a story originally written by a fan that extends the universe of a fandom topic. All texts are written in English, which have an average of 21,400 characters, 4,950 tokens and 345 sentences.

We cleaned each dataset by removing texts with less than 200 tokens (which are meaningless texts) and also by removing duplicated pairs of texts that appear in the original datasets with different problem ids. In total, we just remove 6 problems from the "small" dataset and 19 problems from the "large" dataset.

We split the dataset into three parts: train, validation, and test. We trained our models on the train split, using the validation split to calibrate the hyperparameters, and used the test split to get a reference score of the model. We did not use any of the samples in the test split to calibrate our model, so the score in this set tells us about the generalization ability of the model. Our splits were done using the same fixed pairs given by the dataset. We made these splits author disjoint, that is, no text in one split has the same author as another text in a different split. All the splits were defined with a balanced proportion of true and false problems. Table 4.2 shows the total number of problems and the number of problems in the positive class (those problems belonging to the same author) on our splits.

**Table 4.2:** Total and positive problems for each split.

| | "Small" dataset | | "Large" dataset | |
|---|---|---|---|---|
| | Total | Positive | Total | Positive |
| Train split | 42,077 | 22,560 | 220,438 | 120,201 |
| Val split | 5,259 | 2,636 | 27,554 | 13,783 |
| Test split | 5,259 | 2,633 | 27,554 | 13,777 |

With regard to our proposed technique to model texts as graphs, in the Table 4.3 we include the average number of nodes and edges of each graph construction over all the texts used in the "large" dataset for the PAN 2021 Authorship Verification task; as a reference, each text has an average of 4,950 tokens.

**Table 4.3:** Average nodes and edge number for each graph construction in "large" dataset for the PAN 2021 Authorship Verification task.

| | Short | Med | Full |
|---|---|---|---|
| Nodes | 33 | 138 | 1207 |
| Edges | 407 | 1168 | 3424 |

## 4.3 Baselines

We compare our approach with two baseline methods. The first baseline consist of a text compression method that calculates cross-entropy, based on Teahan and Harper [46] and implemented by to Potthast et al. [37]. This is a character-based method that consists of two steps. First, it calculates the average and absolute difference of document pairs cross-entropy; then, by using a logistic regression model, it calculates the probability that the two texts were written by the same author.

The second baseline is based on the cosine similarity of the documents represented by character 4-grams and TF-IDF weights as used by Kestemont et al. [20]. This method first obtains a vectorization of each pair of documents using the character 4-grams to represent the dimensions of the vector and the TF-IDF weights to represent the value of each dimension. TF-IDF is a statistical measure that evaluates the relevance of a term in a document. For this, two metrics are multiplied: term frequency, which is the number of times a term appears in a document, and inverse document frequency, which measures how common or rare a term is in a set of documents. For each pair of documents, the cosine similarity of the vectors is obtained. Then, the resulting similarities are optimized and projected through a simple re-scaling operation. Via a grid search, the optimal verification threshold is determined.

## 4.4 Experimental settings for GBSN

For all the experiments with our proposed architecture, we trained the network using an ADAM optimizer and binary cross-entropy as loss function. In each epoch, all the pairs in the train split were introduced to the model in shuffled order.

For each architecture, we trained along with a fixed number of epochs (150 or less) and saved the model that achieved the lowest loss in the validation split as our best model. We report the score of the best model in the test split. The same architecture can arrive in different weights because of the randomness in the training, so the scores reported are the average of 3 distinct executions over the same architecture.

When varying the batch size used for training we note a significant change in the performance, so we made all our experiments with a batch size of 256.

## 4.5 Summary

In this chapter we described the metric used to score our models, the data used for train and test, the baseline methods proposed to compare our model with, and the experimental configuration used when training our GBSN architecture.

# Chapter 5

# Results

In this chapter, we present metrics obtained from the experiments performed and an analysis of these results. The following sections show the results of the baseline methods, the graph-based Siamese architecture, and the graph-based Siamese ensemble architecture independently. Next, we present the results of the threshold adjustment for both graph-based Siamese single and ensemble architectures. Finally, we show the performance of our graph-based submission in the PAN@CLEF 2021 Authorship Verification task.

## 5.1 Results of Baselines

We compare our approach with two baseline methods. The first baseline consists of a text compression method that calculates cross-entropy, based on Teahan and Harper [46]. The second baseline is based on the cosine similarity of the documents represented by character 4-grams and TF-IDF weights proposed by Dehak et al. [12].

Each baseline was trained using both train and validation split together and tested on the test split. These two methods were also used in the PAN@CLEF Authorship Verification task 2021 [21] as baselines to compare all submitted models. The detailed results obtained by the baselines are summarized in Table 5.1.

**Table 5.1:** Results obtained by the baseline methods on our train and test splits of the "small" and "large" datasets.

| Baseline | Test split | AUC | F1 | c@1 | F_0.5u | Brier | Overall |
|---|---|---|---|---|---|---|---|
| Compression | "Small" | 77.40 | 70.60 | 68.70 | 68.20 | 80.30 | 73.00 |
| | "Large" | 77.50 | 72.10 | 69.40 | 68.50 | 80.10 | 73.50 |
| Cosine | "Small" | 76.00 | 75.60 | 69.60 | 67.20 | 78.10 | 73.30 |
| | "Large" | 77.60 | 78.80 | 69.50 | 67.20 | 78.30 | 74.30 |

## 5.2 Results of the GBSN architecture

In this section, we report the results of several experiments with the graph-based Siamese architecture varying the graph convolutional layers, the pooling, and the clas-

sification layer.

### 5.2.1 Varying the Graph Convolutional Layers

In Table 5.2 we show the average score obtained when using 3, 6, 9, and 12 layers (L) of each graph convolutional type (Type column), all these experiments were made with $L_{pool} = 4$, $L_{class} = 4$, and a batch size of 256. The experiments were performed with the three graph representations independently (Graph column). We highlight with boldface and underline the best results obtained for each graph representation.

We did not perform the experiments marked as "NR" in the last column. These experiments correspond to the *full graph* representation which is significantly larger than the other graph representations (*short* and *med*). This issue makes the experiments computationally more expensive than the others. Besides, the results obtained with $L = 9$ on the *full graphs* are lower than the results with $L = 6$ for all convolutional layers except the GraphConv layers, so we considered that experiments with $L = 12$ using the *full graph* were not necessary.

