CUKUROVA UNIVERSITY

INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

# Detecting Multilingual Offensive Language in Social Media Using Deep Neural Networks

**Mahmud BİRECİKLİ**

*Computer Engineering Department*

September, 2023

**CUKUROVA UNIVERSITY**

**INSTITUTE OF NATURAL AND APPLIED SCIENCES**

## MSc THESIS APPROVAL

# Detecting Multilingual Offensive Language in Social Media Using Deep Neural Networks

**Mahmud BİRECİKLİ**

***Computer Engineering Department***

This Master's Thesis was evaluated by the following Jury Members on 15/09/2023 and was approved by unanimity / majority of votes.

Jury : Prof. Dr. Selma Ayşe ÖZEL (Advisor) ……

: Assist. Prof. Dr. Mehmet SARIGÜL ……

: Assoc. Prof. Dr. Fatih KILIÇ ……

**This Thesis was written in the Department of Computer Engineering Institute of Natural and Applied Sciences.**

**Thesis Number:**

**Prof. Dr. Sadık DİNÇER**
**Director**
**Institute of Natural and Applied Sciences**

**CONTENTS**

**CUKUROVA UNIVERSITY**
**INSTITUTE OF NATURAL AND APPLIED SCIENCES**

**MSc THESIS**

# Detecting Multilingual Offensive Language in Social Media Using Deep Neural Networks

**Mahmud BİRECİKLİ**

***Advisor: Prof. Dr. Selma Ayşe ÖZEL***

***Department of Computer Engineering***

**ABSTRACT**

The spread of offensive language on social media platforms has become an alarming reality in society today. The utilization of such language for the purpose of insulting and attacking people represents one of the most detrimental forms of online behavior. Its negative consequences extend to users across different communication platforms, significantly impacting their psychological and mental well-being.

To combat this digital malady, data scientists and NLP researchers have taken the task of finding a solution. They have developed several classifier models employing machine learning and deep learning techniques, aimed at identifying several forms of offensive language within textual contexts. These models are designed to process the text by either removing offensive language or preventing its publication on the internet.

This study seeks to address the issue by evaluating the performance of deep learning methods on a collected dataset that is formed by collecting a numerous amount of Arabic texts and labeling them. Additionally, comparison of the performance of different deep neural network classifiers namely, Convolutional Neural Network (CNN), Reccurrent Neural Network (RNN), and Long Short-Term Memory (LSTM), and a Language Model namely RoBERTa, performed on the Arabic dataset, as well as some additional datasets in English and Turkish languages, aiming to show the effects of different preprocessing on texts, feature selection and effectiveness of deep neural networks and Transformers across different linguistic texts.

The results of this study suggest that RoBERTa is a strong candidate for various language, it achieved the highest validation accuracy across most datasets, showcasing its effectiveness for various languages and tasks. Additionally, an ensemble classifier combining RoBERTa and CNN is introduced and tested, demonstrating good results in improving classification performance.

**Keywords**: Detection of Offensive Language, Textual Data, Deep Neural Networks, Natural Language Processing, Classification

**ÇUKUROVA ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

**YÜKSEK LİSANS TEZİ**

# Derin Sinir Ağlarını Kullanarak Sosyal Medyada Çokludilli Saldırgan Dil Tespit Etme

**Mahmud BİRECİKLİ**

***Danışman: Prof. Dr. Selma Ayşe ÖZEL***

***Bilgisayar Mühendisliği Anabilim Dalı***

## ÖZ

Sosyal medya platformlarında hakaret içeren dilin yayılması, bugün toplumda endişe verici bir gerçek haline gelmiştir. Bu tür dilin insanları aşağılamak ve saldırmak amacıyla kullanılması, çevrimiçi davranış biçimlerinin en zararlı formlarından birini temsil eder. Olumsuz sonuçları, farklı iletişim platformlarındaki kullanıcıları etkileyerek psikolojik ve zihinsel iyi oluşlarını ciddi şekilde etkiler.

Bu dijital soruna karşı, veri bilimcileri ve doğal dil işleme araştırmacıları bir çözüm bulma görevini üstlenmişlerdir. Makine öğrenimi ve derin öğrenme tekniklerini kullanarak metin bağlamlarında çeşitli saldırgan dil türlerini tanımlamayı amaçlayan çeşitli sınıflandırıcı modeller geliştirmişlerdir. Bu modeller, metni ya saldırgan dil içeren kısmı çıkararak ya da internet üzerinde yayınlanmasını engelleyerek çalışır.

Bu çalışma, çok sayıda Arapça metin toplayarak ve bunları etiketleyerek elde edilen veri kümesi üzerinde derin öğrenme yöntemlerinin performansını değerlendirmeyi amaçlamaktadır. Özellikle Evrişimsel Sinir Ağı (CNN), Tekrarlayan Sinir Ağı (RNN) ve Uzun Kısa Süreli Hafıza (LSTM) adlı sınıflandırıcı modelleri ile RoBERTa adlı bir Dil Modeli hazırladığımız Arapça veri kümesi üzerinde eğitilecek ve değerlendirilecek, ayrıca İngilizce ve Türkçe dillerinde bazı ek veri kümeleri üzerinde de test edilecektir. Bu değerlendirme, farklı ön işleme yöntemlerinin metinler üzerindeki etkisini, özellik seçiminin, derin sinir ağlarının ve dönüştürücülerin etkinliğini göstermeyi amaçlamaktadır.

Bu tez çalışmasının sonuçları, RoBERTa'nın çeşitli diller için güçlü bir aday olduğunu göstermektedir; çoğu veri kümesinde en yüksek doğrulama başarısını elde etmiş ve çeşitli diller ve görevler için etkinliğini sergilemiştir. Ayrıca, RoBERTa ve CNN'yi birleştiren bir sınıflandırıcı tanıtılmış ve test edilerek sınıflandırma performansını iyileştirdiği gözlenmiştir.

**Anahtar Kelimeler**: Saldırgan Dil Tespiti, Metin Verileri, Derin Sinir Ağları, Doğal Dil İşleme, Sınıflandırma

## GENİŞLETİLMİŞ ÖZET

Bugünün dijital çağında, Facebook, Twitter ve Instagram gibi çevrimiçi sosyal platformlar, tüm yaş, cinsiyet, ülke ve kültürlerden insanlar arasında büyük popülarite kazanmıştır. Bu platformlar, coğrafi sınırlara bakılmaksızın bireylerin birbirleriyle bağlantı kurmasını ve etkileşimde bulunmasını sağlayan pratik bir yol sunar, fiziksel olarak aynı yerde bulunma gerekliliğini ortadan kaldırır. Bu çevrimiçi sosyal platformları kullanan kişi sayısı dünya genelinde önemli ölçüde artmıştır. DataReportal'ın (2023) Temmuz 2023 tarihli bir raporuna göre, dünya genelinde yaklaşık olarak 4,88 milyar sosyal medya kullanıcısı bulunmaktadır, bu da toplam dünya nüfusunun yaklaşık %60,6'sını oluşturur. Bu platformların kullanımı arttıkça, sanal dünyada çeşitli insan etkileşim biçimleri ortaya çıkmıştır, olumlu davranışlardan olumsuz davranışlara kadar geniş bir yelpaze mevcuttur. Birçok insan yeni arkadaşlar edinme, farklı dilleri ve kültürleri öğrenme, çevrimiçi iş yapma gibi, olumlu etkileşimlerde bulunurken, aynı zamanda olumsuz davranışlarda da artış yaşanmaktadır. Endişe verici bir sorun, çevrimiçi sosyal ağlarda hakaret edici dil ve nefret söyleminin artan bir eğilim olarak karşımıza çıkmasıdır. Ne yazık ki, bu zararlı davranışlar günümüz toplumunda oldukça yaygın hale gelmiştir. Hakaret edici dil ve aşağılayıcı ifadelerin kullanımı, diğer kullanıcılar üzerinde ciddi sonuçlara yol açabilir, hayatlarını olumsuz etkileyebilir.

Hakaret edici dil, başkalarını olumsuz etkileyebilecek incitici ve kaba sözcüklerin sözlü veya yazılı olarak kullanılması olarak tanımlanabilir. Birleşmiş Milletlere göre, günlük dilde "Nefret söylemi", bir grubu veya bireyi ırk, din veya cinsiyet gibi içsel özelliklerine dayalı olarak hedef alan aşağılayıcı bir dil kullanımını ifade eder ve toplumsal uyumu tehdit edebilir (United Nations, n.d.). Birleşmiş Milletler, nefret söylemini, din, etnik köken, milliyet, ırk, renk, köken veya cinsiyet gibi kimlik faktörlerine dayalı olarak bir kişiyi veya grubu hedef alan her türlü iletişimi, konuşmayı, yazıyı veya davranışı ayrımcılık veya saldırı olarak tanımlar. Al-Tufay ve Al-Ghizzy'nin (2022) araştırmasında, "kabalık, hakaret, lakap, tabu, müstehcenlik ve küfür" gibi çeşitli hakaret edici kelimeler ve ifadeler tespit edilmiştir. Bu hakaret edici kelimeler vücut parçalarına, sosyal yönlerine, çağrışımlara, metaforlara, cinselliğe ve küfürlere atıfta bulunmak gibi farklı amaçlar için kullanılmıştır. Elektronik sistemlerin sağladığı anonimlik, kullanıcıların kendilerini özgürce ifade etmelerine ve zihinlerinde ne varsa yazmalarına olanak tanımıştır.

Çevrimiçi hakaret edici dilin ele alınma şekli, kullanılan hakaret edici dilin tespit edilmesi ve sosyal ağ kullanıcıları üzerindeki olumsuz etkilerinin azaltılması amacıyla birkaç yaklaşımla gerçekleştirilebilir. Bu yaklaşımlar arasında kullanıcı tarafından üretilen içeriklerin gözden geçirilmesi ve hakaret edici olanların kaldırılması, hakaret içeren içeriklerin bildirilmesi, kullanıcıları böyle bir dil kullanmaktan uyarma ve otomatik tespit gibi yöntemler bulunur. Bu tez, derin öğrenme ve doğal dil işleme tekniklerini kullanarak hakaret edici dilin otomatik ve gerçek zamanlı olarak tespit edilmesini incelemektedir. Hakaret içeren içerik tespit edildiğinde, içerik otomatik olarak gizlenebilir veya daha fazla gözden geçirme için rapor edilebilir.

Başkalarına çevrimiçi olarak hakaret etmek ve saldırmak, insanların bu platformları kullanabileceği en kötü yollardan biridir. Sadece hedeflenen bireyleri etkilemekle kalmaz, aynı zamanda diğerleri için de toksik ve hoş olmayan bir ortam oluşturur. Bu istenmeyen davranış, sağlıklı ve olumlu bir çevrimiçi topluluğun oluşturulmasını zorlaştıran önemli bir zorluk oluşturur. Ayrıca, her gün internet üzerinde oluşturulan büyük veri miktarı, bunu manuel olarak düzenlemenin zorluğunu ortaya koymaktadır (Uban, 2019). Bu nedenle, hakaret edici dil tespitini otomatikleştirmemiz ve içeriği düzenlemek ve ilişkilendirilmiş riskleri azaltmak için silmek veya paylaşmalarını engellemek gibi bir adım atmamız gerekmektedir. Bu sorunun çözümüne yönelik olarak, veri bilimcileri ve doğal dil işleme (DDI) araştırmacıları zaten çözüm yolları aramaya başlamışlardır. Bu nedenle, metin bağlamında nefret söylemini tanımlamak için makine öğrenimi (ML) ve derin öğrenme (DL) tekniklerini kullanarak çeşitli sınıflandırıcı model oluşturmuşlardır. Bu modeller, hakaret içeren dil metinlerini ya kaldırarak ya da internet üzerinde yayınlanmalarını engelleyerek bu sorunu sanal yaşamımızdan kaldırmak amacıyla adım atmaktadır.

Bu tez, elde edilecek yararlı içgörüler ve sonuçlar sunmak için atılan bir dizi adımı ele almaktadır. Bu çalışma, yayınlanmış içerikteki hakaret içeren metinleri tespit etmek için farklı deneyler yapmayı içerir. Bu alandaki Arapça dil veri kaynaklarının eksikliğini dikkate alarak Arapça metin koleksiyonu toplayıp etiketleyerek, büyük bir Arapça veri kümesi oluşturmayı amaçlamaktayız. Bu tez çalışmasında, Evrişimsel Sinir Ağı (CNN), Tekrarlı Sinir Ağı (RNN) ve Uzun Kısa-Devre Bellek (LSTM) gibi farklı derin sinir ağı yapıları, Arapça veri kümesi ve daha önceki çalışmalarda toplanıp kullanılmış olan İngilizce ve Türkçe dillerindeki iki ilave veri kümesi kullanılarak karşılaştırılacaktır. Ayrıca, her dil için o dile özgü çeşitli işlemlerle eğitim verilerinin ön işlemesinin etkinliği ve önemi de araştırılacaktır. Ek olarak, basitten karmaşığa kadar farklı sinir ağı yapıları, sınıflama doğruluğu açısından değerlendirilmek için tasarlanacaktır. Elde edilen doğruluk değerinin yanı sıra, en doğru mimari seçimi, model eğitim süresi de göz önünde bulundurularak belirlenecektir.

Bu tez çalışmasında ayrıca, modelin eğitim verilerini işlem öncesinde seçmek için özellik seçimi yöntemlerinin etkinliği ve önemi de gösterilecektir. Bu yöntemler, eğitim verilerinden en ilgili özellikleri seçerek sınıflandırıcıya, işleme metin içeriklerindeki kelimeler arasındaki desenleri ve ilişkileri belirleme konusunda yardımcı olacaktır. Aynı zamanda bu yöntemler, öğrenilen özelliklerin sayısını azaltarak eğitim süresini kısıtlamaya da yardımcı olacaktır.

Bu çalışma sonunda, İngilizce, Türkçe ve Arapça veriler içeren çok dilli bir veri seti oluşturur. Her dil için farklı miktarda veri içeren dengeli bir veri seti oluşturulur. Her bir sınıflandırıcının seçilen mimarisi bu veri setine uygulanır, bu da dil arası karşılaştırmalara olanak tanır ve modellerin etkinliği hakkında kapsamlı bir bakış sunar. Doğruluk ve eğitim süresi açısından dört veri seti üzerinde sonuçları karşılaştırarak, değerli görüşler elde edilecektir.

Bu tez, farklı derin öğrenme (DL) sinir ağlarından basit mimari modellerle önişleme ve özellik seçimi yöntemlerinin etkinliğini sunmayı amaçlar. Bu sinir ağları arasında Evrişimli Sinir

Ağı (CNN), Tekrarlayan Sinir Ağı (RNN) ve Uzun Kısa Vadeli Bellek (LSTM) gibi farklı türde nöral ağlar bulunmaktadır. Ayrıca, bu çalışma RoBERTa dil modelini, gelişmiş önceden eğitilmiş bir dönüşüm modelini değerlendirme sürecine entegre eder. Temel odak, bu metodolojilerin Arapça, Türkçe ve İngilizce dillerindeki veri setleri üzerindeki etkinliğini değerlendirmektir.

Bu tezin bulguları, RoBERTa'nın çeşitli diller için güçlü bir seçenek olduğunu göstermektedir. Birçok veri seti arasında en iyi sınıflama doğruluğunu göstererek farklı diller ve görevler için iyi çalıştığını ortaya koymuştur. Bununla birlikte, CNN de iyi sonuçlar göstermekte ve RoBERTa kadar fazla bellek kaynağı gerektirmemektedir. Önceki bulgular doğrultusunda bu tez kapsamında RoBERTa ve CNN'i birleştiren bir sınıflandırıcı oluşturduk ve sınıflandırma performansını artırmayı denedik.

**ACKNOWLEDGEMENTS**

Firstly, I would like to thank my thesis supervisor, Prof. Dr. Selma Ayşe ÖZEL, for her guidance and valuable suggestions throughout my MSc thesis work. I am extremely grateful for her efforts and support in teaching me and my colleagues at Çukurova University.

I would like to express my sincere appreciation to the members of the MSc thesis jury for their insightful comments and valuable guidance.

I extend my heartfelt thanks to my late father, Eng. Muhammed Ali, for his love, guidance, and limitless support, even though he is no longer with us. I am deeply grateful for the values and encouragement he instilled in me, which have been a driving force in my academic journey.

Additionally, I want to express my sincere appreciation to my mother, Lütfiye, and my siblings for their continued love and support during this challenging time. Their presence and encouragement have been a source of strength, and I am grateful for their unwavering belief in my abilities.I would also like to thank my dear friends for their support throughout this journey.

I am truly grateful for my precious son, Ali. Despite his tender age, his innocent presence has filled my life with boundless joy and love. His smiles and laughter have been a constant source of encouragement and positivity during the process of working on this thesis.

Last but not least, I want to express my special gratitude to my beloved wife, Raghad, for her endless support and assistance in working together on creating the used datasets of this thesis and on each moment in our life.

**LIST OF TABLES**

**LIST OF FIGURES**

**SYMBOLS AND ABBREVIATIONS**

| | | |
|---|---|---|
| NLP | : | Natural Language Processing |
| ML | : | Machine Learning |
| DL | : | Deep Learning |
| NNs | : | Neural Networks |
| CNN | : | Convolutional Neural Network |
| RNN | : | Recurrent Neural Network |
| LSTM | : | Long Short-Term Memory |
| GRUs | : | Gated Recurrent Units |
| val_accuracy | : | Validation Accuracy |
| val_loss | : | Validation Loss |
| RoBERTa | : | A Robustly Optimized BERT Pretraining Approach |
| Acc | : | Accuracy |
| MLP | : | Multilayer Perceptron |

# 1. INTRODUCTION

Nowadays, it is common for people of all age groups, including teenagers, to use social media. However, it is unfortunate that we often come across a significant amount of hateful and abusive content in posts or comments on various social media platforms, such as offensive tweets on Twitter, inappropriate comments and posts on Facebook, and offensive comments on YouTube. These contents may contain elements of racism, sexism, or cyberbullying, which can have numerous negative effects on the users of these platforms (Mohaouchane et al., 2019).

## 1.1. Overview of Offensive Language

Offensive language can be defined as the use of hurtful and mean words in spoken or written form that can negatively affect other people. According to United Nations (n.d.), "Hate speech" in everyday language refers to offensive language that targets a group or individual based on their inherent characteristics (like race, religion, or gender) and may threaten social harmony. The United Nations (n.d.), defines hate speech as any form of communication, whether spoken, written, or in behavior, that attacks or discriminates against a person or group based on their identity factors such as religion, ethnicity, nationality, race, color, descent, or gender.

## 1.2. Types of Offensive Language

Offensive language is difficult to be defined clearly because of the existing differences of it from culture to another. But it still has some main types. According to Battistella (2005) there are six different types of offensive languages:

### 1.2.1. Epithets

Epithets are offensive words or phrases used to insult or belittle someone based on their race, ethnicity, religion, gender, or other personal characteristics. These words are meant to be hurtful and disrespectful.

### 1.2.2. Profanity

Profanity involves using offensive language that is considered disrespectful or vulgar. It often includes swearing or using inappropriate words to express frustration, anger, or strong emotions.

### 1.2.3. Vulgarity

Vulgarity refers to the use of crude and indecent language that is considered socially unacceptable. It involves using explicit words or expressions related to sex, body parts, or bodily functions.

### 1.2.4. Obscenity

Obscenity involves using offensive language or materials that are sexually explicit, graphic, or lewd. It goes beyond what is considered decent or appropriate in public discourse.

### 1.2.5. Cursing

Cursing is the act of invoking harm or wishing ill upon someone using specific words or phrases. Curses may be religious or non-religious in nature and are intended to bring harm or misfortune to the target.

### 1.2.6. Slang

Slang is informal language that is specific to a particular group or subculture. While not inherently offensive, some slang terms may become offensive when used to demean or insult others.

In the research of Al-Tufay and Al-Ghizzy (2022), various types of offensive words and expressions were identified, including "vulgarity, insults, epithet, taboo, obscenity, and profanity." These offensive words served different purposes, such as referring to body parts, social aspects, connotations, metaphors, sexuality, and expletives. The anonymity provided by electronic systems allowed users to express themselves freely and write whatever they had in their minds without fear of consequences.

## 1.3. Handling of Offensive Language

The treatment of online offensive language can be by several approaches aimed at detecting the used offensive language, and reducing its harmful effects on users of social networks. Such as: review user-generated contents and removing the offensive ones, reporting offensive content, warning users from using such a language, and automated detection.

This thesis studies using deep learning and natural language processing techniques to automatically detect offensive language in real-time. When offensive content is identified, it can be automatically hidden or reported for further review.

## 1.4. Deep Learning

Machine-learning technology has become a crucial part of our lives, powering various applications like web searches, social media content filtering, and personalized recommendations. Deep learning, a specific type of machine learning, is gaining popularity for these tasks (LeCun et al., 2015).

Deep learning is a type of machine learning where machines can automatically find important patterns in raw data. It depends on using neural networks inspired by the structure of the human brain. The term "deep" comes from the multiple layers in these neural networks, allowing them to learn hierarchical representations of data without human engineers having to design them.

A neural network classifier is a system of interconnected units where the input units represent terms and the output unit(s) represent categories. To classify a test document, its term weights are given to the input units. The information then flows through the network, and the output unit(s) decide which category the document belongs to (Korde, 2012).

Deep learning has been very successful in tackling difficult tasks that were challenging for artificial intelligence before. It can find complex patterns in large datasets on its own. It has outperformed other machine-learning methods in tasks like image and speech recognition, predicting drug activity, analyzing particle accelerator data, and reconstructing brain circuits. Additionally, deep learning shows promising results in natural language understanding tasks, such as topic classification, sentiment analysis, question answering, and language translation (LeCun et al., 2015).

### 1.5. Natural Language Processing (NLP)

Speech and language processing has a long history and has been studied in various fields like computer science, electrical engineering, linguistics, and psychology. These different disciplines have contributed to the development of computational linguistics, natural language processing, speech recognition, and computational psycholinguistics (Jurafsky, 2019).

According to IBM (n.d.), Natural Language Processing (NLP) is a branch of computer science and artificial intelligence that enables computers to understand text and spoken words like humans. It combines computational linguistics with statistical, machine learning, and deep learning models to process human language, including its meaning, intent, and sentiment. NLP powers various applications, such as language translation, voice-operated systems, chatbots, and speech-to-text software. It is widely used in consumer products and increasingly in enterprise solutions to improve business operations and productivity.

There are some challenges face NLP due to the complexities of human language, such as homonyms, idioms, and grammar variations. To tackle these challenges, NLP uses tasks like speech recognition, part of speech tagging, word sense disambiguation, named entity recognition, co-reference resolution, sentiment analysis, and natural language generation. These tasks help the computer accurately interpret and generate human language.