GraphConv and LEConv have similar performances. Both have the best score using 6 or 9 layers and almost always the LEConv is better than GraphConv when looking at experiments with equal parameters. Using more than 9 layers did not improve the score in a significant way. The best scores using 12 layers of GCN2Conv are greater than the ones obtained using 9 layers for *short* and *med* graph representations. In fact, the best scores over the *short graphs* are obtained with 12 layers. However, using the *med graph* and the *full graph* representation obtain the best performance with just 6 layers. None of these scores are better than the ones with the LEConv layer. TAGConv convolutional layer achieved the best performance when using just three layers, for the three graph representations tested. This is probably because one single layer can take into account information of nodes at distance 3. For the *full graph* representation the best performance is obtained with this architecture but for the other graph representations, the score is not greater than the architecture using LEConv layers.

When comparing architectures using the same number of layers of LEConv or GraphConv the performance using *med graph* is better than the ones using the *full graph* or the *short graph*. This remains true when using 3 or 6 layers of GCN2Conv but for more layers, the performance over the *short graph* overcome the one over the *med graph*.

When comparing architectures using TAGConv the performance over the *full graph* is better than the one over *short graph* and this last is also better than the one over *med graph*.

From all these results we can note that the best performance of a given architecture depends on the graph representation chosen. We will focus further experiments in architectures using 6 layers of LEConv, 9 layers of LEConv, and 3 layers of TAGConv because these perform the best for *short*, *med*, and *full* graph representations respectively. The architecture with 12 layers of LEConv also obtained good performance with *med graph* representation, but not better than the one obtained with just 9 layers, so we will not consider it due to the high computational cost.

Figure 5.1 shows how the loss and the five scoring metrics varied along the epochs in training for two executions. The two upper images represent a model with 9 layers of LEConv with *med graph*. The two lower images correspond to a model with 3 layers of

**Table 5.2:** Varying $L$ and *type*. With $L_{pool} = 4$ and $L_{class} = 4$; trained with batch size of 256.
NR = Not reported due computational cost , ** = Score of only one execution.

| Type | Graph | $L = 3$ | $L = 6$ | $L = 9$ | $L = 12$ |
|---|---|---|---|---|---|
| LEConv | Short | 85.67 | **86.64** | 85.93 | 86.25 |
| | Med | 86.39 | 86.69 | 87.18 | **87.19** |
| | Full | 85.75 | 86.01 | 85.94 | NR |
| GraphConv | Short | 85.72 | 85.95 | 86.29 | 86.05 |
| | Med | 85.97 | 86.50 | 86.94 | 86.45 |
| | Full | 84.95 | 86.03 | 86.53 | NR |
| GCN2Conv | Short | 85.33 | 86.19 | 85.81 | 86.37 |
| | Med | 85.46 | 86.50 | 85.72 | 85.85 |
| | Full | 83.79 | 84.98 | 84.68 | NR |
| TAGConv | Short | 86.34 | 83.65 | 82.19 | 80.76 |
| | Med | 86.02 | 83.22 | 80.22 | 80.45 |
| | Full | **86.82** | 84.92 | 83.00** | NR |

TAGConv with *full graph*. Figures 5.1(a) and 5.1(c) show the loss in test and validation split and Figures 5.1(b) and 5.1(d) show the five scores and its average in validation.

Looking carefully at the Figures 5.1(a) and 5.1(b) we can notice an inverse correlation between the loss and the scores: when the loss increases the scores decrease and the other way around. We observed these in all the experiments which tells us that our loss objective is a good fit for optimizing our scores. We can notice that the loss in the validation split is more stable for the model with LEConv layer than for the model with TAGConv layer. The loss and scores for both the GraphConv and GCN2Conv layers have a similar behavior to the model with LEConv. In general, the experiments performed with TAGConv are the most unstable.

### 5.2.2 Varying the Pooling and Classification Layers

We varied the number of layers used for the global pooling ($L_{pool}$) and the number of layers used in classification ($L_{class}$). Table 5.3 shows the scores for the models using 6 and 9 layers of the LEConv graph convolutional layer and for the models using 3 layers of the TAGConv graph convolutional layer. Each row shows the number of pooling layers ($L_{pool}$), the graph representation used and the columns that correspond to the number of classification layers, ($L_{class}$). For each architecture tested and graph representation used we highlight with boldface and underline the best results.

- In the experiments over the *short graphs*, we can notice:
  - Using either LEConv or TAGConv layers, the performance is almost always better using 4 classification layers than using 2 classification layers. The only exception is in the experiments performed with 9 LEConv layers and $L_{pool} = 6$.
  - The experiments with 6 and 9 LEConv layers with 4 classification layers present different behaviors. The performance of the architectures with 6

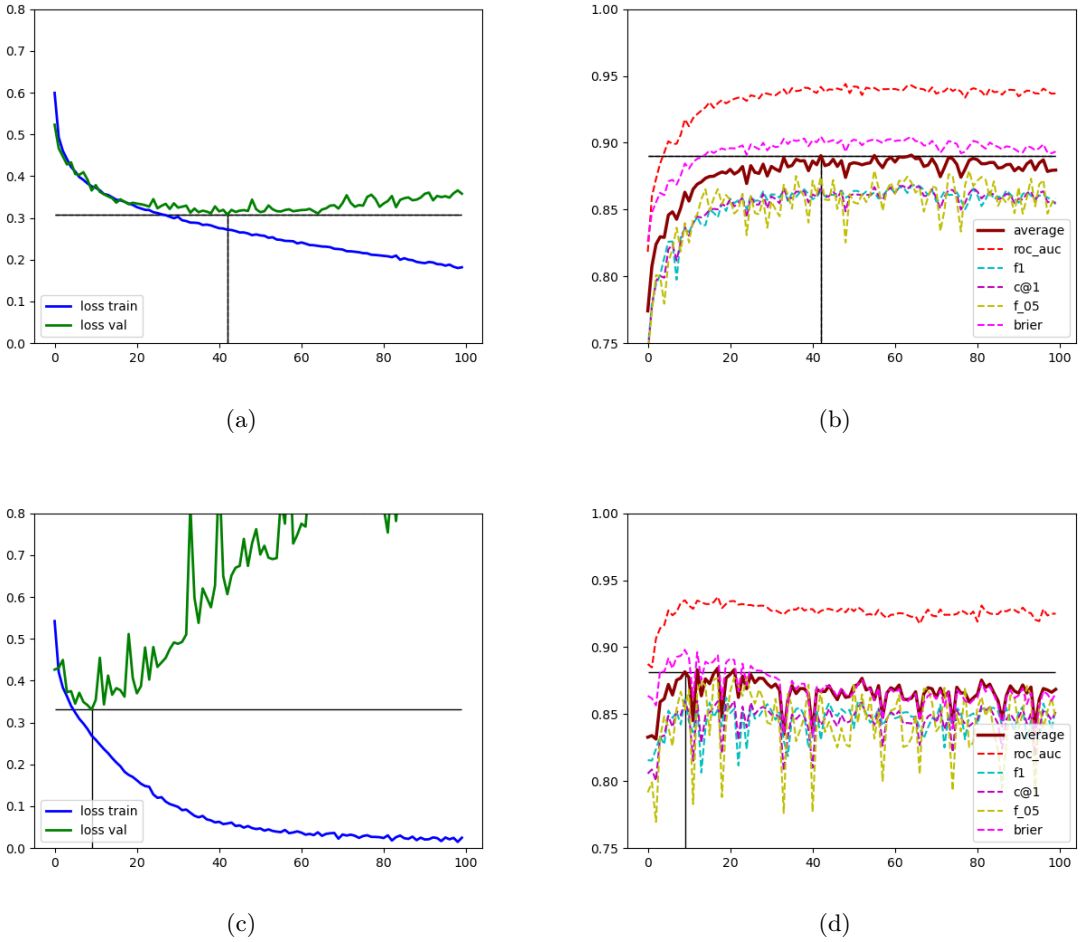



**Figure 5.1:** Loss and scores across epochs of GBSN using LEConv and TAGConv.
Loss (a) and scores (b) for GBSN with $L = 9$, $conv\_type = LEConv$, $L_{pool} = 8$ and $L_{class} = 4$ for *med graph*. Loss (c) and scores (d) for GBSN with $L = 3$, $conv\_type = TAGConv$, $L_{pool} = 4$ and $L_{class} = 4$ for *full graph*.

LEConv layers decreases when adding more pooling layers and for the architecture with 9 layers the performance increases when adding more pooling layers. This may be because the models with 9 LEConv layers can learn more complex patterns from the *short graph* and are improved allowing the attention in the pooling layer to be more expressive. Instead, the models with just 6 LEConv layers overfit when the attention in the pooling layers is too complex.

– When the TAGConv layers are used with 2 classification layers the performance slightly decreases when we increase the pooling layers. In contrast, when using 4 classification layers the performance starts growing when we increase the pooling layers.

| | | TAGConv, $L = 3$ | | LEConv, $L = 6$ | | LEConv, $L = 9$ | |
|---|---|---|---|---|---|---|---|
| | | $L_{class}$ | | $L_{class}$ | | $L_{class}$ | |
| $L_{pool}$ | Graph | 2 | 4 | 2 | 4 | 2 | 4 |
| 4 | Short | 86.06 | 86.34 | 86.08 | **86.64** | 85.72 | 85.93 |
| | Med | 85.52 | **86.02** | 86.60 | 86.69 | 86.85 | 87.18 |
| | Full | 86.50 | **86.82** | **86.84** | 86.01 | **86.62** | 85.94 |
| 6 | Short | 85.95 | 86.38 | 85.62 | 86.22 | 86.22 | 86.08 |
| | Med | 85.18 | 85.85 | **86.91** | 86.72 | 86.51 | 86.82 |
| | Full | 85.92 | 86.80 | 86.81 | 86.37 | 86.43 | 86.11 |
| 8 | Short | 85.85 | **86.59** | 86.03 | 86.08 | 85.57 | **86.60** |
| | Med | 85.16 | 85.82 | 85.96 | 86.66 | 86.90 | **87.45** |
| | Full | 86.65 | 85.88 | 86.06 | 85.81 | NR | NR |

**Table 5.3:** Results of experiments when varying $L_{pool}$, $L_{class}$ and $L$. With convolutional layers TAGConv and LEConv and trained with batch size of 256. NR = Not reported due computational cost.

- With respect to the experiments over the *med graphs*, we can observe:
  - Using TAGConv layers, the performance is better when the architecture uses 4 classification layers than using 2 classification layers. Also, in these experiments, when the pooling layers increase the performance decreases.
- In the experiments over the *full graphs*, we can notice:
  - When the architecture uses LEConv layers, the models improve their performance with 2 classification layers instead of 4.
  - Particularly, when using LEConv layers with 2 classification layers, the models decrease their performance when adding more than 4 pooling layers.
  - When using TAGConv layers the behavior is different, the best performance is obtained when using 4 classification layers and 4 pooling layers; the performance decreases if we vary just one of the parameters, that is, when we increase the pooling layers to 8 or when we decrease to 2 classification layers. But the performance increases again when using both, 8 pooling layers and 2 classification layers.

We aim to tune the GBSN and to select the best architecture for the *short graph* and *med graph* representations because we will use them as a base of the GBSN Ensemble model in Section 5.3. We did not consider architectures over *full graph* because the experiments with these graph representations are computationally more expensive and the performance (86.84 in the best architecture) is not much higher than the performance of the architectures over the *short graph* (86.63 in the best architecture). Table 5.4 shows the score obtained in individual executions of the best architectures chosen for *short graph* and *med graph*. In the first column we show the architecture selected, the second column shows the graph representation, the third column is a number to identify the particular instance, and the last column is the average score of the five evaluation metrics.

**Table 5.4:** Best instances of the architectures for *short graph* and *med graph*.

| Architecture | | | | Graph | Instance | Score |
|---|---|---|---|---|---|---|
| *conv_type* | $L$ | $L_{pool}$ | $L_{class}$ | | | |
| LEConv | 6 | 4 | 4 | *Short* | 1 | 86.73 |
| | | | | | 2 | 86.65 |
| LEConv | 9 | 8 | 4 | *Med* | 1 | 87.50 |
| | | | | | 2 | 87.43 |

## 5.3 Results of the GBSN Ensemble Architecture

In this section, we show experiments using the GBSN Ensemble architecture. For all our experiments, we fixed the number of classification layers used in the GBSN Ensemble to 5. Given that in the ensemble architecture we concatenate the result of more than one feature extraction component, the input vector of the classification network is larger. We choose to use 5 classification layers to allow the classification network to be more expressive.