### 1.6. Text Data Mining

Text mining is a method used to get valuable information and meaningful insights from unstructured text data. Unstructured data refers to information that does not have a specific structure or format, such as articles, emails, social media posts, text in books.

Text mining is becoming increasingly important due to the abundance of electronic documents from various sources, containing unstructured and semi-structured information. Its main purpose is to extract valuable information from textual resources through tasks like retrieval, classification (supervised, unsupervised, and semi-supervised), and summarization. Text mining

involves the collaboration of Natural Language Processing (NLP), Data Mining, and Machine Learning techniques to automatically classify and uncover patterns in different types of documents (Korde, 2012).

### 1.7. Aim and Contribution of this Thesis

Manually regulating the vast amount of data generated on the internet every day is a daunting task (Uban, 2019). Hence, automating the detection of offensive texts becomes crucial. Taking appropriate action, like deleting or preventing the sharing of such content, is essential to organize platform content and mitigate associated risks.

After conducting a review of previous studies in the field of natural language processing (NLP) that cover offensive language detection in various languages, I have concluded that deep learning techniques are most suitable for my research. Specifically, these models have demonstrated acceptable performance in previous studies.

This thesis aims to present the effectiveness of preprocessing, feature selection methods with simple architecture models from different neural networks of deep learning (DL) like Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM) and their efficiency on Arabic, Turkish and English languages datasets, and a multilingual dataset which is also created that contains data in English, Turkish, and Arabic, with varying amounts of data in each language. The dataset is balanced for each language, and the selected architecture of each classifier is applied to it, allowing for cross-language comparisons and to show how these classifiers can learn features from different languages for identifying offensive language in social media websites. The models focus on achieving acceptable evaluation scores in English, Turkish, and Arabic languages. Using a multilingual model can enable social media platforms to utilize a single detection model for multiple languages, utilizing deep neural network algorithms (Ranasinghe, R., and Hettiarachchi, 2020).

To train and evaluate the models, freely available datasets are utilized for English and Turkish languages. Additionally, multiple datasets in Arabic language are combined to obtain a larger and more diverse dataset for training purposes. This will enhance the model's performance by leveraging a broader range of offensive language patterns across various languages.

Actually, there are already a lot of applied studies about detecting offensive language in social media by using deep neural networks and NLP. A significant portion of existing studies in this field has predominantly focused on the English language, disregarding the fact that social media platforms are used globally across different regions and languages. Although there are instances where published posts or tweets are written in multiple languages, the number of studies conducted for multiple languages together remains limited.

Considering this gap, the objective of this study is to develop a multilingual neural network model that can be effectively utilized for Turkish, Arabic, and English languages. By creating a model that is capable of detecting offensive language in multiple languages, aiming to contribute to the broader understanding and implementation of offensive language detection across various linguistic contexts.

## 1.8. Thesis Outline

This thesis is organized into five chapters:

**Chapter 1: Introduction**

In this chapter, we provide an overview of offensive language, its different types, deep learning, natural language processing, text mining, and the aim and contribution of this thesis.

**Chapter 2: Preliminary Works**

Here, we present a summary of relevant studies that focus on detecting offensive language using neural network classifiers.

**Chapter 3: Material and Method**

This chapter introduces the training datasets and the methods used for data processing. We describe the datasets, the programming language and libraries utilized, the editor used, feature selection methods, and the neural network classifiers employed.

**Chapter 4: Results and Discussions**

In this chapter, we present and discuss the results of our experiments.

**Chapter 5: Conclusion**

The final chapter concludes the study, summarizes key findings, and suggests avenues for future research related to this thesis.

## 2. PRELIMINARY WORK

In this section, we will delve into several studies that have employed deep neural network models for detecting offensive language. We will emphasize their key findings concerning different datasets used in the research. By doing so, we aim to gain a comprehensive understanding of the effectiveness of these models in identifying offensive language across various contexts.

### 2.1. Studies on English datasets

Gambäck and Sikdar (2017) focused on hate speech classification using CNN models. They leveraged feature embeddings derived from word embeddings and character n-grams. Experiments on an English Twitter hate-speech dataset show achieving superior precision and F1-scores compared to Logistic Regression. The CNN model utilizing word2vec embeddings without character n-grams achieved the best results in terms of precision, recall, and F1-score. The distribution of the used dataset was as Racism 91, Sexism 946, Both (racism & sexism) non-hate-speech 5600, and Total 6655. As a result, the model based on word2vec embeddings performed best, with higher precision than recall, and a 78.3% F-score.

Pitsilis (2018) makes a contribution by developing a deep learning architecture that can effectively classify text in terms of hateful content. The authors used a Long-Short-Term-Memory (LSTM) model to classify texts. They used a dataset of tweets that have been labeled as either neutral, sexist, or racist. They evaluated their model on a held-out test set and achieve an accuracy of 84.2%. The authors also investigate the impact of incorporating information about the users' past history on the classification performance. They found that this information can improve the performance of the model by up to 2%.

In the study conducted by (Goel et al, 2019), they employed specific methods to gather data for the OffensEval dataset. These methods were detailed in a publication by Zampieri and others in 2019. The data they collected came from Twitter and included the use of an Offensive/Profane Word List with over 1,300 English terms. Their primary goal was to assess the offensiveness of tweets using a Long Short-Term Memory (LSTM) model with Word Embeddings. The researchers highlighted the efforts of major companies like Facebook, Google, and Twitter to combat offensive language on their platforms. However, they pointed out that traditional manual methods were not effective, mainly due to their inability to handle large amounts of data. This led them to recognize the importance of integrating machine learning into their project to address offensive language on social media platforms. They believed that combining machine learning with human efforts could be highly beneficial. In their paper, the researchers also acknowledged the significance of having a substantial amount of data for reliable results. They expressed their interest in future work, aiming to incorporate a new dataset and collect more data. They believed that a more extensive and diverse dataset would improve the accuracy and efficiency of their system. The research produced promising

results, with the LSTM approach achieving an accuracy of approximately 88.75% in detecting offensive language.

Sadiq et al (2020) aimed to identify aggression detection in tweets using a combination of CNN-LSTM and CNN-BiLSTM models. Their proposed model achieved a high accuracy of 92% in detecting cyber-aggressive behavior on a cyber-troll dataset. They developed a framework that involved feature extraction, feature selection, and classification using dense layers of Multilayer Perceptron. The proposed Deep Neural Network (DNN) is composed of three modules: 1.Feature Extraction, 2.Feature Selection, 3.Classification. In addition, the dataset was preprocessed by: 1) Removal of Stop words, 2) Removal of punctuation and numbers, 3) Lowercase, 4) Tokenization, methods. They used four hidden layers with Rectified Linear Unit (Relu) activation function in the first three layers and sigmoid activation function is used in the output layer. Number of neurons of the output layer are 2, the chosen batch size is 128, learning rate is 1e3 and dropout is 0.2. They used Binary cross entropy as a loss function. The sentiment analysis process began with manually annotating the tweets in the dataset. The tweets were then standardized by removing usernames, hashtags, URLs, and other non-standard characters. The tweets were then split into a 75% training set and a 25% test set. Features were then extracted from the tweets. The machine learning model was then trained on the training set to predict the sentiment of the tweets. Finally, the model was evaluated on the test set to see how well it predicted the sentiment of the tweets. Statistical results proved that their proposed model performed best and detects aggressive behavior with 92% accuracy.

In the research paper (Wei et al., 2021), the authors address the pressing issue of toxic online speech, focusing on automatically classifying tweets into three categories: Hate, offensive, and neither. The study begins with experiments to build Bidirectional Long Short-Term Memory (BI-LSTM) models using empty embeddings and pre-trained Glove embeddings, exploring transfer learning techniques with models like BERT, DistilBert, and GPT-2 for hate speech detection. After hyperparameter tuning, the best-performing model (BI-LSTM) achieves over 92% accuracy on test data. The research employs a public tweet dataset of 24,783 tweets, categorized as Hate Speech, Offensive Language, or Neither by CrowdFlower users. Data preprocessing includes steps like converting text to lowercase, removing special characters, handling hashtags, eliminating URLs, and converting emojis into text. The study compares deep learning models, including LSTM and transformers like BERT, while also introducing a custom Bi-Directional LSTM (BI-LSTM) architecture and assessing its performance against transfer learning from pre-trained transformer-based models, all aimed at advancing the detection and mitigation of harmful online speech.

### 2.2. Studies on Arabic datasets

In their study, Mohaouchane et al (2019) explored the effectiveness of different neural network models for hate speech detection. They applied CNN, Bi-LSTM, Bi-LSTM with attention mechanism, and a combined CNN-LSTM architecture on a labeled dataset of Arabic YouTube

comments. The dataset, collected from popular controversial YouTube videos about Arab celebrities, consisted of 15,050 comments collected by Alakrot et al (2018). To preprocess the data and optimize the models, the researchers utilized Arabic word embedding and employed Bayesian optimization techniques to tune the hyperparameters of the neural network architectures. The CNN-LSTM architecture achieved the highest recall rate of 83.46%, followed by CNN with 82.24%, Bi-LSTM with attention mechanism with 81.51%, and Bi-LSTM with 80.97%. In terms of accuracy and precision, the CNN model outperformed the others. These findings demonstrate that the CNN-LSTM architecture, along with the CNN model, holds promise for hate speech detection in Arabic YouTube comments. The utilization of Arabic word embedding and careful hyperparameter optimization through Bayesian optimization further enhanced the performance of the models.

The study of Chowdhury et al (2019) presents LSTM + CNN + NODE2VEC (ARHNet), a novel approach for detecting religious hate speech in Arabic social media, combining Arabic Word Embeddings and Social Network Graphs to emphasize community interactions. The dataset, collected from Twitter, includes 6,000 training tweets and 600 testing tweets, with 1,685 labeled as hateful and 2,265 as non-hateful, after thorough crowdsourced labeling. Preprocessing involves Arabic-specific normalization, diacritic and punctuation removal, emoji, non-Arabic character removal, and stemming. ARHNet utilizes social network graphs to represent user interactions, generating low-dimensional user embeddings through Node2Vec. These "Interactions" combined with word embeddings and sentence representations feed into a dense layer for final hate speech classification. Compared to baseline models, ARHNet consistently outperforms, achieving an F1-score of 0.78 and demonstrating improved recall and area under ROC values, effectively enhancing religious hate speech detection in Arabic social media, aligning with human annotator performance.

The research of (Alsafari et al., 2020) studies an approach to detect hate/offensive speech based on a fine-grained labeled textual Arabic corpus for two-class, three-class, and six-class classification, using CNN and BiLSTM. The trained single classifiers and their ensemble models using trained FastText word embedding and two pre-trained contextual word-embedding models, MBert and AraBert. Alsafari et al. (2020) made an evaluation for all prediction processes, using all the trained models on testing datasets, The optimal ensemble models outperform the optimal single models with an improvement of 2.2%, 4.8% and 3.26% for two-class, three-class and six-class tasks. The average-based ensemble approach was found to be the best performing, as it returned F-scores of 91%, 84%, and 80% for two-class, three-class and six-class prediction tasks, respectively.

In their paper, Haddad et al. (2020) participated in the OSACT4 shared task on offensive language detection (Mubarak et al., 2020). Their goal was to detect offensive language and hate speech using deep neural networks. They presented two models: CNN and Bidirectional Gated Recurrent Unit (Bi-GRU), both enhanced with attention layers. To evaluate their models, they utilized a dataset that was initially introduced at OffensEval 2020 by Zampieri et al. (2020). The dataset consisted of 10,000 tweets, with only 5% labeled as hate speech, 19% as offensive, and the

remaining 81% as inoffensive. The dataset was divided into training data (70%), validation data (10%), and test data (20%). Among the models they developed, the bidirectional GRU model augmented with an attention layer demonstrated the best performance. It achieved an impressive F1 score of 0.859 for the task of offensive language detection and an F1 score of 0.75 for the task of hate speech detection. This study contributes to the field of offensive language detection by showcasing the effectiveness of bidirectional GRU model with attention, in accurately identifying offensive language and hate speech. The utilization of attention mechanisms allows the models to focus on relevant information and make more precise predictions.

In the study (Essefar, 2023), the authors addressed the issue of automatic detection of offensive tweets in the Arabic language. They introduced the Offensive Moroccan Comments Dataset (OMCD), the first dataset for offensive language detection in the Moroccan dialect. The dataset was collected from YouTube comments and underwent thorough preprocessing. Their annotation process involved categorizing each comment as offensive or non-offensive, and the annotators verified whether the comments were written in the Moroccan dialect. The annotation results yielded 4,304 offensive comments and 3,720 non-offensive comments. They also evaluated various machine learning and deep learning models on the OMCD dataset, both with and without the presence of emojis. The results indicated that transformer-based models MARBERT outperformed other models, achieving an accuracy of 84.17% and they got lower accuracies using the other classifiers such as: 78.82% using LSTM, 77.63% using BiLSTM that they are related to our study. Interestingly, the inclusion of emojis had a slightly negative impact on the transformer-based models' performance, except for CAMeLBERT, which showed a minor improvement.

In the study of Khezzar et al., (2023), The authors tackle the escalating problem of hate speech on social media, notably Twitter, within the Arab region. They introduce the arHateDetector framework, designed to identify hate speech in Arabic tweets, encompassing both standard and dialectal Arabic. To bolster this framework, the authors amalgamated multiple Arabic hate speech datasets, resulting in a substantial compilation known as "arHateDataset." The research delves into assessing the performance of various machine learning (ML) and deep learning (DL) models, including Logistic Regression, Support Vector Classifiers, Naive Bayes, Decision Trees, Random Forests, K-Nearest Neighbors, Convolutional Neural Networks (CNN), and AraBERT (an Arabic adaptation of BERT). The experimental findings underscore AraBERT's exceptional accuracy, reaching 93%, thereby surpassing other models, including CNN.

### 2.3. Studies on Turkish datasets

The study of Karayiğit et al (2021) focuses on detecting abusive comments in Turkish on Instagram. They propose a new dataset called Abusive Turkish Comments (ATC) containing 10,528 abusive and 19,826 non-abusive comments collected from tabloid and sports accounts. They evaluated different classification models, including Convolutional Neural Network (CNN), Naive

Bayes, Support Vector Machine, Decision Tree, Random Forest, Logistic Regression, Adaptive Boosting (AdaBoost), and eXtreme Gradient Boosting (XGBoost). The results show that the CNN model performed the best, achieving high accuracy (Micro-averaged F1-score: 0.974, Macro-averaged F1-score: 0.973, Kappa-value: 0.946) on the oversampled ATC dataset. The study also discusses the significance of using deep learning techniques for sentiment analysis on social media comments.

In this study (Coban et al., 2023), researchers addressed the challenging problem of Cyberbullying Detection (CBD) within the context of Turkish users on the Facebook social media platform. They adopted a comprehensive approach that involved the exploration of multiple Machine Learning (ML) and Deep Learning (DL) methods to select the most effective estimator. These methods included traditional ML classifiers such as Logistic Regression (LR), Multinomial Naive Bayes (MNB), and Support Vector Machines (SVM), as well as DL techniques like CNN, Recurrent Neural Networks (RNN), and the powerful BERT model. The researchers meticulously evaluated these methods on various datasets, including their own Facebook dataset (CF-B) and cross-domain evaluation datasets. The results demonstrated the superiority of the BERT model, achieving a remarkable F1-score of 0.928 on CF-B. This performance not only outperformed the traditional ML classifiers but also showcased cross-domain generalization, exhibiting impressive results across various OSN domains. Moreover, they conducted user behavior profiling to gain insights into CB characteristics among Turkish Facebook users, revealing intriguing trends related to gender, age range, and the rise in CB incidents over the years.

### 2.4. Studies on multilingual datasets

Zhang and Tepper (2018) proposed a novel method that combined convolutional and gated recurrent networks, optimized with dropout, pooling layers, and elastic net regularization for better learning accuracy. Their deep neural network was trained on seven different datasets, where most of them contains thousands of tweets on the subjects of religion and refugees, outperforming SVM, SVM+CNN, and previous state-of-the-art results across all datasets.

Elouali et al (2019) employed Convolutional Neural Networks (CNN) for hate speech detection on a multilingual Twitter dataset. Classifying tweets written in seven different languages to hate speech or none are performed by using CNN and character level representation. They experimented with various architectures trying to get better results, achieving an accuracy of 0.8893 using 3 convolutional layers and 100 filters on a five-language dataset. Additionally, their model achieved 0.8300 accuracy on a seven-language dataset. They used collected datasets in different languages and combine them to get their multilingual dataset, that has :Arabic, Italian, Portuguese, Indonesian, English, German and Hindi-English languages, with a total 33727 tweets. They made some data cleaning such as: HTML decoding, deleting mentions from tweets, deleting URL links

and Hashtags, finally they deleted diacritics from the tweets in Arabic language which can be important for my study.

In her work, Kumari (2020) focused on multilingual offensive language identification in social media as part of the OffensEval 2020 task by Zampieri et al. (2020). The task involved three subtasks: A) Identifying offensive or non-offensive tweets in Arabic, Danish, English, Greek, and Turkish languages, B) Detecting targeted or untargeted offensive tweets in English, and C) Categorizing targeted offensive tweets into individual, group, or other classes in English. To address these subtasks, the author used two models: a pre-trained BERT model and a BiLSTM model with an attention mechanism (Attn-BiLSTM). Their analysis showed that the Attn-BiLSTM model outperformed BERT in detecting offensive language in Arabic and Greek, while BERT performed better in identifying offensive language for Danish, English, and Turkish, as well as offense-type categorization and target identification in English. The author also combined data from multiple languages to train their UML_BiLSTM model, which exhibited faster training times, although its accuracy was slightly lower compared to BERT and BiLSTM. The results using BERT-multi-cased (BERT_MC) for different languages were as follows: Arabic-A (F1-score: 0.8172), Danish-A (F1-score: 0.7672), English-A (F1-score: 0.9060), English-B (F1-score: 0.6711), English-C (F1-score: 0.6054), Greek-A (F1-score: 0.8304), and Turkish-A (F1-score: 0.7481). For the BiLSTM Model (Attn-BiLSTM), the results were: Arabic-A (F1-score: 0.8447), Danish-A (F1-score: 0.7000), English-A (F1-score: 0.8970), English-B (F1-score: 0.6200), English-C (F1-score: 0.5774), Greek-A (F1-score: 0.8984), and Turkish-A (F1-score: 0.7425).

The study that made by (Tanase et al, 2020) focuses on the challenging problem of detecting offensive language in social media, particularly on Twitter, in multiple languages including English, Arabic, Danish, Greek, and Turkish. They employed Transformer-based models, such as BERT, mBERT, RoBERTa, XLM-RoBERTa, and ALBERT, which were fine-tuned using various datasets. The study was part of the Offenseval 2020 shared task, where they achieved competitive rankings in the competition. The methodology section outlines the authors' approach, starting with a baseline non-neural network classifier and then transitioning to Transformer-based models like BERT, RoBERTa, XLM-RoBERTa, and ALBERT. They discuss fine-tuning these models on different datasets, including the Offenseval 2020 dataset, OLID dataset, and HASOC dataset. Data preprocessing steps, including tokenization and translation, are also described. In the results of their submissions, they compare their performance to the top-performing teams in the Offenseval 2020 Subtask A competition (Subtask A, the primary objective is to determine whether tweets contain offensive language or not. This task can be simplified as a binary classification problem, where the goal is to categorize each tweet into one of two classes: offensive or non-offensive.). Their models achieved notable F1-scores on the competition's test sets, demonstrating their effectiveness in identifying offensive language in various languages. Specifically, F1-scores were as follows: 91.05% for English, 82.19% for Arabic, 73.80% for Danish, 81.40% for Greek, and 77.89% for Turkish.

These scores showcase the robustness of approach across different languages, with English and Arabic achieving particularly high accuracy in offensive language detection. In the conclusion, the authors highlight the success of using fine-tuned Transformer-based models for offensive language detection in multilingual tweets. They emphasize the potential for further improvement with larger datasets for non-English languages and transfer learning approaches.

### 2.5. Studies on other language datasets

Huynh et al (2019) conducted research on detecting hate speech in Vietnamese. They employed Bi-GRU-LSTM-CNN models on a corpus containing 20,345 of human-labeled comments/posts for training and 5,086 for public-testing. The model achieved a promising F1-score of 70.576%. The created model was based on neural networks, classifies texts in social media written to HATE, OFFENSIVE, or CLEAN. The used dataset contains posts or comments from the social network Facebook which are annotated with three different classes (Hate, Offensive and Clean). After trying three different models they found that the BiGRU-LSTM-CNN achieved the best performance made some data cleaning such as: HTML decoding, deleting mentions from tweets, deleting URL links and Hashtags, finally they deleted diacritics from the tweets in Arabic Language which can be important for my study.

In summary, these studies underscore the importance of using advanced neural network designs and applying different text analysis techniques to achieve improved precision in identifying offensive language. These results stress the value of carrying out a range of tests to create models that perform exceptionally well in this domain. By exploring different strategies, researchers have the opportunity to improve the efficiency of hate speech detection systems, thereby playing a role in providing safer and more welcoming online spaces.

In this thesis, we address the scarcity of resources for Arabic dialects by creating a substantial Arabic dataset that encompasses diverse linguistic contexts on social media platforms. Additionally, we create a multilingual dataset with a unique focus on achieving class balance across languages, facilitating cross-lingual research. Our study explores streamlined neural network architectures, emphasizing simplicity and resource efficiency. We employ feature selection methods to optimize model efficiency, highlighting the potential for resource-efficient offensive language detection systems. Our research offers insights into the use of RoBERTa and XLM-RoBERTa models, considering batch size and preprocessing, demonstrating their effectiveness and adaptability. We also introduce a novel ensemble classifier, RoBERTa-CNN, which combines RoBERTa's language modeling capabilities with CNN's feature extraction, emphasizing the complementary strengths of different deep learning architectures. Overall, our study contributes to the field by offering a fresh perspective and enriching resources for future research in offensive language detection.

## 3. MATERIAL AND METHOD

In this section of the thesis study, I will provide a comprehensive explanation of the datasets utilized and the steps taken to process them for offensive language detection. Due to the scarcity of large datasets, particularly in the Arabic language, it is crucial to create a substantial dataset in Arabic. Furthermore, two additional datasets in Turkish and English will be used for the research.

### 3.1. Datasets

The used dataset is very important factor in deep learning process, and each detail of it really effects on the resulted trained model, such as: type of data, size of it and the balance of the class distribution, all of that has a direct effect on the performance of the classifier. In our study we use three different datasets, in English, Turkish and Arabic languages. The datasets are explained in the below paragraphs.

#### 3.1.1. English Dataset

The English dataset was created by merging two prepared datasets: OLID and SOLID. OLID contains 860 tweets and was used as the official dataset for OffensEval 2019, a shared task competition on detecting and classifying offensive language in social media text (Zampieri et al. 2019), SOLID contains 5993 records and was used as the official dataset for OffensEval 2020 (Zampieri et al., 2020). Both datasets were collected from various social media platforms to be used in similar studies.