From now to the rest of this work, we will refer as *Short 1* to the model that corresponds to the first instance of the architecture using the *short graph* representation from Table 5.4. Similarly, we will refer with the same nomenclature to the other instances listed in this Table. We made our ensemble experiments using only the best architectures for *short* and *med* graph representations because these are computational less expensive than the architectures using *full graph* representations.

### 5.3.1 Results of Different Training Strategies

We evaluated three training strategies:

1. To train the GBSN Ensemble architecture all together with random weight initialization.

2. First to train the GBSN architecture for each component, second to initialize the weights of the GBSN Ensemble architecture with the weights obtained from the base GBSN architectures, and finally to train the ensemble together without freezing any weights.

3. To train the GBSN architecture for each component and to initialize the weights of the GBSN Ensemble as in the previous point and also freeze the weights in the feature extraction components to train only the classification step weights.

Table 5.5 shows the scores when training GBSN Ensembles. Each row in the table corresponds to an ensemble with the feature extraction components described in the first column. The second column corresponds to the score when training without initialization, the third column corresponds to the score when initializing the weights from the single models but without freezing them, and the last column correspond to the score when initializing the weights and freezing the ones in the feature extraction

components. The best performances across different training strategies are highlighted with boldface and underline in each row.

**Table 5.5:** GBSN Ensemble with different training strategies.

| | No transfer | Transfer weights, not freeze | Transfer weights, freeze |
|---|---|---|---|
| Med 1 (reference) | *87.50* | | |
| Med 1 + Med 2 | 86.80 | **88.51** | 88.30 |
| Short 1 + Med 1 | 86.59 | 88.53 | **88.64** |
| Short 1 + Short 2 + Med 1 + Med 2 | 86.66 | 88.34 | **88.85** |

The first row shows the best score obtained by the single GBSN over *med graph* in a single experiment, as reported before in Table 5.4. The second row corresponds to an experiment combining two instances of the same feature extraction components over the *med graph* representation; note that the best individual score was 87.50 and the best ensemble score was 88.51, so using two independent instances let us to increase the performance in more than 1%. The other two rows show models that combine different feature extraction layers. In general, training without transferring the weights showed the worst scores, even lower than the scores achieved when using only single instances (see Table 5.4). When combining different feature extraction components the best score was obtained when transferring and freezing the weights in the feature extraction components.

Using the last training strategy has another advantage: because of the frozen weights, we just need to train the classification step, so the training is faster. With this strategy, the models achieve the best performance usually in no more than 20 epochs. We will focus the next experiments using this training strategy because the best performance was achieved using this strategy and it also provides less computational cost than the others.

### 5.3.2 Adding Stylistic Features Component

Until now we have evaluated the performance of feature extraction components over graph representations. These graph representations capture mainly the grammatical structure of the texts, so the model may be improved if we include stylistic features. Furthermore, until now, the results reported were obtained training and testing on the "small" dataset splits. In this section, we will show the results of the experiments with the ensemble architectures considering stylistic feature components over the "small" and "large" dataset splits.

Table 5.6 shows the comparative performance of the evaluated models. Each row shows the average of the five evaluation metrics achieved by a given model when trained and tested on our splits for the "small" and "large" datasets, respectively. We include in this table the performance for the baselines, GBSN with single feature extraction components, and GBSN Ensembles.

The first two rows show the performance of the two selected baselines. The third and fourth rows correspond to the model using only the feature extraction component

on the *short graph* representation and the model using only the feature extraction component on the *med graph* representation, respectively. The fifth row shows the sores of the model using feature extraction components on both *short graph* and *med graph* representation without stylistic features. The sixth row shows the scores of a model with feature feature component on the *short*, *med graph* representations, along with the stylistic features. The seventh row shows the scores of a model combining feature extraction components on two instances of *short graph* representations and two instances of *med graph* representations. Finally, the last row shows the scores of a model combining feature extraction components on two *short graph* representations, two *med graph* representations, and stylistic features. We highlight with boldface and underline the best results obtained on "small" and "large" dataset splits.

**Table 5.6:** Average performance of Graph-based Siamese Network (GBSN) with single and ensemble feature extraction components. Ensemble was training transferring and freezing feature extraction components weights.
** = Score of only one execution.

| Model | "Small" dataset splits | "Large" dataset splits |
|---|---|---|
| Baseline compression | 73.00** | 73.50** |
| Baseline cosine | 72.90** | 74.30** |
| Short | 86.64 | 89.47 |
| Med | 87.45 | 90.35 |
| Short + Med | 88.64 | 91.35 |
| Short + Med + Stylistic features | 89.13 | 91.36 |
| Short(x2) + Med(x2) | 88.85 | 91.66 |
| Short(x2) + Med(x2) + Stylistic features | **89.31** | **91.68** |

Training on "large" dataset splits improve the performance of all models. Also, all our proposed models considerably outperform the score of the baselines. The inclusion of the stylistic features component allows the architecture to improve the performance when comparing with the equivalent architecture without them. This improvement is more substantial when the models are trained in the "small" dataset splits. When comparing models trained in the "large" dataset splits, adding stylistic features component to the Short + Med model just improve 0.01% and adding stylistic features component to the Short(x2) + Med(x2) just improve 0.02% These results indicate that the graph-based models learn better features when more data is available. The GBSN performance when trained in "large" dataset performed almost as well as when adding the stylistic features.

## 5.4 Results of the threshold adjustment

As we described in Section 3.4, we can adjust the threshold and margin to improve the performance of our model with respect to the metrics. As a final tuning, we performed a grid search over different thresholds and margins to optimize the average of

the scores in the validation split. Then, we selected the best threshold and margin obtained and evaluate the model in the test split.