Our created English dataset with name "En_Detecting Multilingual Offensive Language in Social Media.xlsx" contains as a total 6853 records (3242 offensive and 3611 non-offensive) with 47.31% offensive texts. Figure 3.1 shows the class distribution of the used data set.

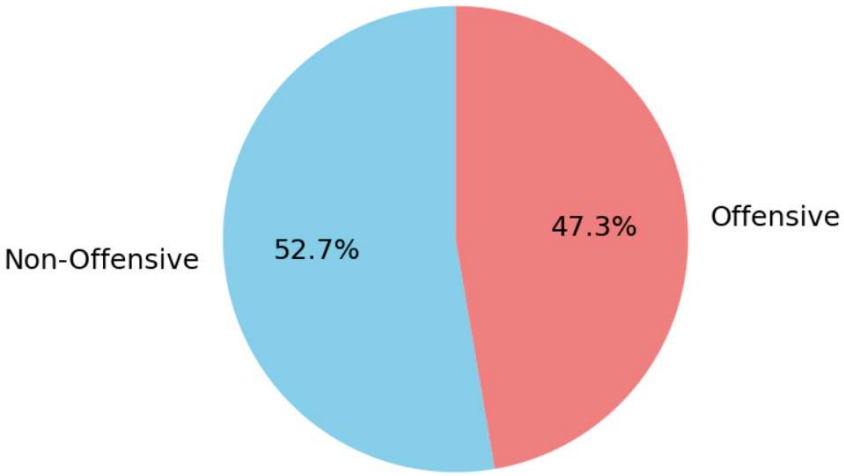


Figure 3.1. Distribution of Offensive vs. Non-Offensive English text

The English dataset contains 112,175 words in total with 21,227 unique words. The average length of the texts is 16.4 words, with a minimum of 1 word and a maximum of 58 words. The dataset is divided into two sets: En_train_dataset.xlsx and En_test_dataset.xlsx. 20% of the dataset is designated as the test set to assess the model's performance.

### 3.1.2. Arabic Dataset

Because of the lack of the collected Arabic dataset and the small number of the records in all existing datasets that were collected and used in another researches, and because of the huge difference of the dialects that are spoken in the Arabic countries, we preferred to cooperate with Raghad BİRECİKLİ to collect as possible as of data and combine prepared datasets labeled and unlabeled ones, so our Arabic dataset was created by combining different Arabic datasets that had already collected from tweets, posts and comments from different social media platforms, by the way the big part of them was a folder that has 150 files of collected comments that were unlabeled. We used three main datasets to form this dataset:

1. "A Multi-Platform Arabic News Comment Dataset for Offensive Language Detection" containing 4000 comments collected from Twitter, Facebook, Instagram, and YouTube by Chowdhury et al, (2020).
2. "Arabic Sentiment Twitter Corpus" with 58,000 Arabic tweets (SAAD, 2020).
3. "L-HSAB: A Levantine Twitter Dataset for Hate Speech and Abusive Language" consisting of 57,058 tweets collected by Mulki et al, (2019).

In order to create a comprehensive Arabic dataset for binary classification, we combined the above mentioned datasets and gathered additional data from various social media platforms. our dataset contains different colloquial Arabic dialects such as Syrian, Lebanese, Algerian, Moroccan, Jordanian, Palestinian, Egyptian, Tunisian, Saudi, Yemeni, and Iraqi.

To ensure the reliability of the dataset, we eliminated duplicate entries, filtered out empty or noisy comments, and assigned labels to records that were previously unlabeled. The resulting dataset comprises a total of 217,082 records, with 107,615 classified as offensive and 109,467 as non-offensive, making up 49.57% of offensive texts.

The dataset contains a total of 2,771,946 words, with 379,752 unique words. On average, the text length is 12.8 words, with the shortest text consisting of 1 word and the longest reaching 628 words. The dataset is divided into two distinct sets: ArFull_train_dataset.xlsx and ArFull_test_dataset.xlsx. To assess the model's performance, 20% of the entire dataset is designated as the test set.

Because of the big difference between the average and the maximum length we make some statistics so we can find that 99.02% of data has less than 70 words in text column so we eliminate

rows that have more than 70 words in text trying to obtain higher accuracy and less need of process time and hardware resources. So, the new statistics of the dataset are 214,994 total records (105,815 offensive and 109,179 non-offensive) with 49.22% offensive texts. Figure 3.2 shows the class distribution of the selected subset of the dataset.

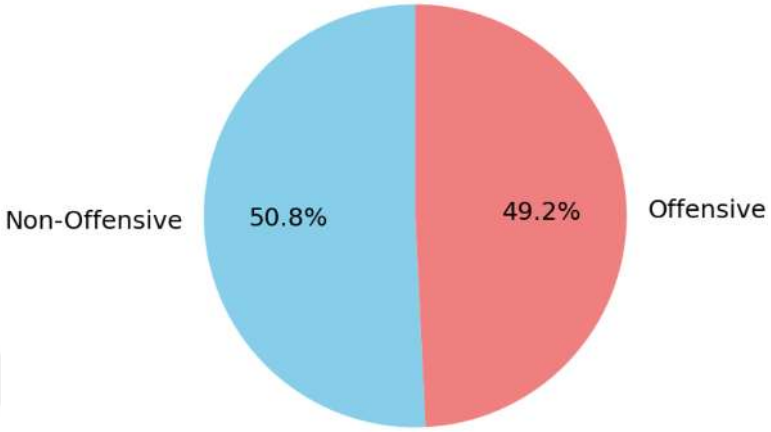


Figure 3.2. Distribution of Offensive vs. Non-Offensive Arabic texts in the selected dataset

The count of the whole words in the dataset is 2,540,333 with 358,635 unique words, the average of the length of texts is 11.82, while the minimum is 1 and the maximum is 70 words. The dataset is divided into two distinct sets: 2ArFull_train_dataset.xlsx and 2ArFull_test_dataset.xlsx. To assess the model's performance, 20% of the entire dataset is designated as the test set.

### 3.1.3. Turkish Dataset

The used Turkish dataset was taken from an archive contains Turkish offensive language corpus as used in OffensEval 2020 shared task shared task by Zampieri et al. (2020).

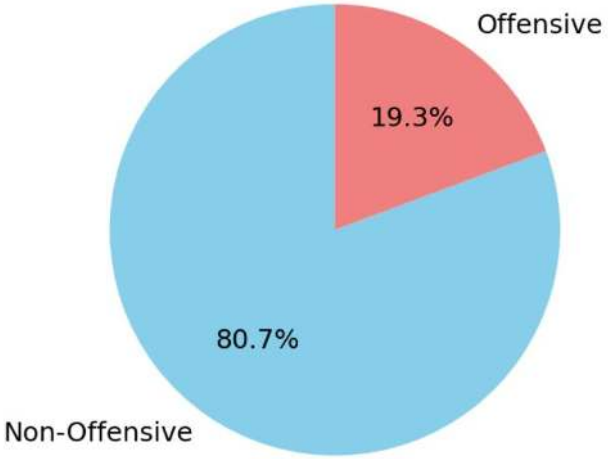


Figure 3.3. Distribution of Offensive vs. Non-Offensive Turkish texts

The dataset contains as a total 31,263 records (6,041 offensive and 25,222 non-offensive) with 19.32% offensive texts, as shown in Figure 3.3.

After finding the average and maximum length of each record in the dataset we find that there is a big difference between them, as in the case of the Arabic dataset so we group the rows depending the number of words in text column and we got as follow :

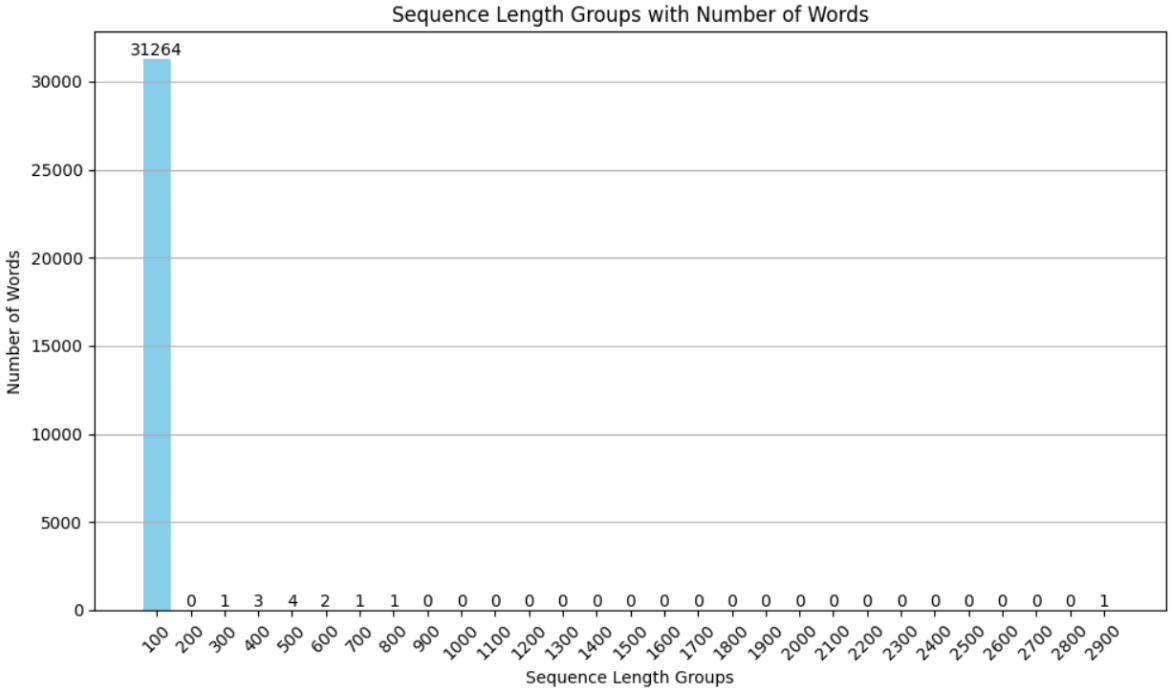


Figure 3.4. Text Group Depending the Number of Words

The Figure 3.4 shows that the majority of the records has at most 100 words, so we eliminate all rows that have more than 100 words in text since they do not represent a noticeable rate in forming the dataset. Trying to get better training and testing steps. We obtain the new statistics as follows: The count of the whole words in the dataset is 483,124 with 130,702 unique words, the average of the length of texts is 15.45, while the minimum is 2 and the maximum is 82.

The dataset is divided into two distinct sets: TrFull_train_dataset.xlsx and TrFull_test_dataset.xlsx. To assess the model's performance, 20% of the entire dataset is designated as the test set.

### 3.1.4. Balanced Turkish Dataset

According to the resulted statistics of the used Turkish dataset we notice that it was imbalanced dataset so we create a new Turkish dataset from the previous one by taking all the offensive labeled that are 3224 records as 49.8% of the created new dataset, trying to obtain a balanced dataset.

The created dataset with the name TrTrain+Test.xlsx has 6,477 records (3,224 offensive and 3,253 non-offensive) with 49.78% offensive texts, as shown in Figure 3.5.

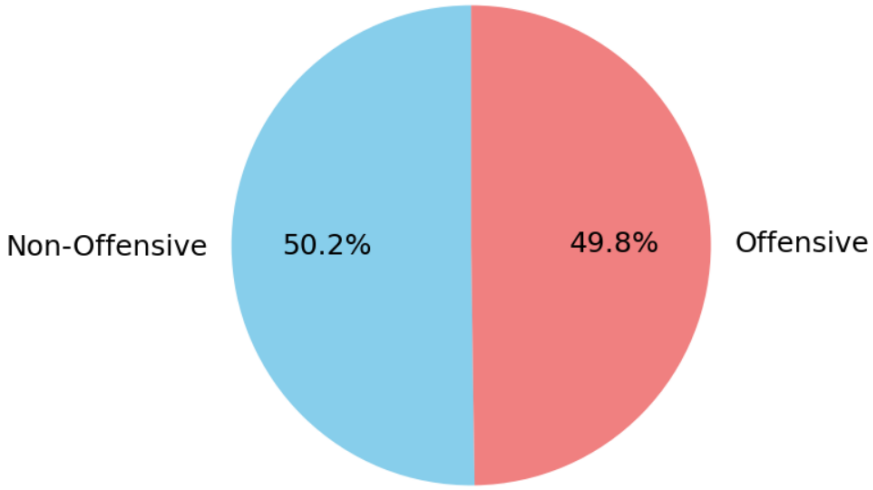


Figure 3.5. Distribution of Offensive vs. Non-Offensive balanced Turkish datasets

The count of the whole words in the dataset is 106,282 with 41,875 unique words, the average of the length of texts is 16.4, while the minimum length is 4 and the maximum is 50 words. This dataset is categorized into two separate sets: Tr_train_dataset.xlsx and Tr_test_dataset.xlsx. To evaluate the model's effectiveness, 20% of the entire dataset is allocated as the test set.

### 3.1.5. Multilingual Dataset

In this thesis we also tried to show the effectiveness of the deep neural networks on detecting offensive language from multilingual textual contents. Although in the different social media platforms there are some texts contains different language written by some users. So, a multilingual dataset is created namely (multilingualDataset.xlsx).

The created dataset has 43,329 records as total, (21,465 offensive and 21,864 non-offensive) with 49.54% offensive texts, as shown in Figure 3.6.

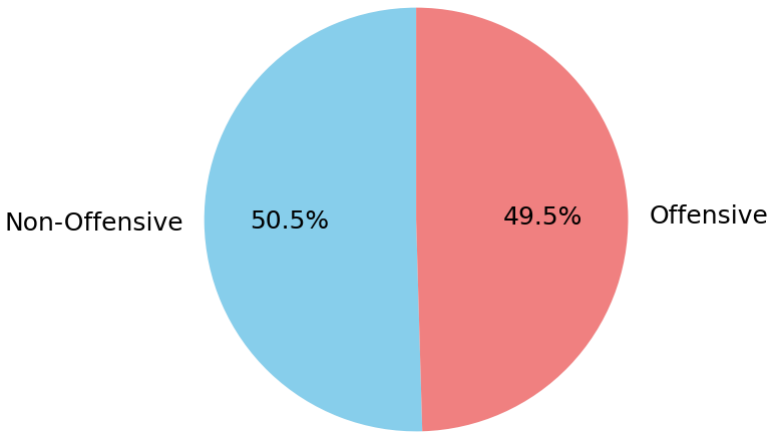


Figure 3.6. Distribution of Offensive vs. Non-Offensive multilingual dataset

The dataset contains a total of 583,740 words, out of which 154,402 are unique. The average text length is 14.22 words, ranging from a minimum of 1 word to a maximum of 70 words. This dataset is categorized into two distinct sets: Train_dataset.xlsx and Test_dataset.xlsx. To evaluate the model's effectiveness, 20% of the entire dataset is allocated as the test set. The distribution of the used languages in the selected dataset is shown in Figure 3.7 and Figure 3.8.

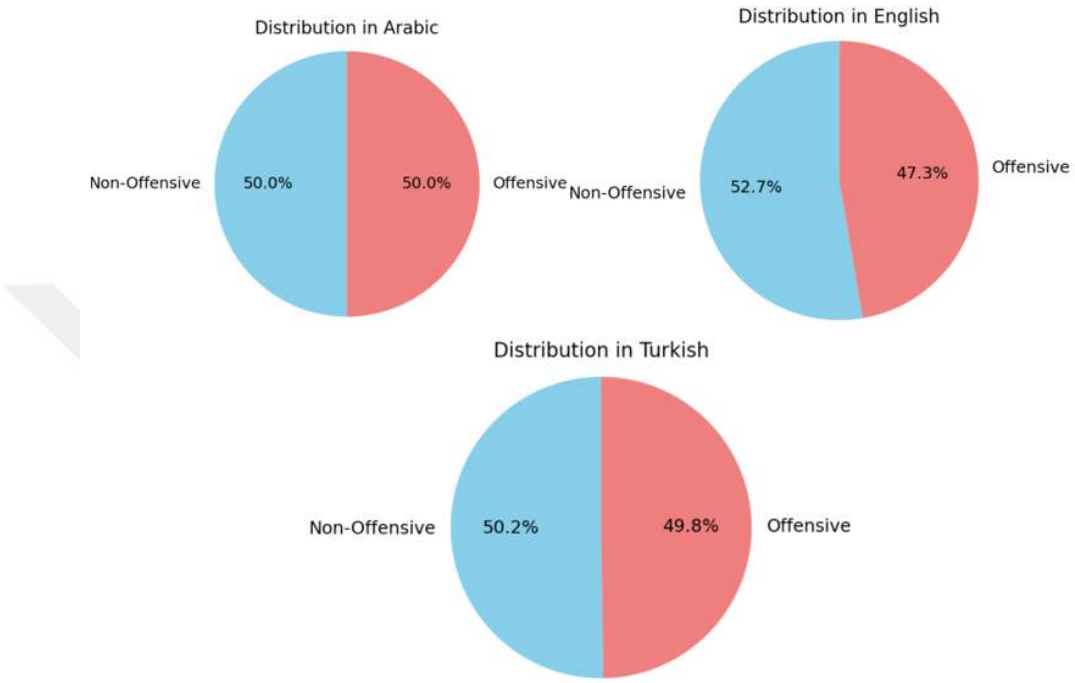


Figure 3.7. Distribution of Offensive vs. Non-Offensive Classes of each Language

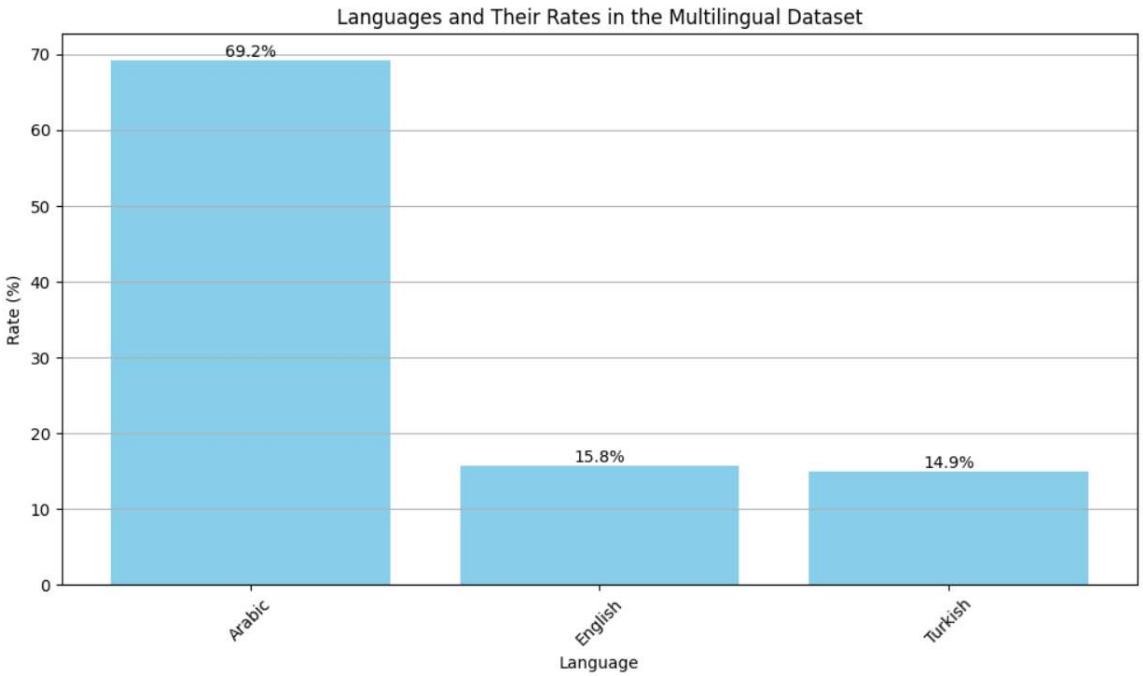


Figure 3.8. Languages' Rates in the multilingual dataset

### 3.2. Programming Language and Libraries

#### 3.2.1. Python

Python is a versatile programming language known for its simplicity and readability. It was created by Guido van Rossum in the late 1980s as a replacement for ABC. Python 0.9.0 was released in 1991, and it has since evolved into a powerful tool for both small and large projects (Wikipedia, 2023). Python's design philosophy emphasizes clear and logical code, which is achieved through features like indentation and object-oriented programming. It supports various programming paradigms, including structured, object-oriented, and functional programming. Python is garbage-collected, dynamically typed, and offers Unicode support. Notably, Python 3.0, introduced in 2008, brought significant updates but lacked full backward compatibility with previous versions. Throughout its development, version 3.8 was widely used.

#### 3.2.2. Visual Studio Code

Visual Studio Code (VS Code) is a powerful code editor that is lightweight and runs on computer. It works on Windows, macOS, and Linux. It comes with built-in support for popular programming languages like JavaScript, TypeScript, and Node.js (InfoWorld, 2023). It is more useful from other editors because of its wide range of extensions. These extensions add extra features and support for different programming needs. For machine learning tasks, there are extensions available that allow us to work with popular machine learning frameworks like TensorFlow and PyTorch.

#### 3.2.3. Jupyter Notebook

The Jupyter Notebook is an open-source web application that enables the creation and sharing of documents containing live code, equations, visualizations, and text. It serves a wide range of purposes, including data cleaning, numerical simulation, statistical modeling, data visualization, and machine learning. Jupyter Notebook provides an interactive computational environment through its web-based interface, allowing users to create Jupyter notebook documents. The project aims to develop open-source software, open-standards, and services for interactive computing across multiple programming languages (Bharath, 2020).

#### 3.2.4. Scikit-learn

Scikit-learn is a free and open-source library used for analyzing data and is widely regarded as the best choice for Machine Learning (ML) in the Python programming language (Activestate, 2022). It provides various important concepts and features:

i) Algorithmic decision-making methods: Scikit-learn offers different methods for making decisions based on data patterns,
ii) Predictive analysis algorithms: The library supports a range of algorithms for predictive analysis, starting from simple linear regression to more complex techniques like neural network pattern recognition.
iii) Compatibility with other libraries: Scikit-learn seamlessly works with other popular Python libraries like NumPy (for numerical computing), pandas (for data manipulation and analysis), and matplotlib (for data visualization).

This compatibility enhances the overall functionality and makes it easier to integrate with existing data analysis workflows.

### 3.2.5. Pandas

Pandas is a Python software library designed for data manipulation and analysis (Wikipedia, 2023). It provides data structures and operations for working with numerical tables and time series. The library is free and released under the BSD license. The name "pandas" comes from "panel data," Its main object, the DataFrame, is a two-dimensional tabular data structure that allows for quick and expressive data manipulation. Pandas is widely used in machine learning applications and offers various features such as data import/export, sorting, resizing, slicing, grouping, and cleaning.

### 3.2.6. NumPy

NumPy is an essential library for scientific computing in Python (NumPy, n.d.). It provides a powerful tool called an ndarray, which is a multi-dimensional array capable of handling various types of mathematical operations efficiently. Overall, NumPy provides a foundation for performing scientific computations in Python by offering a specialized array object and an extensive collection of functions. It enables efficient handling of large datasets, supports advanced mathematical operations, and serves as a building block for numerous scientific and mathematical Python-based packages.

### 3.2.7. TensorFlow

TensorFlow is an open-source library developed by Google for deep learning and machine learning applications (Banoula, 2023). It supports the creation of complex neural networks and can handle large volumes of data efficiently. TensorFlow accepts data in the form of multi-dimensional arrays called tensors. It uses data flow graphs, which are collections of nodes and edges, to perform computations. It is designed to handle large-scale numerical computations and can be distributed across multiple computers, making use of GPUs for faster processing. It is a popular choice for AI and machine learning projects.

### 3.2.8. Keras

According to Keras (n.d.), Keras is a popular deep learning API, that is implemented in Python and operates on TensorFlow, a machine learning platform. Its primary objective is to facilitate rapid experimentation in deep learning models. It is also known for its simplicity, flexibility, and powerful capabilities.

Important organizations and companies such as NASA, YouTube, and Waymo have utilized Keras for their machine learning needs. Serving as the high-level API for the TensorFlow platform, Keras offers an accessible and highly productive interface for tackling machine learning problems, particularly those involving modern deep learning techniques. It provides crucial abstractions and foundational components that expedite the development and deployment of machine learning solutions while emphasizing fast iteration.

## 3.3. Method

### 3.3.1. Dataset Preprocessing Procedures

In this section, we explain the preprocessing procedures executed using the Python programming language and the required libraries across all employed datasets.

The preparatory phase of the data processing entails several steps, each tailored to address the unique linguistic differences of the used languages' datasets. Initially, the dataset is read from Excel files. To ensure data completeness and consistency, rows within the 'Text' column are methodically dropped if they contain missing values. Subsequently, the textual content is uniformly converted to a standardized format known as a "string".

Central to the data preprocessing framework are distinct functions designed to normalize the text, these functions have some differences in their contents. For Arabic dataset, a normalization process encompasses the substitution of particular characters because that they represent the same character in different shapes, the elimination of diacritical marks, and the conversion of Arabic digits to English equivalents. Given that Arabic language letters do not have lowercase forms, a procedure has been implemented to ensure uniformity across the English and Turkish datasets. In these cases, text is converted to lowercase. Additionally, for all datasets, a normalization process is applied. This involves the removal of extraneous elements such as URLs, mentions, hashtags, and non-alphanumeric characters.

The distinctiveness of the Turkish language is addressed through additional preprocessing steps, where dotted characters are systematically converted to their dot less counterparts. Furthermore, language-specific stopwords are meticulously eradicated to streamline the text, and stemming procedures are judiciously applied. The stemming technique in Turkish employs the Porter Stemmer by (Porter, 2006), while the Snowball Stemmer by (Porter, 2014) is utilized for other languages.

The culmination of these tailored preprocessing functions is encapsulated within a comprehensive 'preprocess' function. This versatile function orchestrates the normalization, stopword removal, and stemming processes based on the language of the textual content. To apply this data preparation, the 'Text' column within both the training (train_df) and testing (test_df) datasets undergoes a transformative process, culminating in refined and linguistically harmonized text data. The result of these extensive operations is an optimally prepared dataset primed for subsequent analysis and machine learning endeavors.

For our training dataset, we carefully separated the target information, which we called 'Label,' and stored it in a variable called y_train. At the same time, we identified the actual text content in the 'Text' column and saved it in another variable called x_train. This separation of data into targets and features set the stage for our analysis.

To give our text a numerical representation, we used a special tool called a tokenizer. The technique employed by the tokenizer is known as 'Word Tokenization'. This technique breaks down the input text into individual words and assigns a unique numerical index to each word within the text corpus. Word tokenization is a fundamental step in natural language processing (NLP) as it enables the conversion of textual data into numerical sequences. This tokenizer was taught about the words in the x_train text using tokenizer.fit_on_texts(x_train). This method learns the vocabulary from the training text data, so it could convert them into a sequence of numbers using tokenizer.texts_to_sequences(x_train) that converts each word in the text data into a corresponding integer based on the vocabulary learned during fitting. The result was a sequence of numbers that represent the words in the text. We stored this sequence in a variable called sequences_train.

To ensure all our sequences had the same length, we found out the longest sequence in sequences_train and used that length to add extra numbers to the shorter sequences. This process is called padding, and it helped us make sure all our sequences were the same size. The final padded sequences were stored in a variable called pad_x_train.

We repeated a similar process for our testing dataset. Using the same tokenizer, we converted the text in x_test into sequences, saved as sequences_test. Just like before, we padded these sequences to a consistent length, using the same length we found in the training dataset. The padded sequences for testing were saved as pad_x_test.

For the selected method for Vectorization, we did not use traditional vectorization methods like TF-IDF or binary encoding. Instead, we utilized word embeddings. Specifically, an embedding layer is used in the neural network model. This layer learns dense vector representations for words in the vocabulary during training, capturing semantic relationships between words.

In conclusion, these organized steps transformed our textual data into a format that machine learning models can understand. By converting text into numbers and making sure all sequences are of the same length, we've set the groundwork for effective analysis and machine learning training.

### 3.3.2. Feature Selection

Feature selection is a crucial data mining process that plays a significant role in reducing the complexity of data. It involves choosing the most relevant and meaningful attributes from a comprehensive set of attributes in the training data. By reducing the dimensionality of the input data, this process can enhance the model's performance and lead to a reduction in the time and resources needed for model training. There are several feature selection methods, some of them are explained below:

i) Filter Methods: These techniques assess each attribute independently through statistical tests or other criteria to determine their relevance to the target variable. Attributes are then ranked or assigned scores, and a predefined threshold is employed to select the most important attributes.
ii) Wrapper Methods: Wrapper methods encompass training and evaluating the model using diverse subsets of attributes. These methods often utilize performance metrics, such as accuracy or F1 score, to gauge the influence of each attribute subset on the model's effectiveness.

In this thesis, after the preprocessing operations, we employ three filter methods, namely Information Gain (IG), Chi-Square Test (Chi2), and Correlation Coefficient.

*Information Gain (IG)*

Information Gain (IG) serves as an entropy-based technique for evaluating features, and it holds substantial significance in the field of machine learning. IG serves to quantify the extent to which a feature contributes valuable information pertaining to a specific target group. By focusing on this target group, IG adeptly identifies features that harbor the most informative content. Notably, features characterized by elevated IG values exhibit a pronounced relevance to the target group, making them prime candidates for optimal classification outcomes. However, it's worth noting that IG does not possess the capability to eliminate features that offer redundant information (Win and Kham, 2019).

Features that lead to a significant reduction in uncertainty are considered more informative and relevant.

$$E(X) = -\sum_{i=1}^{k} P(x_i) \,.\, \log_2 P(x_i) \tag{3.1}$$

E(X) in the Equation 3.1, represents the entropy of a random variable *X*. It's calculated by summing over all possible outcomes (*i*) for the random variable *X* and multiplying the probability of

each outcome ($P(x_i)$) with the base-2 logarithm of the probability. This measures the uncertainty or disorder in the variable *X*. The lower the entropy, the less uncertain the variable is.

$$IG(X,Y) = E(X) - E(X|Y) \tag{3.2}$$

*IG(X, Y)* in the Equation 3.2, represents the information gain of variable *X* when you consider the presence of attribute *Y*. It's calculated as the difference between the entropy of *X (E(X))* and the conditional entropy of *X* given *Y (E(X|Y))*. Information gain quantifies how much knowing attribute *Y* reduces the uncertainty (entropy) in variable *X*. A higher information gain implies that attribute *Y* is more informative about variable *X*.

$$E(X|Y) = -\sum_j P(y_j) \sum_i P(x_i|y_j) \log_2 P(x_i|y_j) \tag{3.3}$$

In Equation 3.3, *E(X|Y)* represents the conditional entropy of variable *X* given attribute *Y*. It is calculated by summing over all possible outcomes for the target variable $Y_j$ and for the attribute $Y_i$, and multiplying the probability of *Y* ($P(Y_j)$) with the conditional probability of *X* given *Y* ($P(X_i| Y_j)$) times the base-2 logarithm of this conditional probability. This measures the entropy of variable *X* after considering the information provided by attribute *Y*. It quantifies the remaining uncertainty in *X* after observing *Y*.

Where:
*k:* Number of classes,
*E(X):* Entropy of *X*,
*E(X|Y):* is the conditional entropy of target *X* given attribute *Y*

*Chi-Square Test (Chi2)*

The Chi-Square (Chi2) method serves as a widely employed tool for exploring potential relationships among distinct data categories. It enables the comparison of observed data against expected patterns, a particularly valuable approach when dealing with datasets characterized by varying categories. This method, often implemented using the Chi2 formula, becomes especially effective when discrepancies from anticipated outcomes arise. This scenario can have notable implications for datasets where smaller values play a significant role (Liu and Setiono, 1995).

Furthermore, the Chi2 Test is a statistical technique harnessed to assess the link between two categorical variables. In the context of feature selection, Chi2 measures the degree of independence between a specific feature and the target variable. It does so by gauging the contrast between the projected and actual event frequencies recorded within a contingency table. Notably, features

endowed with a higher Chi2 score signify a stronger connection to the target variable, making them more relevant within the given context.

$$X^2 = \sum_{i=1}^{2} \sum_{j=1}^{k} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \quad (3.4)$$

Where:

*k:* Number of classes,

$A_{ij:}$ Number of patterns in the ith interval, jth class,

$C_j$*:* Number of patterns in the jth class,

$R_i$*:* Number of patterns in the ith interval,

*N:* is the total number of patterns,

$E_{ij}$*:* Expected value of $A_{ij}$.

In Equation 3.4, the chi-squared ($X^2$) statistic is used to determine the strength of association or independence between two categorical variables. It is calculated based on observed ($A_{ij}$) and expected ($E_{ij}$) frequencies in a contingency table. $X^2$: This is the chi-squared test statistic, a numerical measure that quantifies the association or independence between two categorical variables. $A_{ij}$: Represents the observed frequency of events falling into the *i*th interval and *j*th class in the contingency table. $E_{ij}$: Denotes the expected frequency of events in the *i*th interval and *j*th class, calculated based on marginal totals and the total number of observations (N). It's what we expect under the assumption of independence between the two variables. *i* (1 to 2): Signifies the two intervals or categories within one of the categorical variables. *j* (1 to k): Indicates the k categories in the other categorical variable. The formula calculates the sum of squared differences between the observed and expected frequencies in each cell, normalized by the expected frequency. The result ($X^2$) provides insight into the degree of discrepancy between what is observed and what would be expected if the two variables were independent. A higher $X^2$ suggests a stronger association between the variables, while a lower $X^2$ implies independence.

*Correlation Coefficient*

The Correlation Coefficient is a valuable tool for understanding the strength and direction of a linear connection between two continuous variables. It is widely employed in situations where there is a need to explore potential relationships within distinct categories of data. One of the primary applications of the Correlation Coefficient is in feature selection, where it assists in identifying features that exhibit a significant correlation with a specific target variable.

In essence, a positive correlation coefficient indicates that as one variable's values increase, the values of the target variable tend to increase as well, and vice versa. Conversely, a negative

correlation coefficient suggests that as one variable's values increase, the values of the target variable tend to decrease, and vice versa. Within the realm of feature selection, a high positive or negative correlation implies that variations in one variable can be associated with corresponding changes in the target variable (Guyon and Elisseeff, 2003).

The Pearson product-moment correlation coefficient ($P_{xy}$) is equal to the covariance of variables $x$ and $y$ ($Cov(x,y)$) divided by the product of the standard deviations of $x$ ($\sigma_x$) and $y$ ($\sigma_y$) as in equation 3.5:

$$P_{xy} = \frac{Cov(x,y)}{\sigma_x \sigma_y} \tag{3.5}$$

In this equation: $P_{xy}$ represents the Pearson product-moment correlation coefficient, which measures the linear relationship between variables $x$ and $y$. $Cov(x,y)$ stands for the covariance of variables $x$ and $y$, which quantifies how x and y vary together. $\sigma_x$ represents the standard deviation of variable $x$, which measures the spread or variability of data in $x$. $\sigma_y$ represents the standard deviation of variable $y$, which measures the spread or variability of data in $y$. The covariance of two variables, $X$ and $Y$, is a measure of how they vary together. It is calculated as shown in equation 3.6:

$$Cov(x,y) = \sum(X_i - \bar{X})(Y_i - \bar{Y}) / n \tag{3.6}$$

where:

$X_i$ and $Y_i$ are the individual values of variables $X$ and $Y$, respectively

$\bar{X}$ and $\bar{Y}$ are the means of variables $X$ and $Y$, respectively

$n$ is the number of observations

### 3.3.3. Text Representation in NLP

Text representation in Natural Language Processing (NLP) involves converting raw textual information into a numerical format that can be processed by computer. Text representation techniques play an important role in facilitating machine learning algorithms to effectively handle and understand human language.

The raw text corpus is preprocessed and converted into a format that can be an input of machine learning models, starting by preprocessing steps such as we described in details in the previous section, then this data is converted to the required format to be input of the designed model.

There are several common approaches to text representation in NLP such as: Word Embeddings, Transformers and Pre-trained Models, and Neural Network Architectures.

The text representation approach adopted in this thesis aligns with the principles of neural networks architecture, specifically focusing on learning representations by processing words in a

textual sequence. Our methodology employs the embedding layer from Keras, that is used to create word embeddings, which are dense vector representations of words that capture semantic relationships and are essential for effectively processing and understanding textual data in neural network architectures. It is important to note that the embedding layer in Keras does not rely on a specific formula or method like traditional word embedding techniques such as Word2Vec or GloVe. Instead, it learns the word embeddings directly from the data while training the neural network. Initially, it sets embeddings randomly, while training it adjusts these word embeddings to minimize the loss function, The word embeddings change through backpropagation, over time, these embeddings start to represent the meanings of words based on how they are used in data (Brownlee, 2021).

It transforms integer-encoded words into dense vectors that the neural network can process. The embedding layer maps each word to a point in the embedding space, where similar words are closer together, allowing the network to capture semantic and contextual information.

When engaging the Keras embedding layer, the user generally specifies two key parameters: the size of the vocabulary (the total count of unique words within the dataset) and the dimensionality of the embedding vectors (the length of each vector). During training, the layer refines these embeddings, and it can also integrate pre-existing embeddings if provided.

This embedding layer commonly serves as the foundational element within neural network architectures tailored for NLP tasks. As the network matures, the acquired embeddings are fine-tuned in harmony with the overall network. The embedding values are randomly initialized and then fine-tuned during training of the model. As the network learns, the embedding vectors are adjusted to capture meaningful relationships between words based on the task. thereby enhancing performance specific to the task at hand (Saxena, 2020).

### 3.3.4. Convolutional Neural Network (CNN)

The Convolutional Neural Networks (CNNs) are the most commonly used architecture in deep learning which was proposed by LeCun et al. (1989) in their work on handwritten digit recognition.

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that most commonly applied to recognize data such as images and videos, which have digital representation, by processing their pixel data. CNNs are designed to detect the patterns in the processed data, that give the computer the ability to see. By the same way with different design, they can be used to detect the patterns in the textual data by extracting simple and complex patterns from texts and finding the relations between existing words. It achieved remarkable performance in various fields.

The architecture of a convolutional neural network is quite straightforward and is made up of three layers: an input layer, a hidden layer, and an output layer. However, the hidden layer is where

things get interesting. Unlike regular neural network setups, this layer includes a series of consecutive convolution and pooling layers. There are three main layers in CNN architecture:

*Convolutional layer*

A fundamental component of CNNs, which executes convolution operations on the input data by convolving a set of learnable filters (kernels) with the input data, extracts relevant features from the input data (LeCun et al., 1998).

*Pooling layer*

In the pooling layer, a process known as subsampling is applied to the feature vector derived from the previous convolutional layer. This process involves different techniques such as max-pooling, min-pooling, and average-pooling. The primary purpose of the pooling layer serves two functions. First, it ensures a consistent-sized output, accommodating inputs of varying lengths. Second, it diminishes the feature dimensions while preserving crucial information. Moreover, the pooling layer decreases the spatial dimensions of the Convolution layer's output by consolidating the values of neighboring neurons. This action effectively lowers the network's computational complexity and concentrates on the most important features.

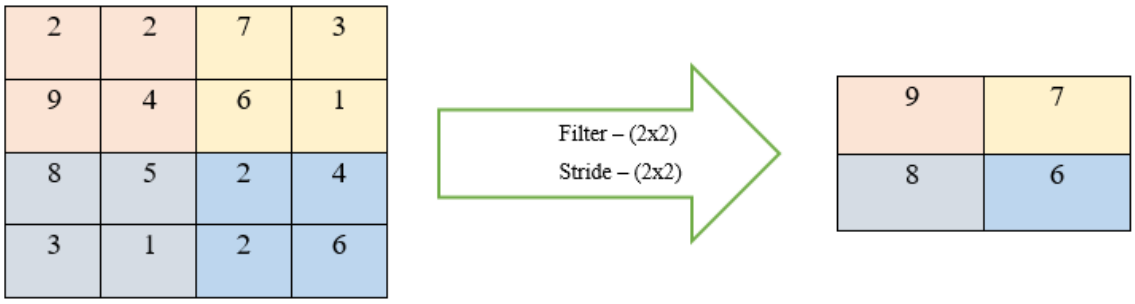


Figure 3.9. Max Pooling Method

The above Figure 3.9 represents how max-pooling technique is done: in this example the input is a 4x4 grid of numbers. The pooling layer has a filter size of 2x2 and a stride of 2. This means that the filter will slide over the input grid, taking the maximum value from each 2x2 block. The output of the pooling layer is a 2x2 grid of numbers. The purpose of max pooling is to reduce the size of the input while preserving the most important features. This helps to make the CNN more efficient and to prevent overfitting. In this example, the max pooling layer has reduced the size of the input from 4x4 to 2x2. This has reduced the number of parameters in the CNN, which makes it more efficient. The max pooling layer has also preserved the most important features in the input, such as the large values in the top-left and bottom-right corners.

*Fully connected layer*

Fully connected layer allows the network to learn complex patterns from the input by connecting all neurons in the previous and the next layers to get flatten matrix that can classify the processed data.

Here there are some of CNN applications in our world today:

- Object detection
- Image captioning
- Video analysis
- Natural language processing
- Image recognition

*CNN in Text Classification*

CNNs can be used in the text classification process when we want to make the input layer takes sentences or words to be classified in the output layer. To make the CNNs able to process textual input data we need to edit its architecture, so we can add Embedding layer, that represents each word in the input by vector (word embedding), we can also add several convolutional and pooling layers, using different hyperparameters and activation functions.

*When to Use CNN in Texts*

Since CNN has the ability to detect patterns from input data, so it is obviously that it will be better in operations that need to extract some specific features such as detecting spam email and sentiment analysis operations, since the previous one has similar features generally (Ghelani, 2019).

*How to Use CNN in Texts*

- Collect and Preprocess Text Data: Gather the text data you'll be working with and perform necessary preprocessing, such as removing special characters and converting text to lowercase.
- Tokenize Texts: Split the preprocessed text into individual words or tokens. This process is called tokenization and forms the basis for further processing.
- Convert to Integer Vectors and Apply Padding: Convert the tokenized text into integer values and ensure that all text sequences have the same length by adding padding. This uniformity is essential for processing in a CNN.
- Split Data into Training and Test Sets: Divide your dataset into two groups: training data, which the CNN will learn from, and test data, which will be used to evaluate the model's accuracy.

- Design CNN Architecture: Create the architecture of your Convolutional Neural Network. Decide on the number of layers, filter sizes, and pooling operations.
- Compile the Classifier: Specify the loss function, optimizer, and evaluation metrics for your CNN model. This configuration defines how the model learns and updates its parameters.
- Train the Model: Start training your CNN model using the fit method. Set parameters such as batch size (number of samples processed before updating the model) and the number of epochs (iterations over the entire training dataset).
- Evaluate Model Performance: Once training is complete, assess the model's performance on the test dataset using the evaluate method. This provides metrics like accuracy, which indicates how well the model generalizes to unseen data.
- Make Predictions: Use the trained CNN model to predict labels for new, unseen data using the predict method. This step applies the model's learned patterns to make informed predictions.

### 3.3.5. Recurrent Neural Network (RNN)

A recurrent neural network (RNN), presented by Elman (1990), introduced the concept of using recurrent connections within neural networks to capture sequential dependencies and patterns in data. This network is a type of artificial neural network that works with data that comes in sequences or over time such as translating languages, processing human language, recognizing speech, and adding captions to images. RNNs are different from other neural networks because they remember previous information and use it to affect what they do next. While normal neural networks assume that things happening earlier and later are separate, RNNs consider what happened before to help decide what happens next (IBM, n.d.).

This artificial neural network architecture is designed to work like the human brain. It's built to remember what happened before and use that memory to guess what might happen next. A feed-forward neural network follows a single path for information flow: it starts from the input layer, moves through the hidden layers, and reaches the output layer, without revisiting any node along the way.

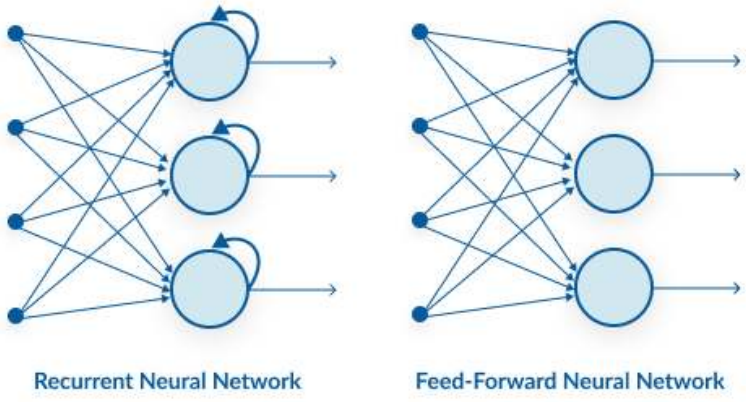


Figure 3.10. Recurrent Neural Network vs Feedforward Neural Network (Source: Uditvani.com)

The Figure 3.10 shows two types of neural networks: a feed-forward neural network (FNN) and a recurrent neural network (RNN).

- Feed-forward neural network is a type of neural network that processes data in a linear fashion. The input data is passed through a series of layers, and the output is generated at the final layer. FNNs are commonly used for tasks such as classification and regression.
- Recurrent neural network is a type of neural network that allows information to flow back and forth between layers. This makes RNNs well-suited for tasks that involve sequential data, such as natural language processing and speech recognition.