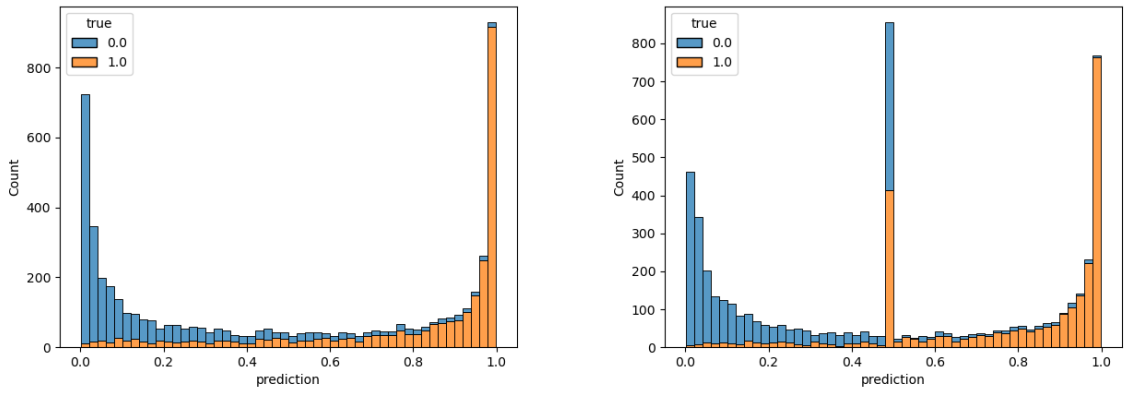
Figure 5.2 shows a heatmap of the scores when varying the threshold and margin for an architecture using 9 layers of LEConv, $L_{pool} = 8$ and $L_{class} = 4$ over *med graph*. We searched thresholds values between 0.05 and 0.95 and margin values between 0 and 0.25, both with increments of 0.05. From these results, we can notice that the score increases from 89.04 in the default threshold-margin of 0.50-0.00 to 90.24 in the best configuration of 0.50-0.20.


**Figure 5.2:** Threshold grid search in val split for GBSN with parameters *conv_type* = LEConv, $L = 9$, $L_{pool} = 8$, $L_{class} = 4$ over *med graph*.

Figure 5.3(a) and 5.3(b) show the distribution for positive and negative instances before and after the threshold adjustment, respectively. The X-axis corresponds to the score obtained by our model and the Y-axis shows the number of problems obtained with these scores. The blue bars are the proportion of problems with a different-author label (0) and the orange bars are the proportion of problems with a same-author label (1).

The different-author labeled problems (blue) are mainly classified with low values and the same-author labeled ones (orange) are mainly classified with high values. With the threshold adjustment, the scores increase mainly because some of the difficult to solve problems were sent to 0.5, i.e., the problems were left unanswered. These unanswered problems are represented in the bar at 0.5 in Figure 5.3(b).

The threshold adjustment significantly improves the performance of the model in the test split. Table 5.7 shows the scores and improvements in the "small" and "large" dataset splits for some of our single and ensemble architectures. Each row corresponds to a model presented in Table 5.6 with the exception of the baselines. The first column shows the name of the model, the second and third are the performance after the threshold adjustment and the improvement when compared with no adjustment in the "small" dataset splits and the fourth and fifth columns are the performance after the threshold adjustment and the improvement when compared with no adjustment in




(a) (b)

**Figure 5.3:** Histogram of distribution for positive and negative instances.
(a) Before threshold adjust (b) With threshold adjust of $th = 0.5$ and $m = 0.2$.

the "large" dataset splits. We highlight with boldface and underline the best results obtained in each dataset split.

**Table 5.7:** Average performance after threshold and margin adjust of Graph-based Siamese Network (GBSN) with single and ensemble feature extraction components. Ensemble was trained transferring and freezing feature extraction component weights.

| | "Small" dataset | | "Large" dataset | |
|---|---|---|---|---|
| | After adjustment | Increment | After adjustment | Increment |
| Short | 87.87 | 1.23 | 90.63 | 1.16 |
| Med | 88.59 | 1.14 | 91.53 | 1.18 |
| Short + Med | 89.70 | 1.06 | 92.45 | 1.10 |
| Short + Med + Stylistic features | 90.30 | 1.17 | 92.45 | 1.09 |
| Short(x2) + Med(x2) | 89.86 | 1.01 | 92.80 | 1.14 |
| Short(x2) + Med(x2) + Stylistic features | **90.56** | 1.25 | **92.83** | 1.15 |

## 5.5 GBSN performance in the 2021 PAN@CLEF Authorship verification shared task

The CLEF 2021 conference is the twenty-second edition of the popular CLEF campaign and workshop series that has run since 2000 contributing to the systematic evaluation of multilingual and multimodal information access systems, primarily through

experimentation on shared tasks. In 2010 CLEF was launched in a new format, as a conference with research presentations, panels, poster and demo sessions, and laboratory evaluation workshops. These are proposed and operated by groups of organizers volunteering their time and effort to define, promote, administrate and run an evaluation activity.

CLEF 2021 (http://clef2021.clef-initiative.eu/) was organized by the University "Politehnica" of Bucharest, Romania, from 21 to 24 September 2021. Their 12 selected labs represented scientific challenges based on new data sets and real-world problems in multimodal and multilingual information access. These data sets provide unique opportunities for scientists to explore collections, develop solutions for these problems, receive feedback on the performance of their solutions, and discuss the issues with peers at the workshops.

One of these labs is PAN: Digital Text Forensics and Stylometry (http://pan.webis.de/). It is a networking initiative for digital text forensics, where researchers and practitioners study technologies that analyze texts with regard to originality, authorship, and trustworthiness. PAN provides evaluation resources consisting of large-scale corpora, performance measures, and web services that allow for meaningful evaluations. The main goal is to provide for sustainable and reproducible evaluations, to get a clear view of the capabilities of state-of-the-art algorithms.

In the 2021 year's edition of PAN, the authorship identification track focused on open-set authorship verification, so that systems are applied to unknown documents by previously unseen authors in a new domain. This edition was the second task installment in a renewed three-year program on the PAN authorship track (2020–2022), in which the scope, the difficulty and, the realism of the tasks are gradually increased each year.

After last year's edition focused on providing participants with the largest pool of calibration material by far of any previous authorship shared task at PAN, we sought to improve the difficulty this year by sampling a fully disjoint test set. This year's test set comes with document pairs of exclusively unseen authors writing in unseen fandom domains, which results in an open-set or "true" authorship verification scenario, which is conventionally considered a much more demanding problem than attribution.