The Figure 3.10 shows the basic structure of each type of neural network. The FNN has three layers: an input layer, a hidden layer, and an output layer. The RNN has an input layer, an output layer, and a recurrent layer. The recurrent layer allows information to flow back and forth between the input and output layers. In other words, Feed-forward neural networks are not good at making predictions about future events because they can't remember the information they receive. They only consider the current input and don't understand the order of events over time. Once they are trained, they forget about the past. In contrast, Recurrent Neural Networks (RNNs) use a loop to cycle through information. They consider both the current input and what they've learned from past inputs before making a decision. RNNs have internal memory, so they can recall previous information. They generate output, store it, and then use it again in the network.

While Recurrent Neural Networks (RNNs) are good at using what they have learned in the past, RNNs can run into two problems: too-big changes and too-small changes. These are called exploding gradients and vanishing gradients. They can only go back a short way. When they try to correct their mistakes, they can only fix a small part of the problem. This problem is called the "vanishing gradient problem". If changes are too big, the program becomes unstable, and things might break. This problem is called the "exploding gradients". This type of RNN cannot solve these problems on its own. To make up for this, other RNN types have been created, like bidirectional

RNN, deep bidirectional RNN, and LSTM. These new types help fix the limitations of the original RNN (Kalita, 2022).

*When to Use RNN*

Recurrent Neural Networks are chosen for tasks that involve sequences of data and situations, where remembering past information matters. They are particularly useful when there is a connection between current and previous data points over time. RNNs are suitable for tasks that need to understand the history of the data, manage different sequence lengths, create new content, or use output as input for the next step (Ghelani, 2019).

*RNN Applications*

- Recognizing spoken language
- Creating music
- Automating translations
- Analyzing actions in videos
- Investigating genetic sequences and DNA patterns

### 3.3.6. Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks stand out as a unique kind of recurrent neural network architecture. They effectively tackle the challenge of vanishing gradient problems, ensuring that information doesn't get lost over time, and they're also adept at capturing important connections across long sequences of data. These networks were introduced by Hochreiter and Schmidhuber, (1997) and have become a crucial foundation in various fields that involve dealing with sequences of data, such as natural language processing, speech recognition, and predicting trends over time. Although it's more intricate than a standard RNN, its structure is still based on repeating blocks, similar to RNNs. However, LSTM introduces a significant enhancement. While traditional RNNs have just a hidden state, LSTM features two distinct states: a hidden state and a cell state. Within the LSTM design, there are three gates responsible for managing the flow of information to and from the cell state. These gates, along with the cell state itself, effectively address the problem of information vanishing over time. In an LSTM structure, each block comprises these two states and four layers, contributing to its unique capabilities and versatility.

*LSTM Architecture*

The LSTM network architecture consists of three parts, as shown in the Figure 3.11 below, and each part performs an individual function.

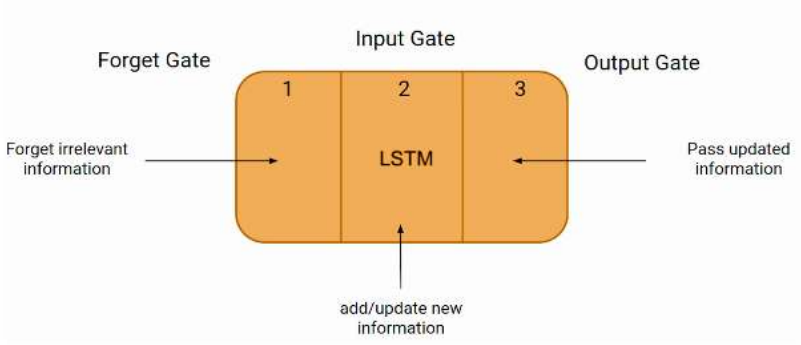


Figure 3.11. LSTM Architecture (Saxena, 2021)

In an LSTM, there are three important parts (Saxena, 2021):

- **Forget Gate:** This decides what information to throw away from the memory cell. It learns to keep what is important and forget what is unnecessary from previous steps.
- **Input Gate:** by quantifying the importance of the new information carried by the input, it chooses that if it needs to store it in the memory cell. It adds new data to the cell's memory based on what is happening now.
- **Output Gate:** This controls how information flows out of the memory cell. It helps to keep and retrieve information from the memory cell when making predictions.

  The Figure 3.12 shows a block diagram of a long short-term memory (LSTM) module showing the General scheme of LSTM.

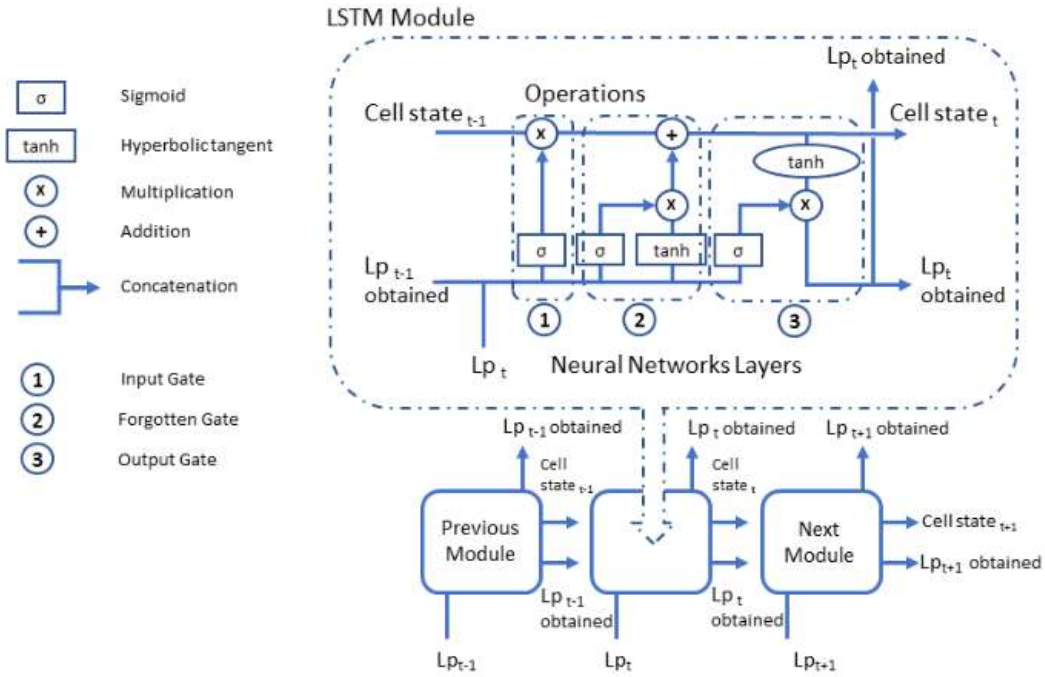


Figure 3.12. General scheme of LSTM

*LSTM Applications*

Long Short-Term Memory stands out as a highly effective type of Recurrent Neural Network, having found valuable roles in various fields (Banoula, 2021), such as:

1. Understanding Language: LSTM models are harnessed for tasks involving natural language, like translating languages, forming language models, and creating summarized text. By grasping the connections between words in a sentence, these models are trained to craft coherent and correct sentences.
2. Recognizing Voices: LSTMs are employed for speech-related tasks such as turning spoken words into written text and identifying spoken commands. They're trained to detect patterns in speech and match them with the appropriate written words.
3. Evaluating Emotions: LSTMs can classify the sentiment of text as positive, negative, or neutral by learning the connections between words and their associated emotions.
4. Predicting Time-Related Data: LSTMs come in handy for foreseeing upcoming values in sequences over time. This is done by learning the links between past and future values.
5. Analyzing Videos: LSTMs find utility in video analysis, learning the relationships between frames and the actions, objects, and scenes they depict.
6. Deciphering Handwriting: LSTMs can recognize handwriting by understanding the connections between images of written text and the corresponding textual content.

### 3.3.7. Language Models

As mentioned by Brown et al. (2020), recent research has shown significant advancements in different areas of Natural Language Processing (NLP) through a specific methodology. This approach comprises two primary stages: initially, the model undergoes pre-training on a substantial text collection, and subsequently, it is refined for a specific task. However, these models still require specialized datasets for the fine-tuning phase, which should contain numerous examples. This differs from human capabilities, where we can often understand new language-related tasks with only a limited number of examples or basic instructions, whereas present NLP systems find it challenging to achieve a similar level of adaptability.

A language model is a type of machine learning model created to understand and work with language. It forms the foundation for various tasks involving language, such as answering questions, conducting semantic searches, creating summaries, and many more activities related to human language (Çelik, 2022). The core purpose of these models is to predict the upcoming word or sequence of words in a sentence based on the words that precede them.

### 3.3.8. Bert Language Model

BERT, an acronym for "Bidirectional Encoder Representations from Transformers," is a sophisticated natural language processing (NLP) model created by Devlin et al. (218). The fundamental architecture it employs, known as the Transformer, utilizes self-attention mechanisms to efficiently handle input data concurrently. This trait makes it particularly well-suited for processing sequences of data, such as textual content. Notably, BERT possesses bidirectional capabilities, which means it comprehends the context of words within a sentence from both the left and right sides (Lutkevich, 2020). Regarding its training process, BERT follows a two-step approach:

i) Pre-training: Initially, the model undergoes unsupervised pre-training. It is exposed to extensive volumes of text data, for example, excerpts from Wikipedia. During this phase, BERT learns to predict missing words within sentences, a technique known as masked language modeling. Additionally, it gains an understanding of how words interrelate and form meaningful sentences.
ii) Fine-tuning: Following the pre-training phase, BERT can be fine-tuned for specific tasks known as downstream tasks. These tasks encompass various NLP applications such as text classification, named entity recognition, or question-answering. Fine-tuning makes BERT highly adaptable and capable of excelling in diverse language-related tasks.

### 3.3.9. A Robustly Optimized BERT Pretraining Approach (RoBERTa)

RoBERTa is an advanced language model that has been created to enhance the training of models that understand human language. It's an improvement upon a model called BERT. This new model, RoBERTa, is designed to perform better on various tasks involving language comprehension. It is a version of BERT that can handle different language-related challenges more effectively. The name "RoBERTa" stands for "Robustly Optimized BERT approach".

The study of (Liu et al., 2019) centers on this language model, that is taken from improvement in language model pretraining. Comparing different approaches presents challenges due to factors like computing expenses, confidential datasets, and influential hyperparameter selections. The study duplicates BERT pretraining and evaluates hyperparameters and the size of the training dataset. This scrutiny uncovers that BERT was insufficiently trained and introduces an upgraded training methodology termed RoBERTa. RoBERTa can rival or surpass later BERT models. Notable refinements encompass extended training duration, larger data groups, omission of next sentence prediction, lengthier sequences, and adaptable masking. In summary, the "FULL-SENTENCES" implementation of the reimplementation approach achieves notably strong results across multiple tasks, outperforming or closely rivaling baseline models like "BERTBASE" and "XLNetBASE."

According to Phung (2021), RoBERTa is a significant advancement built upon BERT, making some key adjustments to optimize its performance. These changes include longer training times, a substantial increase in training data, larger batch sizes, the removal of the Next Sentence Prediction (NSP) task, a larger vocabulary, and the use of longer input sequences while retaining a token limitation of 512. Despite these simple-sounding changes, RoBERTa significantly improves model performance across various tasks compared to BERT. It shares a similar timeframe of publication with XLNet, which also enhances performance but employs more complex modifications that are challenging to implement. RoBERTa adopts the same architecture as BERT and is pretrained using the Masked Language Modeling (MLM) task while omitting the Next Sentence Prediction (NSP) task.

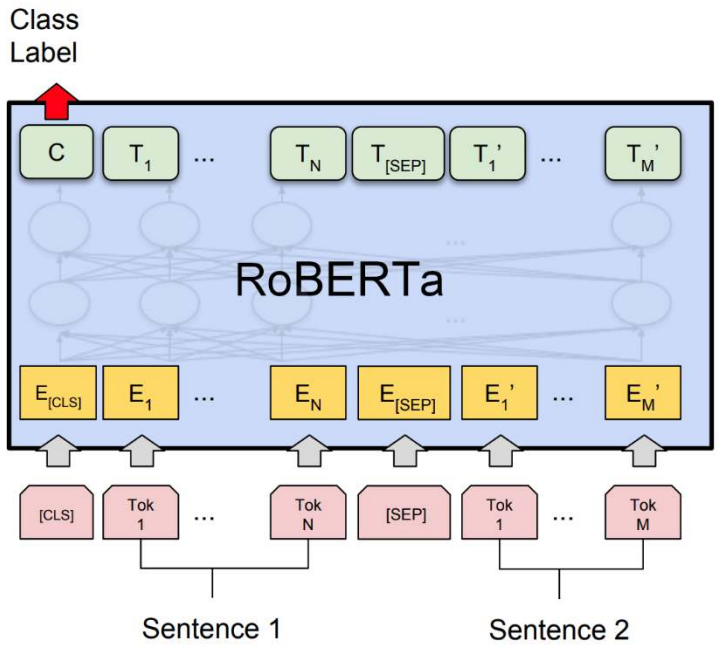


Figure 3.13. Structure of RoBERTa (Source: https://www.labellerr.com/blog/roberta-a-robustly-optimized-bert-pretraining-approach/)

The Figure 3.13 represents a class label for RoBERTa, the class label in the figure is used to identify the text that has been classified as RoBERTa. The label is composed of the following parts:

- CLS: This is a special token that is used to identify the beginning of the text that should be classified.
- Class: This is the name of the class.
- Tokens: These are the individual words that make up the text.
- SEP: This is a special token that is used to separate the tokens.
- Mask: This is a special token that is used to mask out the tokens that are not used for classification.

The class label is used by the RoBERTa model to identify the text that it should be trained on. The model learns to classify text by comparing the encoded representation of the text to the encoded representations of the training text.

### 3.3.10. RoBERTa-CNN Ensemble Classifier

In this thesis, we combine the optimized architectures of CNN and RoBERTa to create a new ensemble model. This ensemble model brings together the strengths of both architectures.

First, we train the ensemble model using RoBERTa's contextual embeddings as its input. Then, we apply convolutional operations through a CNN classifier. This approach allows our ensemble model to benefit from RoBERTa's language understanding abilities while also capturing specific patterns and features through the convolutional layers. To simplify further, the architecture does the following:

1. Loads and preprocesses text data for training and testing.
2. Utilizes the RoBERTa language model to extract contextual embeddings from the text.
3. Applies a CNN layer on top of RoBERTa's output to capture local patterns and features.
4. Adds a classification layer for making predictions.
5. Trains the model using the loaded data, optimizing for accuracy.
6. Evaluates the model's performance on the test data.

The figure 3.14 below shows the designed ensemble architecture:

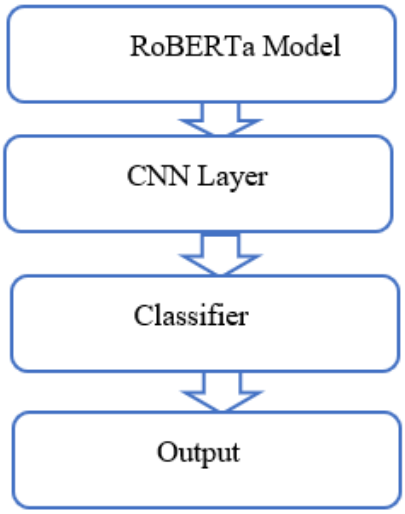


Figure 3.14. Used Architecture of RoBERTa-CNN

### 3.3.11. Performance Evaluation Metrics

Once the process of selecting relevant features and creating the model is complete, the effectiveness of classification models is assessed using metrics such as precision, recall, accuracy, and F-score.

In the field of machine learning, there is a valuable tool known as a confusion matrix. Confusion Matrix is used for performance measurement for machine learning classification problem where output can be two or more classes. It often is used in supervised learning, functions within the context of statistical classification. It is a table with four different combinations of predicted and actual values as shown in Figure 3.15. Each row in the matrix corresponds to instances from a true class, while each column represents predictions made for a specific class, or vice. Its primary purpose is to swiftly detect if the system is mixing up two types of data. It has two dimensions labeled "actual" and "predicted," with both dimensions containing the same sets of "classes." Each pairing of dimension and class is a distinct variable in this table. It represents critical metrics like True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) (Narkhede, 2018).

**True Positive:** classified as positive and it's true.
**True Negative:** classified as negative and it's true.
**False Positive:** classified as positive and it's false.
**False Negative:** classified as negative and it's false.

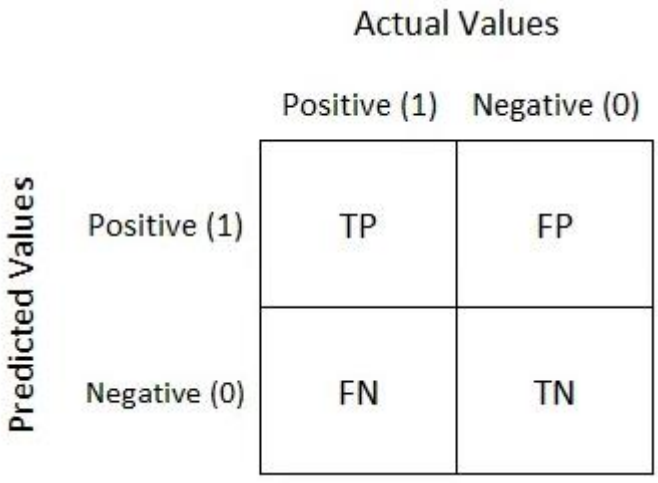


Figure 3.15. Confusion Matrix (Narkhede, 2018)

*Recall*

Recall, also known as Sensitivity or True Positive Rate, is a metric used to measure the ability of a classification model to correctly identify all positive instances out of all the instances that are actually positive. A higher recall indicates that the model is good at finding positive cases even if it sometimes identifies positives as negatives.

$$Recall = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} \tag{3.6}$$

*Precision*

Precision is a metric that gauges the accuracy of positive predictions made by a classification model. It calculates the ratio of correctly predicted positive instances to all instances that the model labeled as positive. A higher precision means the model is cautious in labeling positive cases and tends to avoid false positives.

$$\text{Precision} = \frac{True\ Positives}{(True\ Positives + False\ \text{Positives})} \tag{3.7}$$

*Accuracy*

Accuracy is a measure of overall correctness in a classification model. It calculates the proportion of correctly predicted instances to the total number of instances. Accuracy provides an overall view of how well the model performs across all classes.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}} \tag{3.8}$$

In this thesis, we use this metric to measure the performance of our used classifiers. We mainly use accuracy as a way to assess how well our classifiers are doing. We chose accuracy because it is a simple and easy-to-understand measure of how often our model gets things right when it is classifying data. It is like looking at the percentage of correct answers out of all the questions.

Actually, accuracy is not always the best choice, especially when we have situations where there are very few examples of one class. In this cases, other metrics like the F1-score might be more helpful because they consider different aspects of classification performance.

*F-measure*

F-measure, also referred to as the F1-score, is a metric that considers both precision and recall to provide a balanced evaluation of a classification model. It's especially useful when dealing with imbalanced datasets. A higher F-measure indicates a good balance between precision and recall.

$$\text{F1} - \text{Score} = \frac{2 * (\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})} \tag{3.9}$$

As we used almost balanced datasets in this thesis, we used accuracy measure.

### 3.3.12. Research Approach and Experimental Workflow

This section outlines our research approach and the experimental workflow we follow to conduct our study on offensive language classification. We detail the various stages of our investigation, including data preparation, model selection, feature selection exploration, and the deployment of the RoBERTa-based classifier.

*Classification*

To build effective offensive language classifiers, we adopt a two-pronged approach involving deep neural network models and the advanced transformer-based model, RoBERTa. The NN models encompass Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM) networks. Our goal is to assess their performance on various datasets and languages and compare them. Subsequently, we use the RoBERTa model, to compare with NN-Based methods.

*Deep Neural Network Models*

Our initial exploration involves training and evaluating neural network models, including CNN, RNN, and LSTM, on the offensive language datasets. These models undergo training and validation processes, and their performance metrics are documented for comparison with the RoBERTa-based models. Then we apply the process of feature selection to refine our model's performance. Our investigation involves testing three distinct feature selection methods: 1) Information Gain, 2) Chi-Squared, and 3) Correlation Coefficient.

*RoBERTa-based Offensive Language Classification*

After working extensively with neural network classifiers, we introduce the RoBERTa language model to our study. Our workflow for RoBERTa-based classification involves dataset preparation, model training, and evaluation.

*RoBERTa-Based Offensive Language Classification without Preprocessing*

In an effort to explore the impact of preprocessing on the performance of RoBERTa-based offensive language classification, we conduct experiments without applying preprocessing operations to the datasets. The objective is to assess the intrinsic capabilities of the RoBERTa model in handling raw text without any additional linguistic refinement.

In this thesis, we create a substantial Arabic dataset covering a wide range of language use on social media platforms. Furthermore, we develop a multilingual dataset that pays special attention to ensuring a fair representation of different languages. Our research focuses on simplified neural network designs. We employe methods to pick the most important features for optimizing the

efficiency of our models, highlighting the potential for creating resource-efficient systems for detecting offensive language. Our study delves into the effectiveness and adaptability of RoBERTa and XLM-RoBERTa models, considering batch size and preprocessing. We also introduce a new ensemble classifier RoBERTa-CNN, which combines RoBERTa's language modeling abilities with CNN's feature extraction.

The selected hyperparameters were initially based on findings from various studies. However, after conducting experiments, some of these hyperparameters were adjusted based on the results we obtained. For our Convolutional Neural Network (CNN), we tested hyperparameters that had been previously discussed in studies by Gambäck and Sikdar (2017), Kim (2014), and Janakiev (n.d.). In the case of RNN, we began with a simple architecture, including an embedding layer whose parameters were informed by our CNN experiments, one RNN layer, and one dense layer for classification. Subsequently, we enhanced this architecture by drawing inspiration from ideas and methodologies presented in studies by DataTechNotes (2019) and Kcsener (2019). However, for LSTM, we utilized the RNN architectures we had developed earlier, converting each RNN layer into an LSTM layer and incorporating insights from (Pitsilis et al., 2018).

## 4. RESULTS AND DISCUSSIONS

During the classification phase, we employ CNN, RNN, LSTM, and RoBERTa classifiers on our datasets. To determine the optimal configurations for these classifiers, a series of diverse experiments is conducted. These experiments leverage validation accuracy and validation loss metrics to assess the performance of the classifiers and the relative efficacy of the classification techniques in relation to each other.

The central focus of this thesis revolves around a binary classification approach, aimed at identifying offensive language within textual content. This involves assessing the classifiers' performance by computing their validation accuracy.

The hardware configuration of the system used to conduct the experiments and analyses presented in this thesis for the neural network classifiers is as follow:

*Device 1:*

- Processor: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
- Installed RAM: 16.0 GB
- System Type: 64-bit operating system, x64-based processor
- Does not have GPU

When we start to use RoBERTa language model we face some problems using the previous described hardware such as out of memory problem and the very long training time, because of the big hunger to resources that transformer-based language models have, so we run RoBERTa experiments on another device that have the described details below:

*Device 2:*

- Processor: 12th Gen Intel(R) Core(TM) i7-12700H  2.30 GHz
- Installed RAM: 16.0 GB (15.7 GB usable)
- System type: 64-bit operating system, x64-based processor
- Display device NVIDIA GeForce RTX 3050 Ti Laptop GPU
- Has GPU

### 4.1. CNN-based Offensive Language Classification

As in (Kim, 2014) that had made different experiments on simple CNN architecture and one of them was CNN-rand that is a model where all words are randomly initialized and then modified during training, my first CNN architecture is  a form that is one of the simplest architectures of CNN that has one Embedding layer, one conv layer, one globalmaxPooling layer and two dense layers just like the one that was prepared by (Janakiev, n.d.).

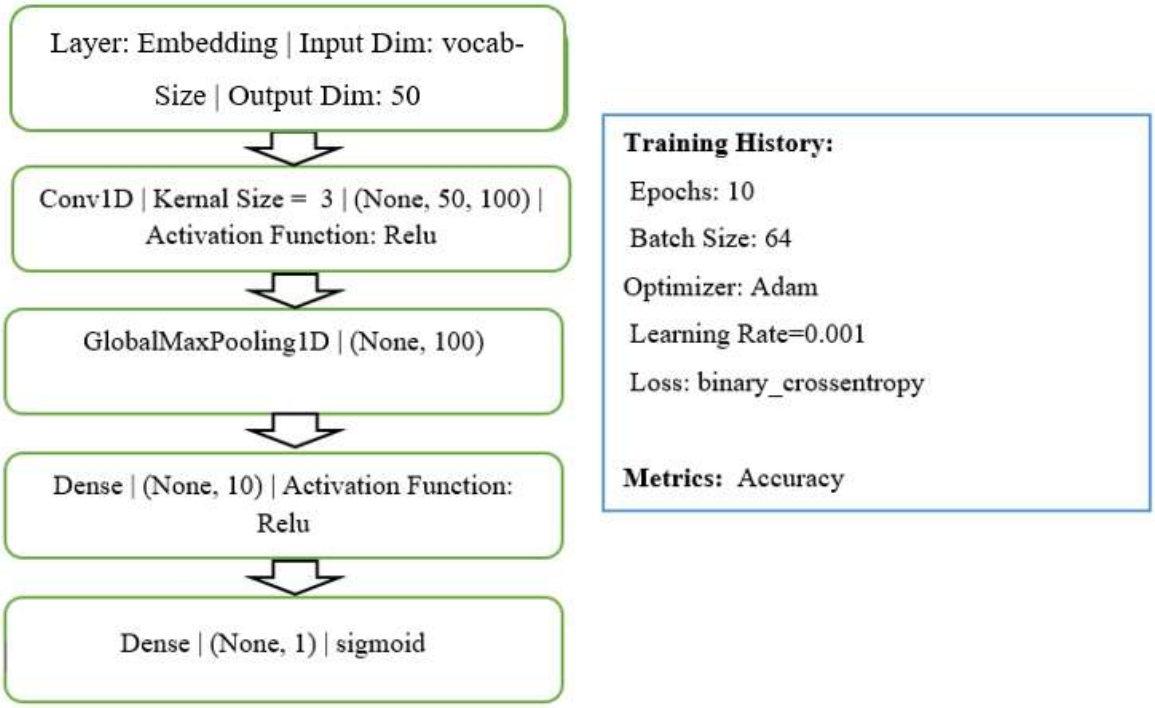


Figure 4.1. 1.Architecture of CNN

Hyperparameters are set as the Figure 4.1 above :

The batch size is set to 64 that is a common choice because it strikes a balance between performance and efficiency. With a batch size of 64, the model can still learn effectively without taking too long to train, kernel size of conv is set 3 (Gambäck and Sikdar, 2017; Kim, 2014), with 10 epoch size as in (Janakiev, n.d.). with adam optimizer and 0.001 learning rate (Gambäck and Sikdar, 2017).

This section delves into the description of the constructed CNN model, which is tested using English-language dataset in pursuit of achieving optimal accuracy scores. The tuning of hyperparameters was a pivotal part of refining the model's architecture.

We start with the English dataset due to its inherent formality, unlike the Arabic language which encompasses diverse dialects and distinct forms of Arabic letters contingent on their placement within words.

In the context of the chosen dataset, I conducted experiments employing diverse convolutional feature map sizes, namely 32, 64, 100, and 128 where they all were mentioned in (Gambäck and Sikdar, 2017; Kim, 2014; Janakiev, n.d.), each accompanied by 3 as a kernel size such as in (Gambäck and Sikdar, 2017; Kim, 2014), within the Conv1D layer. The results of these investigations are presented in Table 4.1, outlining the performance of the CNN model across different feature map sizes:

Table 4.1. Experimental Results of CNN Using Different Feature Map Sizes

| Feature Maps | val_accuracy | val_loss |
|---|---|---|
| 32 | 0.8782 | 0.4141 |
| 64 | 0.8820 | 0.4237 |
| 100 | **0.8859** | 0.4035 |
| 128 | 0.8736 | 0.4507 |

This systematic exploration enabled us to pinpoint that 100 feature map sizes led to the highest accuracy score, underscoring its suitability for the subsequent stages of our analysis. Based on the findings, I determined the optimal configuration for the CNN model as utilizing 3 filters and 100 feature maps.

Within the architecture of my model, the embedding layer employed is the Keras Embedding layer. This component operates by transforming individual words within the input sentence into fixed-length vectors, the size of which is designated by the embedding_dim parameter. The vocab_size parameter dictates the count of unique words within the vocabulary, while the input_length parameter defines the maximum sequence length of the input.

When no specific word embedding method is indicated, the Embedding layer initializes word embeddings randomly. By default, this layer employs the 'uniform' initialization technique to establish the embedding matrix. These initial embeddings are then fine-tuned during training, adapting to the specific characteristics of the training dataset.

Further experiments were conducted by varying the dimensional vector (embedding_dim value), yielding the subsequent results:

Table 4.2. Results of Different Dimensional Vectors

| Dimensional Vector | Val_accuracy | val_loss |
|---|---|---|
| 50 | **0.8867** | 0.4208 |
| 100 | 0.8843 | 0.4212 |
| 200 | 0.8836 | 0.4032 |
| 300 | 0.8859 | 0. 4035 |

According the Table 4.2 above we can say that we get the highest validation accuracy by setting embedding_dim value as 50. I conducted additional experiments to explore various numbers of units in the first dense layer. The highest accuracy was achieved when the number of units was set at 10. To enhance accuracy and prevent overfitting, I integrated two dropout layers into the architecture inspired by Kim's work (Kim, 2014). This resulted in my second CNN architecture comprising 3 Conv1D layers, 2 MaxPooling1D layers, 1 GlobalMaxPooling1D layer, 2 Dropout layers, and 2 Dense layers. The word vector dimension was set to 50, diverging from Kim's architecture which employed a word vector dimension of 300 due to the high accuracy observed with the former configuration. The second architecture of CNN in this thesis is presented in the Figure 4.2 below:

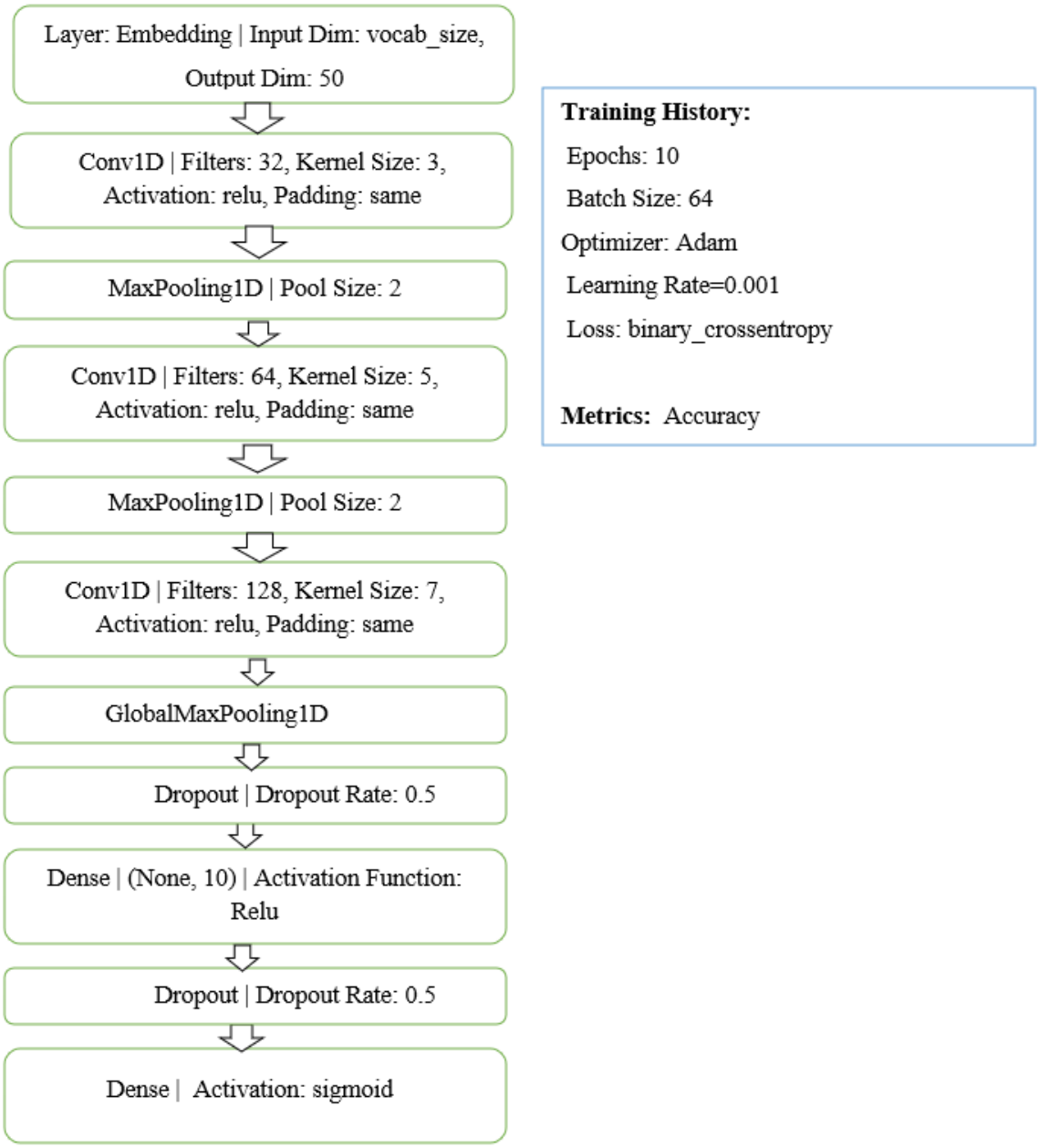


Figure 4.2. 2.Architecture of CNN

Describing the model architecture outlined in the study by Gambäck and Sikdar (2017), it encompasses the following components:

- Embedding Layer: This layer converts words from a vocabulary size of 10,000 into dense vectors of size 300.
- Conv1D Layer (with filters=128, kernel_size=5): Applying 128 filters with a kernel size of 5 to the input sequence, this convolutional layer employs the ReLU activation function.
- GlobalMaxPooling1D Layer: Operating as a global max pooling layer, it identifies the maximum value across the entire sequence, effectively reducing dimensionality.

- Flatten Layer: This layer transforms the preceding output into a 1D vector.
- 2 Dense Layers.

The model is then compiled with the Adam optimizer, employing the binary cross-entropy loss function and accuracy metric. Training takes place over 10 epochs using the provided training data (pad_x_train and y_train), with validation occurring using the validation data (pad_x_test and y_test). The achieved results are summarized in the Table 4.3 below:

Table 4.3. Model Performance Comparison on English Dataset

| CNN Models | Val_accuracy | val_loss |
|---|---|---|
| (Gambäck and Sikdar, 2017) | 0.8836 | 0.4719 |
| (Kim, 2014) | 0.8736 | 0.6360 |
| 1.Architecture | 0.8828 | 0.4211 |
| 2.Architecture | **0.8843** | 0.6649 |

Considering the obtained accuracy results, it's evident that the second architecture - featuring 3 Conv1D layers, 2 MaxPooling1D layers, 1 GlobalMaxPooling1D layer, 2 Dropout layers, and 2 Dense layers - stands out as the most proficient model for the English dataset. Subsequently, we extended the analysis to include training all architectures on both Arabic and Turkish datasets. Across all model designs, the following standardized settings were applied: 'adam' optimizer, 'binary_crossentropy' loss function, 10 epochs, and a batch size of 64, beside the same hyperparameters that are set in the first and the second architectures as are presented in the Figures 4.1 and 4.2. These settings were favored due to consistently high accuracy values they yielded. The results on the Arabic dataset are summarized in the Table 4.4 below:

Table 4.4. Model Performance Comparison on Arabic Dataset

| CNN Models | Val_accuracy | val_loss |
|---|---|---|
| (Gambäck and Sikdar, 2017) | 0.8237 | 0.8147 |
| (Kim, 2014) | 0.8241 | 0.9555 |
| 1.Architecture | **0.8416** | 0.4178 |
| 2.Architecture | 0.8273 | 0.5299 |

And the Table 4.5 below, provides a comparison of accuracy scores for the CNN architectures on the Turkish dataset.

Table 4.5. Model Performance Comparison on Turkish Dataset

| CNN Models | val_accuracy | val_loss |
|---|---|---|
| (Gambäck and Sikdar, 2017) | 0.8336 | 1.2148 |
| (Kim, 2014) | 0.8268 | 1.0987 |
| 1.Architecture | **0.8394** | **0.5886** |
| 2.Architecture | 0.8227 | 1.4259 |

For the duration of training sessions, the following timings were observed: training models on the English dataset took less than 5 minutes, while on the Turkish dataset, it ranged between 20 to 40 minutes. However, training on the Arabic dataset extended beyond 3 hours. These durations reflect the time taken by each training iteration for every architecture. If we compare the sizes of the three datasets, where there are more than 6,000 record in the English, more than 30,000 rows in the Turkish, and about 215,000 rows in the Arabic dataset, so larger datasets typically require more time to train because there are more data points to process during each training iteration. Initially, all models were trained using the default learning rate of 0.001. However, upon observing diminishing validation accuracies during training, adjustments were made. Specifically, the simplest architecture model was subjected to extended training on the English dataset, spanning over a hundred epochs. The results of this prolonged training period are depicted in the curves presented in Figure 4.3 and Figure 4.4:

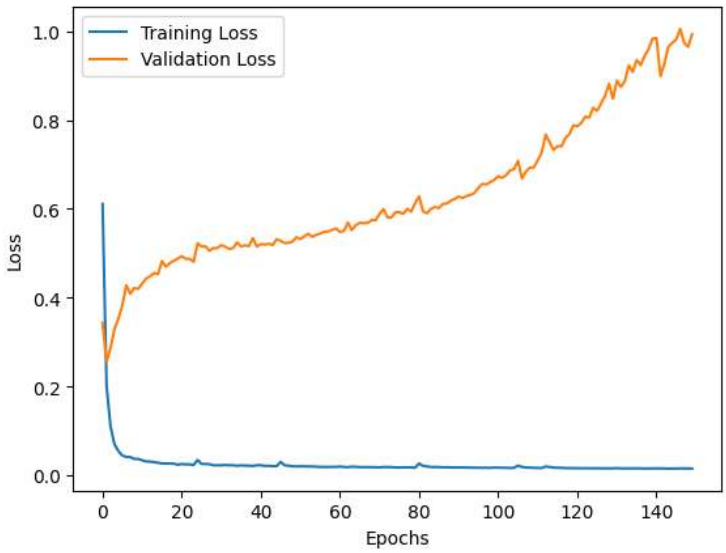


Figure 4.3. Validation Loss Over 150 Epochs with 0.001 Learning Rate

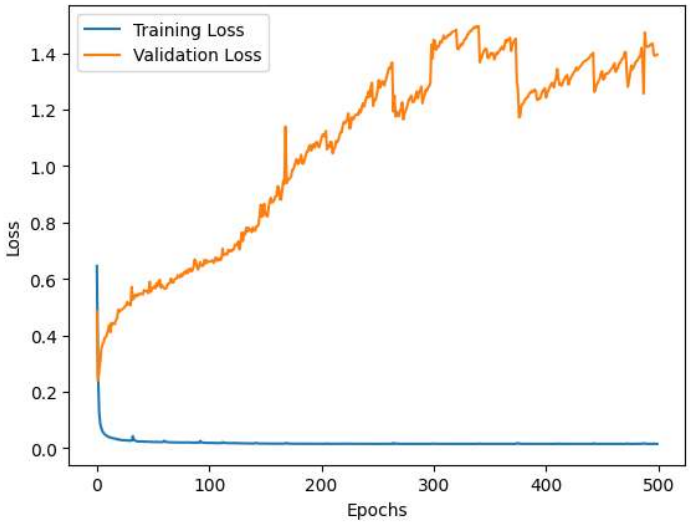


Figure 4.4.Validation Loss Over 500 Epochs with 0.001 Learning Rate

Evidently, the curves from the printed diagrams reveal a trend of increasing validation loss, indicating a deviation from the optimal point. As a response, the learning rate was modified to a lower value of 0.0001.

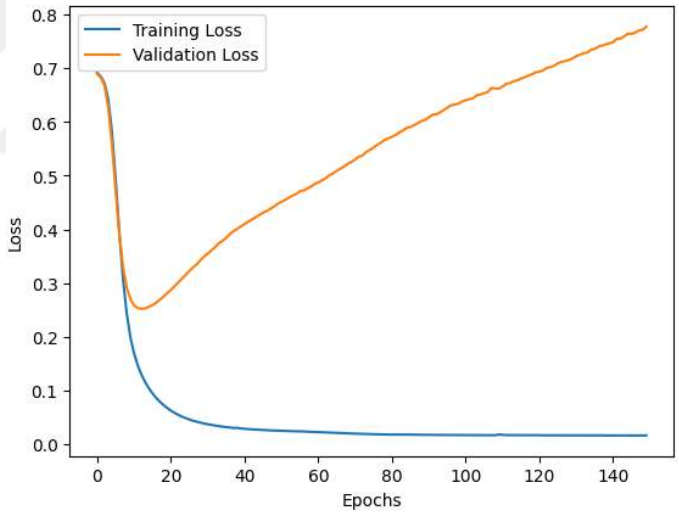


Figure 4.5.Validation Loss Over 150 Epochs with 0.0001 Learning Rate

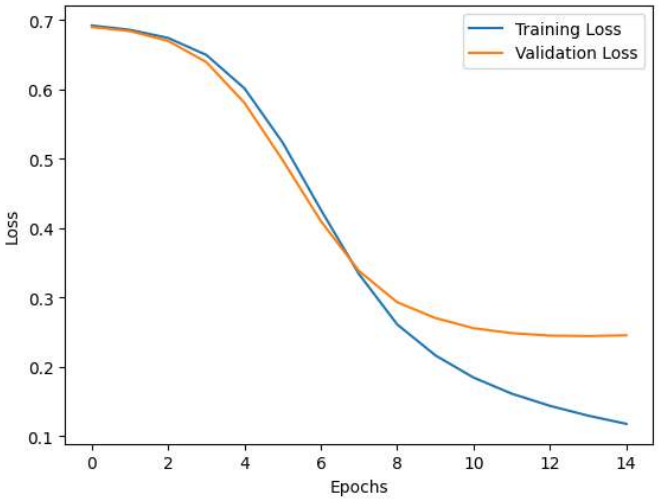

Figure 4.6. Validation Loss Over 15 Epochs with 0.0001 Learning Rate

In the subsequent diagrams, Figure 4.5 and Figure 4.6 display the validation loss over varying epochs, showcasing the effect of the revised learning rate. Notably, the revised learning rate of 0.0001 appears to bring the validation loss closer to the desired minimum using 10 epochs.

After we set the epochs number to 10 and the Learning Rate to 0.0001, we retrain the different architectures on the all used datasets, and the obtained results are shown in the Table 4.6 below:

Table 4.6. CNN Model Performance Comparison with Learning Rate 0.0001

| | English Dataset | | Turkish Dataset | | Arabic Dataset | | Balanced Turkish Dataset | |
|---|---|---|---|---|---|---|---|---|
| | val_accuracy | val_loss | val_accuracy | val_loss | val_accuracy | val_loss | val_accuracy | val_loss |
| (Gambäck and Sikdar, 2017) | 0.8921 | 0.3377 | 0.8389 | 0.9198 | 0.8237 | 0.8147 | 0.6939 | 0.9421 |
| (Kim, 2014) | 0.8736 | 0.4222 | 0.8334 | 1.2288 | 0.8241 | 0.9555 | 0.6862 | 1.3368 |
| 1.Architecture | **0.9090** | **0.2643** | **0.8394** | **0.5886** | **0.8416** | **0.4178** | 0.6777 | 0.6297 |
| 2.Architecture | 0.8967 | 0.2947 | 0.8227 | 1.4259 | 0.8273 | 0.5299 | 0.6785 | 0.6132 |

From the comprehensive comparison presented in Table 4.6, several noteworthy observations can be made regarding the performance of different architectures across diverse datasets with a learning rate of 0.0001. In addition, because of the imbalanced state of the Turkish dataset, it is not correct to pay attention to its accuracy measure.

1. **English Dataset:**
    - The 1.Architecture exhibited the highest accuracy of 0.9090, outperforming other models on the English dataset.
    - This architecture achieved this accuracy while maintaining a low loss of 0.2643.

2. **Turkish Dataset:**
    - The 1.Architecture also demonstrated notable performance on the Turkish dataset with an accuracy of 0.8394.
    - In terms of loss, it achieved 0.5886, showcasing good generalization.
3. **Arabic Dataset:**
    - The 1.Architecture maintained its strong performance on the Arabic dataset as well, with an accuracy of 0.8416.
    - Its low loss of 0.4178 further confirms its adaptability to different languages.
4. **Balanced Turkish Dataset:**
    - Once again, the 1.Architecture stood out with an accuracy of 0.6777, showcasing its consistency across datasets.
    - Its loss of 0.6297 reflects its ability to manage overfitting even in challenging cases.

These results collectively highlight the effectiveness of the 1.Architecture, particularly when employing a learning rate of 0.0001. Its ability to consistently achieve high accuracy across different datasets underscores its robustness and adaptability in multilingual offensive language classification tasks.