This 2021 Authorship Verification shared task adopt the five evaluation metrics used also to evaluate our models: AUC ROC, C@1, F1, F0.5u and Brier. They provided three baseline systems (calibrated on the "small" training set) for comparison:

1. A compression-based approach [37, 46].
2. A naive distance-based, first-order bag-of-words model [20].
3. A short-text variant of Koppel et al. [25] unmasking by Bevendorff et al. [3] which had yielded good empirical results in the recent past.

First, two baselines are the same used by us as baselines methods. This year the authorship verification task received 13 submissions from 10 participating teams. Teams were allowed to hand in exactly one submission per training dataset ("large" and "small").

We submit two models to the 2021 PAN@CLEF Authorship Verification task, one trained in the "small" dataset and the other trained in the "large" dataset. Both models

have the same architecture: GBSN Ensemble architecture with two short, two med, and stylistic components. Each graph-based component has an architecture of 6 LEConv layers and 4 layers for the fully connected network in the pooling step. The final classification network has 5 layers. The ensemble architecture was trained by transferring and freezing weights from the feature extraction components. Both models are improved with the threshold adjust method proposed in this work. The details about the implementation appear in our published paper in the notebooks of these conferences [14].

We obtained the second-best score for the model trained in the "large" dataset and the third-best score for the model trained in the "small" dataset. Table 5.8 shows the official scores reported in the task overview [21].

The table shows system rankings for PAN 2021 submissions across five evaluation metrics: AUC, c@1, F 1 , F 0.5u , Brier, and an overall mean score (as the final ranking criterion). The dataset column indicates which calibration dataset was used. Bold digits reflect the per-column maximum. Horizontal lines indicate the range of scores yielded by the baselines (in italics). We just include the submissions with performance above baselines.

**Table 5.8:** System rankings for all PAN 2021 submissions. Taken from [21].

| Team | Dataset | AUC | c@1 | F1 | F0.5u | Brier | Overall |
|---|---|---|---|---|---|---|---|
| boenninghoff21 | "large" | **0.9869** | **0.9502** | **0.9524** | **0.9378** | **0.9452** | **0.9545** |
| embarcaderoruiz21 | "large" | 0.9697 | 0.9306 | 0.9342 | 0.9147 | 0.9305 | 0.9359 |
| weerasinghe21 | "large" | 0.9719 | 0.9172 | 0.9159 | 0.9245 | 0.9340 | 0.9327 |
| weerasinghe21 | "small" | 0.9666 | 0.9103 | 0.9071 | 0.9270 | 0.9290 | 0.9280 |
| menta21 | "large" | 0.9635 | 0.9024 | 0.8990 | 0.9186 | 0.9155 | 0.9198 |
| peng21 | "small" | 0.9172 | 0.9172 | 0.9167 | 0.9200 | 0.9172 | 0.9177 |
| embarcaderoruiz21 | "small" | 0.9470 | 0.8982 | 0.9040 | 0.8785 | 0.9072 | 0.9170 |
| menta21 | "small" | 0.9385 | 0.8662 | 0.8620 | 0.8787 | 0.8762 | 0.8843 |
| rabinovits21 | "small" | 0.8129 | 0.9129 | 0.8094 | 0.8186 | 0.8129 | 0.8133 |
| ikae21 | "small" | 0.9041 | 0.7586 | 0.8145 | 0.7233 | 0.8247 | 0.8050 |
| *unmasking21* | "small" | 0.8298 | 0.7707 | 0.7803 | 0.7466 | 0.7904 | 0.7836 |
| tyo21 | "large" | 0.8275 | 0.7594 | 0.7911 | 0.7257 | 0.8123 | 0.7832 |
| *naive21* | "small" | 0.7956 | 0.7320 | 0.7856 | 0.6998 | 0.7867 | 0.7600 |
| *compressor21* | "small" | 0.7896 | 0.7282 | 0.7609 | 0.7027 | 0.8094 | 0.7581 |

We want to note several relevant points:

- For our submitted models, the scores reported by PAN were higher than the ones obtained in our test split. So the scores for the model described in Table 5.7 are not directly comparable with the scores reported by the PAN task committee.

- We do not use any up-sampling technique over the given dataset and, because of our train framework, for each dataset, we train our submitted model using only 90% of the available data; that is, the "small" model was trained using just 47,336 problems and the "large" model using just 247,992 problems; this is relevant because our architecture showed to work better with more training pairs.

- The submitted model is not the model with the best performance found in all our experimentation. It was submitted because at the death-line of the task it was the best performing stable architecture.

The results obtained show us that our proposed approach has performance comparable with the state of the art in this research area. Also, our experiments show us that the approach can improve their performance or be modified to achieve good results with considerably less computational cost.

## 5.6 Summary

In this chapter we present the experiments performed, the metrics obtained, and an analysis of the results. Table 5.7 summarize the best performance of our models concerning the baselines and compare the use of single and ensemble architectures.

In section 5.5 we present system rankings for PAN 2021 Authorship Verification tasks, showing that our approach has a performance comparable with the state of the art for this task.

# Chapter 6

# Conclusions and Future Work

In this paper, we presented a novel Siamese network architecture composed of two graph convolutional neural networks with graph level pooling, and classification layers to approach the authorship verification task. For the text representation, we propose three graph models and evaluate their appropriateness for the above-mentioned task. The graph representations from texts are based on the relation of the POS labels and co-occurrence of the words. These representations, allow us to choose which POS labels are masked as a single node in the graph, let us reduce the graph complexity, and focus the representation in POS labels relevant to a specific task.

## 6.1 Conclusions

Specifically, we evaluated the following graph representations: *Short*, where all words with the same POS label are identified as the same node; *Med*, where POS labels corresponding to adjectives, nouns, adverbs, verbs, cardinal numbers, foreign words, list markers, and symbols are identified as the same node and *Full* where each tuple of (word, POS) is represented as a node; this corresponds to the traditional co-occurrence graph.