### 4.2. RNN-based Offensive Language Classification

To explore RNN-based offensive language classification, we start our investigation with the simplest RNN architecture. This architecture comprised an embedding layer, a simple RNN layer, and a Dense layer. The experiments primarily focused on the English dataset due to its structured nature.

The architecture we employ is as follows:

- embedding (Embedding) (None, max length of sequences, 300) 2497500 the Embedding layer is responsible for converting words or tokens in input sequences into dense vectors of fixed size (in this case, we start with 300 dimensions). And the input shape is the max length of sequences, indicating that we are working with sequences of length max length of sequences that is the max length of the sequences after the preprocessing operations on the train texts. and, 2,497,500 is the total number of parameters that need to be learned by the model during training to create these word embeddings.
- simple_rnn (SimpleRNN) (None, 128) 54912 recurrent neural network (RNN) layer takes the embedded sequences as input and processes them sequentially. the output shape (None, 128), which indicates that the layer produces 128-dimensional output for each input sequence. And this layer is used for capturing sequential dependencies in the data. "54912" represents the total number of trainable parameters in the SimpleRNN layer.

- dense (Dense) (None, 1) 129 the Dense layer is a fully connected layer that comes after the RNN layer. Indicate the output shape (None, 1), which implies that this layer produces a single output value (for binary classification). And the number of trainable parameters for this layer (129).

Initially, I set the number of epochs to ten, using an initial learning rate of 0.0001, consistent with the approach taken in the CNN model. The loss curve was plotted as shown below:

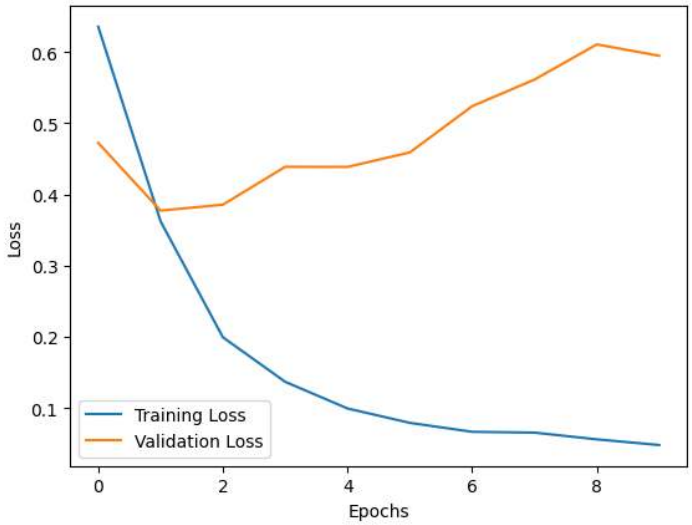


Figure 4.7. RNN Validation Loss Over 10 Epochs with 0.0001 Learning Rate

The Figure 4.7 shows the validation loss of an RNN model over 10 epochs with a learning rate of 0.0001. The validation loss decreases over the first few epochs, but it starts to increase after epoch 10. This is a sign of overfitting.

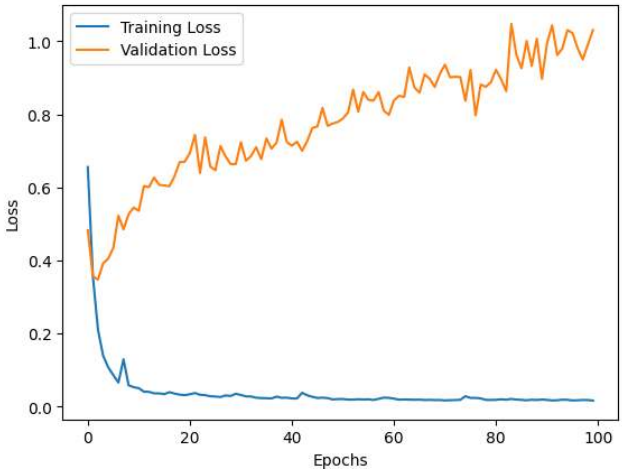


Figure 4.8. RNN Validation Loss Over 100 Epochs with 0.0001 Learning Rate

These experiments were conducted to prove that a large number of epochs will increase the validation loss. The results of the experiment support that hypothesis. However, it is also possible that the increase in validation loss is due to overfitting. To rule out overfitting, I try the following:

- Use a smaller learning rate.

After ten epochs, the validation accuracy reached 0.8334 with a validation loss of 0.5446. Extending the training to 100 epochs resulted in a validation accuracy of 0.7926 and a validation loss of 1.0308. In light of these results, I adjusted the learning rate to 0.0001, aligning it with the modification made in the CNN model. Similarly, I employed 10 epochs to facilitate comparison with the CNN experiments.

Hyperparameter tuning commenced by varying the dimensional vector number. The results are shown in Table 4.7 below:

Table 4.7. RNN Dimensional Vector Size Experiment Results

| Dimensional Vector Size | val_accuracy | val_loss |
|---|---|---|
| 50 | **0.8658** | **0.4334** |
| 100 | 0.8088 | 0.6931 |
| 200 | 0.6476 | 0.8572 |
| 300 | 0.8504 | 0.5819 |

Based on these results, the highest accuracy was achieved with 50-dimensional vectors, aligning with the CNN findings. To potentially create a hybrid model, a choice between 50 and 300-dimensional vectors was considered, with 50 being preferred for consistency.

The units' number of the RNN layer was subsequently adjusted, yielding the following outcomes in the Table 4.8:

Table 4.8. RNN Units Experiment Results

| Units Number | val_accuracy | val_loss |
|---|---|---|
| 32 | 0.8450 | 0.5868 |
| 64 | 0.8134 | 0.6494 |
| 100 | 0.6831 | 0.9424 |
| 128 | **0.8658** | **0.4334** |

The model exhibited the highest accuracy when employing 128 units in the RNN layer. And the resulted architecture is presented in the below Figure 4.9:

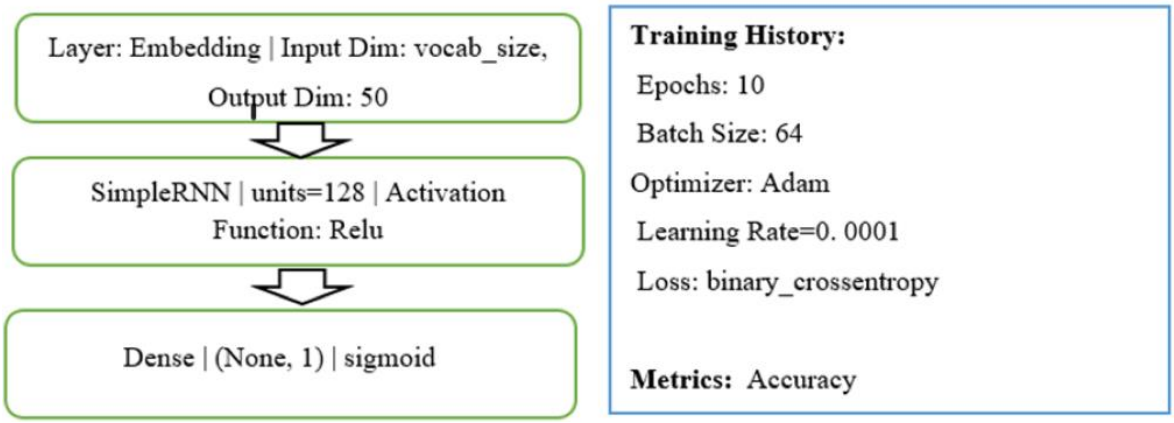


Figure 4.9. 1.Architecture of RNN

Expanding upon the first architecture, which represents the simplest form of an RNN, we start with a configuration consisting of an embedding layer, followed by a single Simple RNN layer equipped with 128 units, and a dense output layer with a sigmoid activation function. Our objective is to enhance model accuracy systematically, so we embark on an iterative journey of architectural development.

In the pursuit of refinement, we introduce Architecture 2. Here, we modify Architecture 1 by introducing an additional dense layer with 8 units and ReLU activation inspired by (DataTechNotes, 2019) while maintaining all other hyperparameters. This addition aims to add a degree of intricacy to the model's configuration and examine its effect on predictive accuracy.

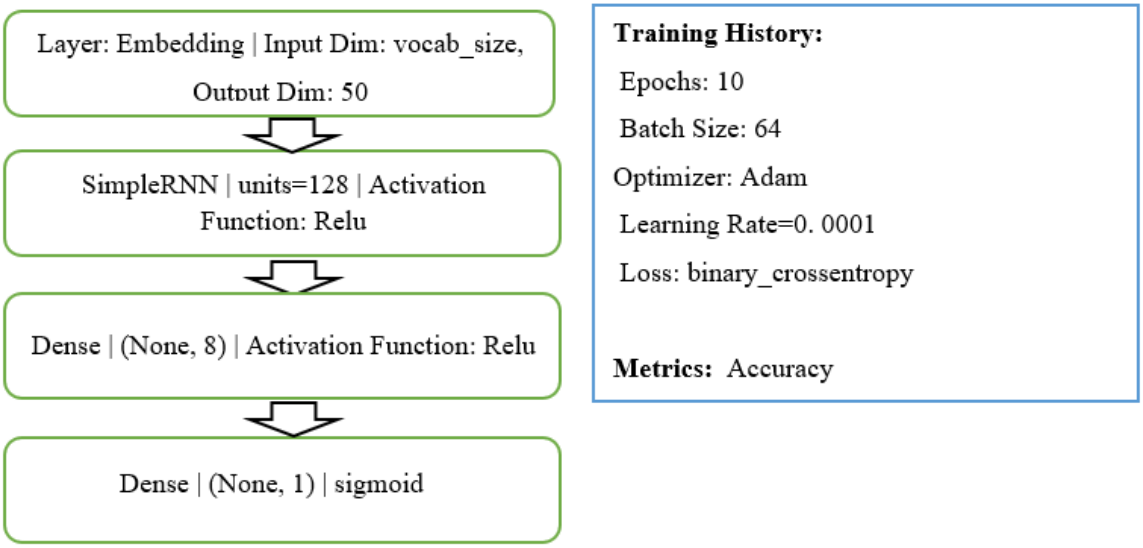


Figure 4.10. 2. Architecture of RNN

As we continue our quest for improved performance, we present Architecture 3. To create this version, we build upon the foundation of Architecture 2 by replacing the single Simple RNN layer with a Bidirectional Simple RNN layer as shown in Figure 4.11. This change allows us to explore bidirectional sequential processing, adding a layer of complexity.

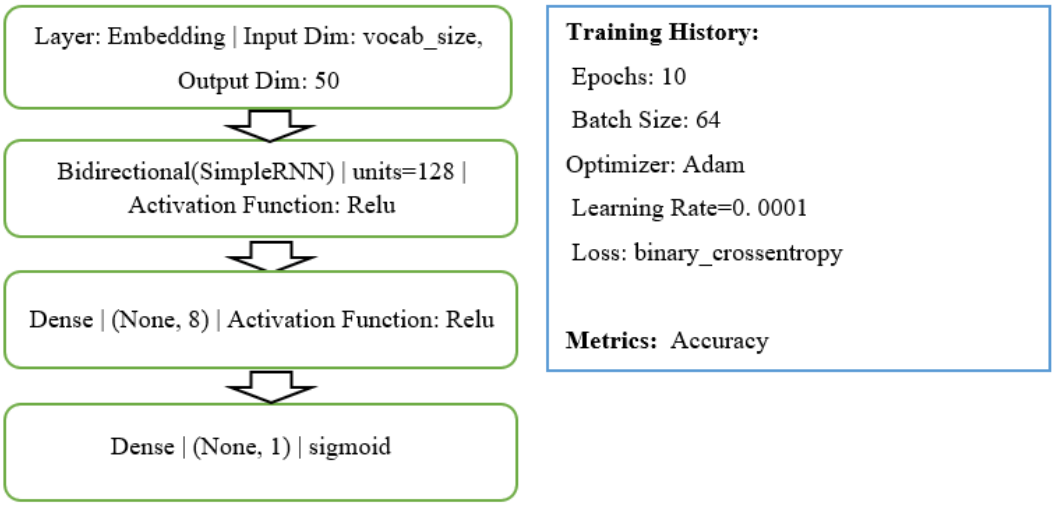


Figure 4.11. 3.Architecture of RNN

Our final architectural iteration, building upon the foundation of Architecture 3, we introduce 2 dropout layers with a dropout rate of 0.2 inspired by (Kcsener, 2019). These additions create a more intricate model configuration, allowing us to explore the impact of dropout regularization on our RNN architecture.

By systematically evolving our RNN architectures, we aim to uncover the nuanced interplay between model complexity and predictive accuracy.

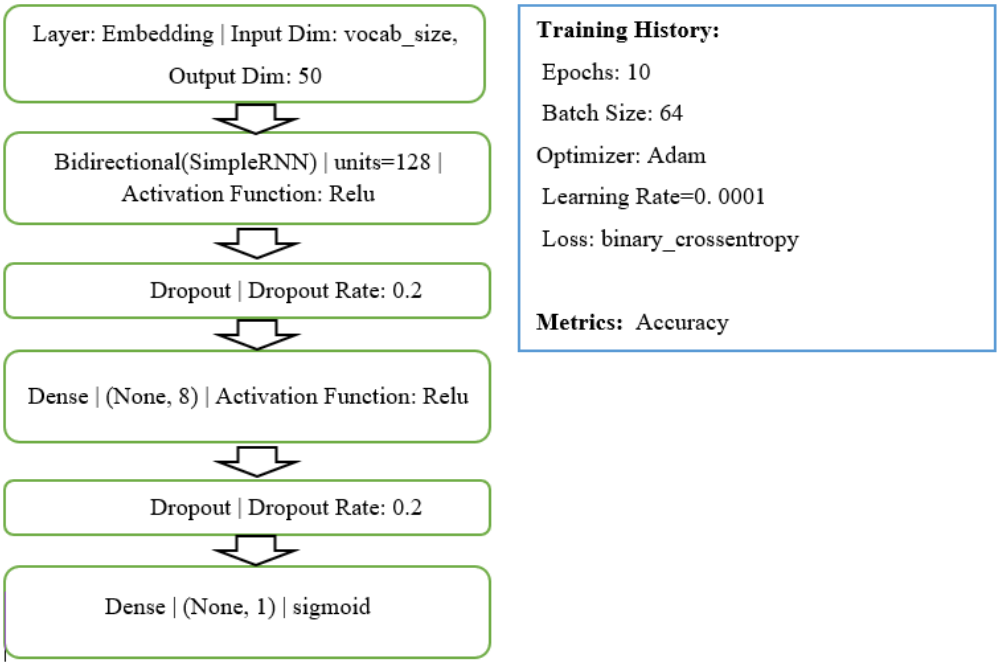


Figure 4.12. 4.Architecture of RNN

Their performances of the 4 RNN models, are summarized below in Table 4.9:

Table 4.9. RNN Architecture Experiment Results for English Dataset

| RNN Models | val_accuracy | val_loss |
|---|---|---|
| Architecture 1 | **0.8658** | 0.4334 |
| Architecture 2 | 0.8404 | 0.5285 |
| Architecture 3 | 0.8419 | 0.4096 |
| Architecture 4 | **0.8658** | 0.4803 |

Both Architecture 1 and Architecture 4 yielded the highest accuracy rates, with Architecture 1 chosen for its simplicity and because it has lower loss. We choose the first one to train it on the other datasets, because of that it is the simplest architecture and does not consume a lot of time for training the model, when we test the fourth architecture for the two Turkish datasets, we can see that overfitting happens because of the complexity of the architecture giving us very big validation loss values. The first architecture is tested on different datasets, it produces varying accuracy and loss values when applied to different datasets.

Table 4.10. 1. RNN Architecture Performance Across Different Datasets

| Dataset | val_accuracy | val_loss |
|---|---|---|
| English Dataset | 0.8658 | 0.4334 |
| Turkish Dataset | 0.7997 | 0.5016 |
| Arabic Dataset | 0.8214 | 0.4358 |
| Balanced Turkish Dataset | 0.5818 | 0.8411 |

It is important to note that training on the Arabic and Turkish datasets took approximately 45-55 minutes each. The extensive experimentation with RNN architectures underscores the importance of architecture modifications and hyperparameter tuning in achieving optimal results across different datasets. The selected architecture demonstrated robust performance while maintaining simplicity. In summary, the selected RNN architecture exhibited consistent performance across different datasets, showcasing its adaptability to varying linguistic and contextual characteristics. While achieving high accuracy on English and Arabic datasets, the architecture proved somewhat less effective on the Turkish datasets, both balanced and unbalanced. These results highlight the importance of dataset-specific fine-tuning and optimization to fully harness the architecture's potential.

### 4.3. LSTM-based Offensive Language Classification

In the pursuit of a simplified LSTM architecture, a modification was made to the existing 1. Architecture of RNN classifier by substituting the simple RNN layer with an LSTM layer while retaining other layers and hyperparameters. This approach aimed to create a preliminary LSTM architecture, akin to the one used in the work of (Pitsilis et al., 2018). It's noteworthy that in their research, Pitsilis and colleagues introduced an additional dense layer with 200 units after the LSTM layer.

Initially, training commenced with a learning rate of 0.001 over 100 epochs, as depicted in the accuracy curve in Figure 4.13:

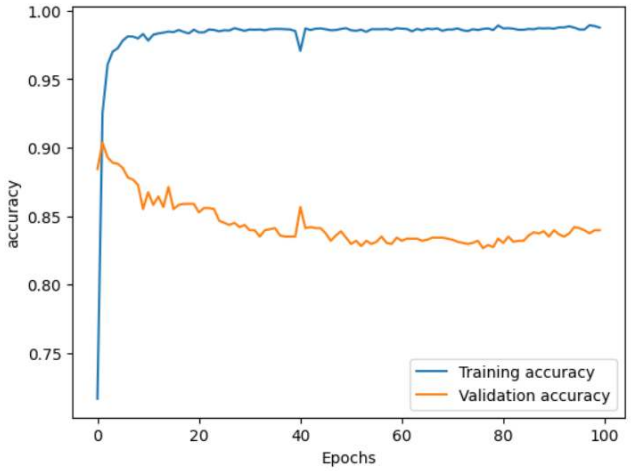


Figure 4.13. Accuracy Over 100 Epochs with 0.001 Learning Rate Using LSTM

The accuracy graph in Figure 4.14 demonstrates consistent accuracy levels of around 85%. Subsequently, lowering the learning rate to 0.0001 and extending the number of epochs to 100 produced the following curve:

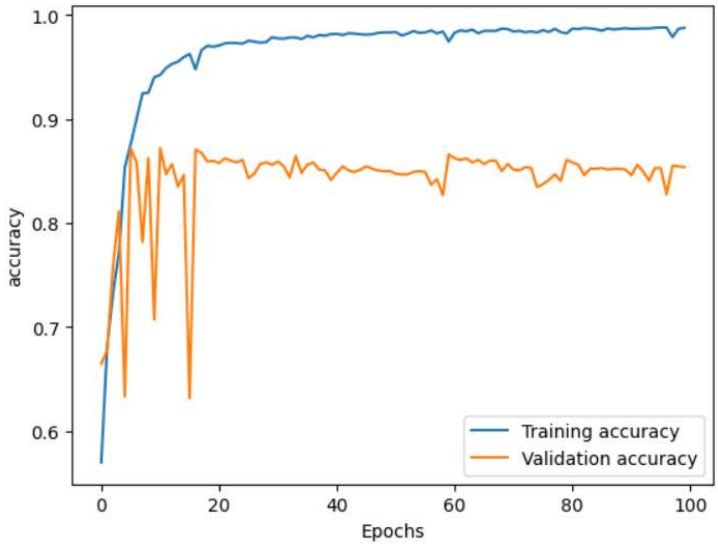


Figure 4.14. Validation Accuracy Over 10 Epochs with 0.0001 Learning Rate using LSTM

Interestingly, reducing the learning rate to 0.0001 led to a noticeable decrease in accuracy, potentially reaching approximately 65%. Consequently, the Figure 4.14 shows the validation accuracy over 10 epochs with a learning rate of 0.0001 using LSTM. The validation accuracy is the accuracy of the model on the validation data. A higher validation accuracy indicates that the model is performing better on the validation data. Therefore, the learning rate was set back to 0.001 for the LSTM model.

Hyperparameter tuning initiated by varying the dimensional vector size resulted in the following outcomes:

Table 4.11. LSTM Dimensional Vector Size Experiment Results

| Dimensional Vector | val_accuracy | val_loss |
|---|---|---|
| 50 | **0.8651** | **0.4487** |
| 100 | 0.8604 | 0.4983 |
| 200 | 0.8574 | 0.5783 |
| 300 | 0.8489 | 0.5924 |

As shown in Table 4.11, the highest accuracy was attained with 50-dimensional vectors. It's noteworthy that these findings resonate with the results from the RNN model, highlighting the significance of this vector size.

Continuing the experimentation, different numbers of units in the LSTM layer were explored, yielding the subsequent outcomes in Table 4.12:

Table 4.12. LSTM Units Experiment Results

| Units Number | val_accuracy | val_loss |
|---|---|---|
| 32 | 0.8558 | 0.5266 |
| 64 | 0.8643 | 0.6614 |
| 100 | 0.8520 | 0.6420 |
| 128 | **0.8651** | **0.4487** |

As shown in Table 4.12, the model exhibited the highest accuracy with 128 units in the LSTM layer, aligning with the trend observed in the RNN experiments. The obtained architecture is described in the Figure 4.15 below:

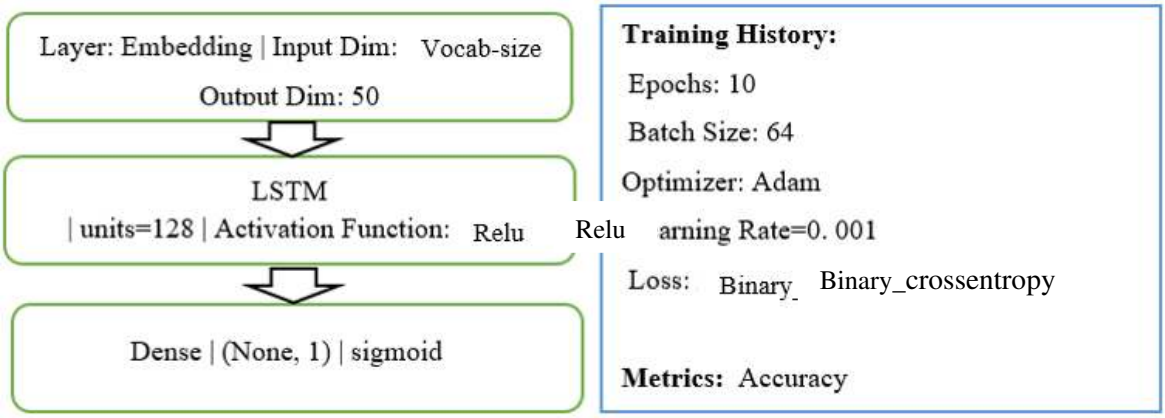


Figure 4.15. 1.Architecture of LSTM

Now we add the complexity of our first architecture of LSTM model trying to get higher accuracies so the description of the tested architectures as follow:

The first architecture of LSTM in the Figure 4.15**:**

1. **Input Layer:** The architecture starts with an embedding layer that takes the input text data. The embedding layer converts the input text into dense vectors of size 50.
2. **Hidden Layer:** A single LSTM layer follows the embedding layer. This LSTM layer has 128 units, which means it has 128 memory cells to capture sequential patterns in the data.
3. **Output Layer:** The LSTM layer is followed by a dense output layer with a sigmoid activation function. This layer produces a single output, which is useful for binary classification tasks.