As part of our proposed architecture, we evaluated several graph convolutional layers (LEConv, GraphConv, GCN2Conv, and TAGConv) over the graph representations. We presented detailed scores of our experiments and from them, we can conclude the following:

- Concerning the graph representation used:
  - The performance of the architecture (graph convolutional layers, pooling layers, classification layers) are strongly dependent of the graph representation chosen.
  - The performance of the models with the *med graph* is in general better than the performance of the models with *short graph* and *full graph* representation. The GBSN model with individual components achieved the best score with the *med graph* representation.
  - The performance of the models with *short graph* representations show good performance, achieving average scores of 86.64% and 89.47% in the "small" and "large" dataset splits respectively. Even being a relatively small graph

with usually just 33 nodes and 407 edges, this graph is a good alternative to trade off performance for computational cost.

- Concerning the graph convolutional layers used:
  - Models with LEConv layer have in general (but not always) better performance than models with GraphConv and GCN2Conv layers over all the graph representations tested. With this kind of layer good performance is obtained when using 6 or 9 layers.
  - Models with TAGConv layer have their best performance when using just 3 layers. The scores obtained with these models are the best over *full graph* representations. Also, the score obtained over the *short graph* representation (86.59%) is comparable with the best score obtained by the model using LEConv layer over the *short graph* representation (86.64%).
  - When varying the layers used in pooling and the classification layers we cannot conclude a general rule to improve the performance, but our experiments show that both are relevant hyperparameters to consider when tuning a model.

We also proposed to combine more than one component for feature extraction, we evaluated graph-based and stylistic-based components with different training strategies. We found that transferring the weights from a single component architecture and freezing these in the ensemble architecture is the best training strategy because it has the best performance and the lowest computational cost.

In general, the combined use of more than one graph-based component improves the performance, even if we use several components based on the same graph representation. Finally, we showed that our architectures can be improved with a simple threshold adjustment, giving us final scores comparable with the state of the art in this task.

## 6.2 Future work

The new Graph-based Siamese network showed good performance in the authorship verification task. For future work, we want to evaluate this new approach in other authorship analysis tasks. Also, the proposed graph representation is based on the relation of words, capturing mainly the structural information of the POS labels. Character level information has shown to have good performance in authorship attribution task [43] [39], so it will be interesting to generalize the graph representation strategy to include information at the character level.

# Bibliography


[1] E. Araujo-Pino, H. Gómez-Adorno, and G. Fuentes-Pineda. Siamese Network applied to Authorship Verification. *Notebook for PAN at CLEF 2020*, page 8, 2020.

[2] D. Bagnall. Author identification using multi-headed recurrent neural networks. *Notebook for PAN at CLEF 2015*, page 11, June 2015.

[3] J. Bevendorff, B. Stein, M. Hagen, and M. Potthast. Generalizing Unmasking for Short Texts. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 654–659, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1068.

[4] B. Boenninghoff, J. Rupp, R. M. Nickel, and D. Kolossa. Deep Bayes Factor Scoring for Authorship Verification. *Notebook for PAN at CLEF 2020*, page 12, 2020.

[5] G. W. Brier. Verification of forecasts expressed in terms of probability. *MONTHLY WEATHER REVIEW*, 78:1–3, 1950. doi: 10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2.

[6] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature Verification using a "Siamese" Time Delay Neural Network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:669–688, 1993. doi: 10.1142/S0218001493000339.

[7] E. Castillo, O. Cervantes, and D. Vilariño. Text Analysis Using Different Graph-Based Representations. *Computación y Sistemas*, 21(4):581–599, Dec. 2017. ISSN 1405-5546. doi: 10.13053/cys-21-4-2551.

[8] E. Castillo, O. Cervantes, and D. Vilariño. Authorship Verification using a Graph Knowledge Discovery Approach. *Journal of Intelligent & Fuzzy Systems*, 36(6): 6075–6087, Jan. 2019. ISSN 1064-1246. doi: 10.3233/JIFS-181934.

[9] C. E. Chaski. Who's At The Keyboard? authorship Attribution in Digital Evidence Investigations. *IJDE*, 4(1):14, 2005.

[10] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and Deep Graph Convolutional Networks, July 2020.

[11] L. Cruz. Authorship recognition with short-text using graph-based techniques. In *Proceedings of the 2019 Workshop on Widening NLP*, pages 153–156, Florence, Italy, Aug. 2019. Association for Computational Linguistics.

[12] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798, 2011. doi: 10.1109/TASL.2010.2064307.

[13] J. Du, S. Zhang, G. Wu, J. M. F. Moura, and S. Kar. Topology Adaptive Graph Convolutional Networks. *arXiv:1710.10370 [cs, stat]*, Feb. 2018.

[14] D. Embarcadero-Ruiz, H. Gómez-Adorno, I. Reyes-Hernández, A. García, and A. Embarcadero-Ruiz. Graph-based Siamese Network for Authorship Verification. *Notebook for PAN at CLEF 2021*, page 11, 2021. ISSN 1613-0073.

[15] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas. Effective identification of source code authors using byte-level information. In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 893–896, 2006. doi: 10.1145/1134285.1134445.

[16] H. Gómez-Adorno, G. Sidorov, D. Pinto, D. Vilariño, and A. Gelbukh. Automatic authorship detection using textual patterns extracted from integrated syntactic graphs. *Sensors*, 16(9), 2016. ISSN 1424-8220. doi: 10.3390/s16091374.

[17] F. Jafariakinabad, S. Tarnpradab, and K. A. Hua. Syntactic Recurrent Neural Network for Authorship Attribution. *arXiv:1902.09723 [cs]*, Feb. 2019.

[18] P. Juola. Authorship Attribution. *Foundations and Trends® in Information Retrieval*, 1(3):233–334, 2007. ISSN 1554-0669, 1554-0677. doi: 10.1561/1500000005.

[19] M. Kestemont, K. Luyckx, W. Daelemans, and T. Crombez. Cross-Genre Authorship Verification Using Unmasking. *English Studies*, 93(3):340–356, May 2012. ISSN 0013-838X, 1744-4217. doi: 10.1080/0013838X.2012.668793.

[20] M. Kestemont, J. Stover, M. Koppel, F. Karsdorp, and W. Daelemans. Authenticating the writings of Julius Caesar. *Expert Systems with Applications*, 63:86–96, Nov. 2016. ISSN 09574174. doi: 10.1016/j.eswa.2016.06.029.

[21] M. Kestemont, E. Manjavacas, I. Markov, J. Bevendorff, M. Wiegmann, E. Stamatatos, B. Stein, and M. Potthast. Overview of the Cross-Domain Authorship Verification Task at PAN 2021. *Proceedings of the Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum*, page 17, 2021.

[22] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. *Conference paper at ICLR*, Feb. 2017.

[23] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese Neural Networks for One-shot Image Recognition. *ICML deep learning workshop*, 2:8, 2015.

[24] M. Koppel and Y. Winter. Determining if two documents are written by the same author. *Journal of the Association for Information Science & Technology*, 65(1): 178–187, Jan. 2014. ISSN 23301635. doi: 10.1002/asi.22954.

[25] M. Koppel, J. Schler, E. Bonchek-Dokow, and B. Dokow. Measuring Differentiability: Unmasking Pseudonymous Authors. *Journal of Machine Learning Research*, 8:1261–1276, 2007.

[26] M. Koppel, J. Schler, and S. Argamon. Authorship attribution in the wild. *Language Resources and Evaluation*, 45:83–94, Mar. 2011. doi: 10.1007/s10579-009-9111-2.

[27] N. M. Kriege, F. D. Johansson, and C. Morris. A Survey on Graph Kernels. *Applied Network Science*, 5(1):6, Dec. 2020. ISSN 2364-8228. doi: 10.1007/s41109-019-0195-3.

[28] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated Graph Sequence Neural Networks. *Proceedings of ICLR*, Sept. 2017.

[29] T. Lippincott. Graph convolutional networks for exploring authorship hypotheses. In *Proceedings of the 3rd Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 76–81, Minneapolis, USA, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-2510.

[30] M. Marcus. Building a Large Annotated Corpus of English: The Penn Treebank:. Technical report, Defense Technical Information Center, Fort Belvoir, VA, Apr. 1993.

[31] S. Mekala and V. V. Bulusu. A Survey On Authorship Attribution Approaches. In *International Journal of Computational Engineering Research (IJCER)*, volume 8, page 8, Oct. 2018.

[32] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609, July 2019. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33014602.

[33] A. Nandy, S. Haldar, S. Banerjee, and S. Mitra. A Survey on Applications of Siamese Neural Networks in Computer Vision. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–5, June 2020. doi: 10.1109/INCET49848.2020.9153977.

[34] A. Penas and A. Rodrigo. A Simple Measure to Assess Non-response. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 1415–1424, 2011.

[35] V. Petras, P. Forner, and P. D. Clough, editors. *Notebook Papers of CLEF 2011 Labs and Workshops, 19-22 September, Amsterdam, the Netherlands*. CEUR-WS.org, 2011. ISBN 978-88-904810-1-7.

[36] D. Pinto, H. Gomez Adorno, D. Vilariño, and V. Singh. A graph-based multi-level linguistic representation for document understanding. *Pattern Recognition Letters*, 41:93–102, May 2014. doi: 10.1016/j.patrec.2013.12.004.

[37] M. Potthast, S. Braun, T. Buz, F. Duffhauss, F. Friedrich, J. M. Gülzow, J. Köhler, W. Lötzsch, F. Müller, M. E. Müller, R. Paßmann, B. Reinke, L. Rettenmeier, T. Rometsch, T. Sommer, M. Träger, S. Wilhelm, B. Stein, E. Stamatatos, and M. Hagen. Who wrote the web? revisiting influential author identification research applicable to information retrieval. In N. Ferro, F. Crestani, M.-F. Moens, J. Mothe, F. Silvestri, G. M. D. Nunzio, C. Hauff, and G. Silvello, editors, *ECIR*, volume 9626 of *Lecture Notes in Computer Science*, pages 393–407. Springer, 2016. ISBN 978-3-319-30670-4.

[38] E. Ranjan, S. Sanyal, and P. P. Talukdar. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. *Thirty-Fourth AAAI Conference on Artificial Intelligence*, Feb. 2020.

[39] U. Sapkota, S. Bethard, M. Montes, and T. Solorio. Not All Character N-grams Are Created Equal: A Study in Authorship Attribution. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–102, Denver, Colorado, May 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1010.

[40] S. Seidman. Authorship Verification Using the Impostors Method. *Notebook for PAN at CLEF 2013*, page 4, 2013.

[41] S. S.Sonawane and P. A. Kulkarni. Graph based Representation and Analysis of Text Document: A Survey of Techniques. *International Journal of Computer Applications*, 96(19):1–8, June 2014. ISSN 09758887. doi: 10.5120/16899-6972.

[42] E. Stamatatos. A survey of modern authorship attribution methods. *Journal of the American Society for Information Science and Technology*, 60(3):538–556, 2009. ISSN 1532-2890. doi: 10.1002/asi.21001.

[43] E. Stamatatos. On the Robustness of Authorship Attribution Based on Character N-gram Features. *JOURNAL OF LAW AND POLICY*, 21(2):20, 2013.

[44] E. Stamatatos, W. Daelemans, B. Verhoeven, M. Potthast, B. Stein, P. Juola, M. A. Sanchez-Perez, and A. Barrón-Cedeño. Overview of the Author Identification Task at PAN 2014. *Proceedings of the CLEF PAN Conference*, page 21, 2014.

[45] E. Stamatatos, W. Daelemans, B. Verhoeven, P. Juola, A. López-López, M. Potthast, and B. Stein. Overview of the Author Identification Task at PAN 2015. *Proceedings of the CLEF PAN Conference*, page 17, 2015.

[46] W. J. Teahan and D. J. Harper. Using Compression-Based Language Models for Text Categorization. In W. B. Croft and J. Lafferty, editors, *Language Modeling for Information Retrieval*, pages 141–165. Springer Netherlands, Dordrecht, 2003. ISBN 978-90-481-6263-5 978-94-017-0171-6. doi: 10.1007/978-94-017-0171-6_7.

[47] J. Weerasinghe and R. Greenstadt. Feature Vector Difference based Neural Network and Logistic Regression Models for Authorship Verification. *Notebook for PAN at CLEF 2020*, page 6, 2020.

[48] J. Weerasinghe, R. Singh, and R. Greenstadt. Feature Vector Difference based Authorship Verification for Open-World Settings. *Notebook for PAN at CLEF 2021*, page 7, 2021.

[49] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2020.2978386.