In the pursuit of refinement, we introduce Architecture 2. Here, we modify Architecture 1 by introducing an additional dense layer with ReLU activation inspired by the architecture of used model in (Pitsilis et al., 2018) but in different units' number that is set to 8, while maintaining all other hyperparameters, as shown in Figure 4.16.

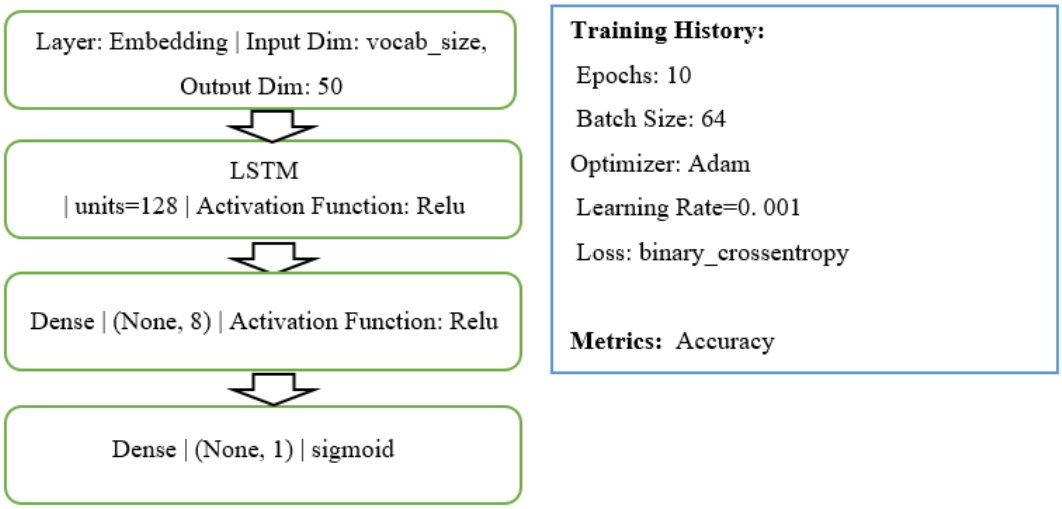


Figure 4.16. 2.Architecture of LSTM

3.architecture is built on Model 2, and the following components are added:

LSTM layer from Model 2 is replaced with bidirectional LSTM layer. A bidirectional LSTM processes sequences in both forward and backward directions, allowing the model to capture patterns in both directions of the sequence. Figure 4.17 shows the 3rd architecture.

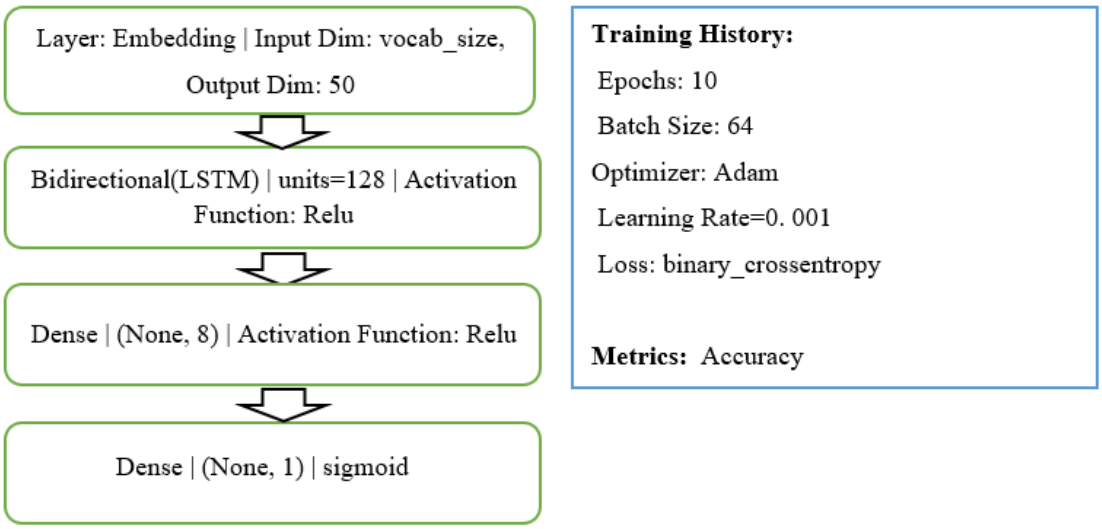


Figure 4.17. Architecture 3 of LSTM

4.architecture is built on Model 3 and the Added Components are follows:

Regularization (Dropout): Dropout layers are added after bidirectional LSTM layer, with a dropout rate of 0.2 and one another with the same rate after the first dense layer. Dropout helps prevent overfitting by randomly setting a fraction of the input units to zero during each training batch. the 4$^{th}$ architecture is summarized in Figure 4.18. In summary, the architecture progression involves increasing model complexity through the addition of more LSTM layers, bidirectional processing, and dropout layers. Each added component contributes to the model's capacity to learn and generalize patterns from the input sequence data.

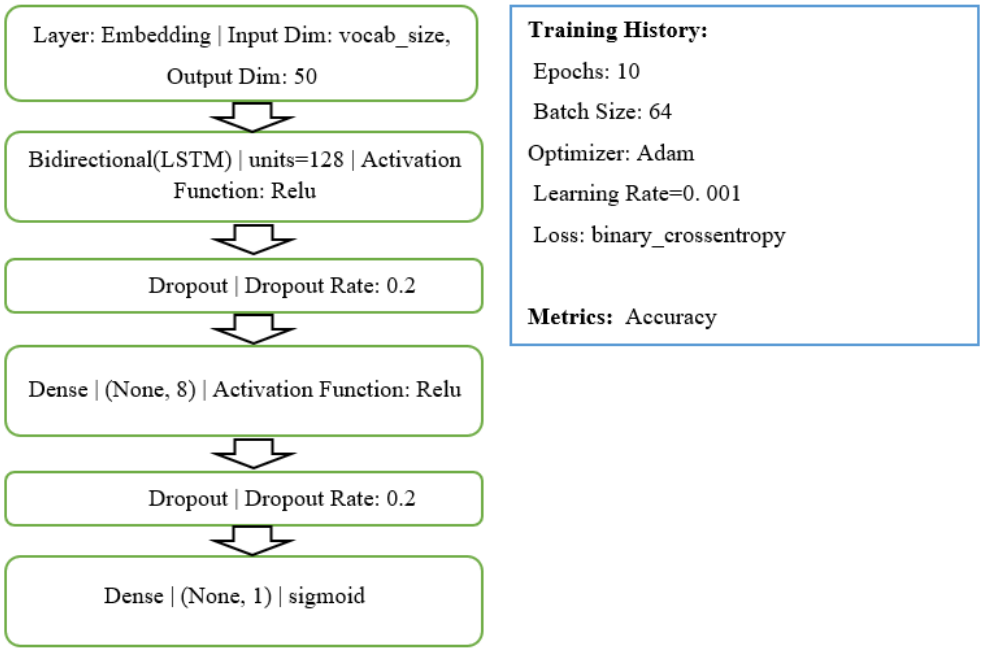


Figure 4.18. Architecture 4 of LSTM

As a final refinement, I expanded the architecture by adding two dropout layers to enhance generalization. The resulting performance of different LSTM architectures is summarized in the Table 4.13 below:

Table 4.13. LSTM Architectures Experiment Results for English Dataset

| LSTM Models | val_accuracy | val_loss |
|---|---|---|
| (Pitsilis, 2018) | 0.8450 | 0.6180 |
| Architecture 1 | **0.8689** | **0.4375** |
| Architecture 2 | 0.8574 | 0.4463 |
| Architecture 3 | 0.8512 | 0.5413 |
| Architecture 4 | 0.8635 | 0.4671 |

Based on the Table 4.13, Architecture 1 yielded the highest accuracy and the lowest loss, further validating the viability of the selected LSTM architecture.

After that we have been trying the simplest architecture on the Turkish and Arabic datasets for days, but it always gave us the same resulted accuracy as 0.4950 for Turkish dataset and 0.5050 for the Arabic one. After changing all hyperparameters without making any progress, then, by making some researches, we have many reasons that can lead to this statue, one of them was insufficient model complexity, that says the designed model architecture might not be able to capture the underlying patterns in the rows of data. So, we have increased the complexity of our architecture from a simple LSTM classifier to multilayers model, but there is still no different result. So, we have made a check on preprocessing methods until we have found the specification of the padding in the pad_sequences function, the parameter padding='post' is used to specify where to add padding in a sequence. When we have texts of different lengths, we often need to make them equal in length by adding padding. Padding means adding special tokens (typically with a value of 0 the same as in our codes ) to the sequences until they all have the same length.

By setting padding='post', we specify that the padding tokens should be added at the end (or "post") of each sequence. This means that the padding will come after the original content of the text. After removing that specification, the default state became set that is setting the padding as 'pre' resulting in the addition of padding tokens at the beginning of each sequence. The resulting accuracies had started to be changed. As a result, we can see removing the padding='post' parameter from the pad_sequences function in LSTM and another neural networks models that have memory resulted in higher accuracies for the Arabic and Turkish datasets. so, removing post-padding resulted in improved performance for the models. By allowing the models to access the complete sequences, including the original content at the end, models were able to better capture and utilize the contextual information throughout the entire sequence. This enhanced utilization of context contributed to the observed performance improvement.

Based on the Table 4.14, Architecture 1 yielded the highest accuracy and the lowest loss, further validating the viability of the selected LSTM architecture. The architecture's efficacy was then tested across different datasets are presented in Table 4.14.

Table 4.14. LSTM 1.LSTM Architecture Performance Across Different Datasets

| Dataset | val_accuracy | val_loss |
|---|---|---|
| English Dataset | 0.8651 | 0.4487 |
| Turkish Dataset | 0.7781 | 0.6590 |
| Arabic Dataset | 0.8311 | 0.5610 |
| Balanced Turkish Dataset | 0.6685 | 1.9807 |

While fitting our model to the Turkish dataset, we noticed something interesting during the first three epochs of training. The model achieved higher accuracy, around 85%, in those early epochs. This was because we used a different training speed, 0.001, compared to what we used for CNN and RNN, which was 0.0001, and the obtained high validation loss values show us there is an overfitting happens here. So, we realized we didn't need to train for 10 epochs here. The validation loss started going up after the second epoch. However, this quicker learning came at the cost of much longer training times – lasting several days. For the Arabic dataset, training was even longer, taking more than 10 hours.

After trying out different approaches in previous tests, we finally choose the best setup for each type of algorithm. We fine-tuned the learning rate to 0.001 and set the number of training epochs to 3. These changes were based on the settings that consistently gave us the best results. Then we used these optimized settings to retrain our chosen models on each dataset, so we can fairly compare how well they perform. Table 4.15 summarizes the results of the optimized architectures for each dataset. This careful process ensures that our selected models are ready to give their best performance across different situations.

Table 4.15. Algorithms' Performance on Multilingual Datasets

| Dataset | | CNN | RNN | LSTM |
|---|---|---|---|---|
| English Dataset | val_accuracy | **0.9075** | 0.8897 | 0.8913 |
| | val_loss | 0.2691 | 0.3181 | 0.3063 |
| Turkish Dataset | val_accuracy | 0.8373 | 0.8093 | **0.8400** |
| | val_loss | 0.5969 | 0.4828 | 0.6823 |
| Arabic Dataset | val_accuracy | 0.8487 | 0.8421 | **0.8537** |
| | val_loss | 0.3829 | 0.3831 | 0.3480 |
| Balanced Turkish Dataset | val_accuracy | **0.7016** | 0.5355 | 0.6669 |
| | val_loss | 0.6795 | 0.9954 | 0.8315 |

For the English Dataset: The CNN algorithm achieved the highest validation accuracy (0.9075), followed closely by the RNN (0.8897) and LSTM (0.8913) algorithms. The validation loss followed a similar trend, with CNN achieving the lowest loss (0.2691), followed by RNN (0.3181), and LSTM (0.3063).

For the Turkish Dataset: The LSTM algorithm achieved the highest validation accuracy (0.8400), followed by the CNN (0.8373), and then RNN (0.8093). The validation loss showed a similar pattern, with RNN having the lowest loss (0.4828), followed by CNN (0.5969), and LSTM (0.6823).

For the Arabic Dataset: The LSTM algorithm achieved the highest validation accuracy (0.8537), followed by CNN (0.8487), and then RNN (0.8421). The validation loss trend mirrored the accuracy results, with LSTM having the lowest loss (0.3480), followed by RNN (0.3831), and CNN (0.3829).

For the Balanced Turkish Dataset: The CNN algorithm achieved the highest validation accuracy (0.7016), followed by LSTM (0.6669), and then RNN (0.5355). In terms of validation loss, CNN had the lowest loss (0.6795), followed by LSTM (0.8315), and RNN (0.9954).

In summary, the specific algorithm that performs the best varies across different datasets. The CNN algorithm generally demonstrated competitive performance across the datasets, often achieving the highest accuracy and relatively low loss. The LSTM algorithm also showed consistent strong performance in terms of accuracy. The RNN algorithm, while generally being competitive, exhibited relatively higher loss values on some datasets.

When we trained the LSTM model on the Turkish dataset, it took the longest time around 110 minutes. In contrast, the RNN model's training finished in about 20 minutes, and the CNN model was even quicker, taking no more than 5 minutes.

For the Arabic dataset, things were a bit different. The CNN model's training time was just 5 minutes, while the RNN model took around 15 minutes. However, the LSTM model required more time, about 90 minutes, to complete its training. This information gives us an idea of how much time each model needs to learn from the data for these specific datasets.

We extended our evaluation to a multilingual dataset containing English, Turkish, and Arabic records. This dataset includes three columns: "Text," indicating the content of each record; "Language," indicating the language of the record, which helps us apply the appropriate preprocessing steps; and "Label," representing the classification label.

In this context, we applied the same set of models - CNN, RNN, and LSTM - that were previously optimized on individual datasets. The models were tested on this diverse multilingual dataset to measure their effectiveness in handling content in different languages.

The results of these tests are summarized in the Table 4.16 below:

Table 4.16. Performance of Algorithms on Multilingual Dataset

| Models | Multilingual Dataset | | |
|---|---|---|---|
| | Run Time | val_accuracy | val_loss |
| CNN | 1 min, 13 s | **0.8246** | 0.5219 |
| RNN | 3 min, 24 s | 0.7962 | 0.5879 |
| LSTM | 25 min, 17 s | 0.8177 | 0.5312 |

From this table, we can glean insights into how these algorithms perform when confronted with multilingual data. The CNN model demonstrated relatively swift processing times while maintaining competitive accuracy levels. On the other hand, both the RNN and LSTM models exhibited good accuracy, albeit with longer execution times. These results provide valuable context for selecting an appropriate algorithm for tasks involving content in multiple languages.

### 4.4. Feature Selection Exploration

In this section, we delved into the process of feature selection, aiming to refine our model's performance using the Arabic dataset, which offered a substantial amount of data. Our investigation involved testing three distinct feature selection methods: 1) Information Gain, 2) Chi-Squared, and 3) Correlation Coefficient.

While we testing these methods we use our chosen architecture of each classifier, that are 1.architecture of CNN from subsection 4.1, 1. Architecture of RNN subsection 4.2, and 1.architecture of LSTM described in subsection 4.3, and trying to obtain the highest accuracy and avoiding the overfitting, we set the learning rate to 0.001 and epochs number to 3 as we discussed in the previous subsection 4.3, The results are documented in the Table 4.17 below:

Table 4.17. Analysis of CNN with Feature Selection Methods on Arabic Dataset

| Feature Selection Method | Features Selection Time | Features Rate (%) | Model Training Time | val_accuracy | val_loss |
|---|---|---|---|---|---|
| Without F.S. | | 100 | 5 min | 0.8497 | 0.3709 |
| | | | | | |
| information gain | 5min 2s | 50 | 4min,22s | 0.7720 | 0.4911 |
| | | | | | |
| Chi-Squared | 1s | 50 | 4min,22s | 0.8496 | 0.3752 |
| | 1s | 25 | 3min,58s | 0.8469 | 0.3754 |
| | 1s | 10 | 4min,22s | 0.8135 | 0.4367 |
| | | | | | |
| Correlation Coefficient | 1s | 50 | 4min,25s | 0.8484 | 0.3785 |
| | 1s | 25 | 3min,58s | 0.8468 | 0.3716 |
| | 1s | 10 | 3min,40s | 0.8149 | 0.4311 |
| | | | | | |
| Chi-Squared and Corr. Coef | 1s | 25 | 3min,54s | **0.8501** | 0.3740 |
| | | | | | |
| İnfo.gain, and Chi-Squared | 5min,20s | 25 | 4min,16s | **0.8529** | 0.3652 |
| İnfo.gain, and Corr. Coef. | 5min,53s | 25 | 8min,16s | 0.8486 | 0.3741 |
| İnfo.gain, Chi-Sq. and Corr. Coef. | 5min 2s | 25 | 1min,3s | 0.8444 | 0.3777 |

The results highlight the highest accuracy rates through the integration of the (Information Gain - Chi-Squared) and (Chi-Squared - Correlation Coefficient) feature selection methods. In order to optimize our feature selection process and achieve improved training times while maintaining uniformity across our algorithms, we implemented a combination of two and three feature selection methods that we use in this thesis:

Here's a brief overview of how we combined these methods:

**Combination:** After applying each method independently, we combined their results by taking the **union** of the selected feature indices. This approach allowed us to include features that were deemed important by either of the two or three methods. By doing so, we aimed to capture a broader range of relevant features that could benefit our machine learning models.

The resulting set of selected feature indices formed the basis for our reduced feature set, which we used for training our machine learning algorithms. This combination approach enabled us

to achieve consistent and efficient results across various algorithm models. These methods were chosen due to our intent to optimize training times and achieve uniformity across our algorithms. Consequently, we implemented these chosen techniques across various algorithm models for consistent and efficient results.

We extended our exploration to encompass 25% feature subsets of the Arabic, Turkish, and English datasets. The following tables illustrate the outcomes of our investigation on each dataset:

Table 4.18. Models Performance on Arabic Dataset with 25% Features

| Feature Selection Method | 25% of Features on Arabic datasets | | | |
|---|---|---|---|---|
| | Algorithm | Training Time | val_accuracy | val_loss |
| Without feature selection | CNN | 5 min | 0.8497 | 0.3709 |
| | RNN | 13min,19s | 0.8421 | 0.3831 |
| | LSTM | 89min,10s | 0.8475 | 0.3686 |
| Chi-Squared and corre.Coef. | CNN | 3min,54s | 0.8501 | 0.3740 |
| | RNN | 5min,46s | 0.8500 | 0.3611 |
| | LSTM | 21min,38s | **0.8537** | 0.3498 |
| İnfo.gain, and Chi-Squared | CNN | 4min,12s | 0.8503 | 0.3747 |
| | RNN | 6min,51s | 0.8488 | 0.3574 |
| | LSTM | 28min,50s | 0.8518 | 0.3502 |

According to the results of the Table 4.18, although the feature selection methods did not lead to substantial accuracy improvements, they did result in a remarkable reduction in the time required to train the models. This efficiency can significantly impact the practical usability of the models in real-world scenarios, making them more responsive to rapid deployment and time-sensitive tasks. Therefore, the focus on training time efficiency through feature selection could prove to be a valuable aspect of model development.

Table 4.19. Models Performance on Turkish Dataset with 25% Features

| Feature Selection | 25% of Features on Turkish datasets | | | |
|---|---|---|---|---|
| | Algorithm | Training Time | val_accuracy | val_loss |
| Without feature selection | CNN | 5 min | 0.8373 | 0.5969 |
| | RNN | 20 min | 0.8093 | 0.4828 |
| | LSTM | 110 min | **0.8400** | 0.6823 |
| Chi-Squared and corr. Coef. | CNN | 1min,8s | 0.8269 | 0.5870 |
| | RNN | 3min,38s | 0.8000 | 0.5962 |
| | LSTM | 35min,27s | 0.8238 | 0.6838 |
| İnfo.gain, and Chi-Squared | CNN | 1min,12s | 0.8288 | 0.6193 |
| | RNN | 4min,9s | 0.7924 | 0.5076 |
| | LSTM | 45min,22s | 0.8253 | 0.6815 |

From the previous table, we can see that, feature selection methods such as Chi-Squared and Correlation Coefficient, as well as Information Gain and Chi-Squared, seem to have an impact on the model performance. In some cases, they slightly improve accuracy, while in others, they lead to a reduction in accuracy or similar performance.

We continue our experiments by applying feature selection methods to the English dataset and observed their outcomes in terms of training time and accuracy. The results are detailed in Table 4.20 below:

Table 4.20. Models Performance on English Dataset with 25% Features

| Feature Selection Method | 25% of Features on English datasets | | | |
|---|---|---|---|---|
| | | Training Time | val_accuracy | val_loss |
| Without feature selection methods | CNN | 2s | 0.9075 | 0.2691 |
| | RNN | 3s | 0.8897 | 0.3181 |
| | LSTM | 10s | 0.8913 | 0.3063 |
| Chi-Squared and corr. Coef. | CNN | 2s | 0.8635 | 0.3582 |
| | RNN | 2s | 0.8489 | 0.4096 |
| | LSTM | 5s | 0.8751 | 0.3673 |
| İnfo.gain, and Chi-Squared | CNN | 2s | 0.8705 | 0.3525 |
| | RNN | 2s | 0.8558 | 0.3891 |
| | LSTM | 5s | 0.8658 | 0.3544 |

Using the (Chi-Squared - Correlation Coefficient) feature selection approach, we observed a reduction in the required training time for all models across Turkish and Arabic datasets. While the accuracy remained on par or slightly improved in the Turkish and Arabic dataset, the English and Turkish datasets displayed a marginal decline of around 1 to 3% across all models.

Table 4.21. Models Performance on Multilingual Dataset with 25% Features

| | 25% of Features on Multilingual datasets | | | |
|---|---|---|---|---|
| | | training Time | val_accuracy | val_loss |
| Without feature selection | CNN | 1 min, 13 s | **0.8246** | 0.5219 |
| | RNN | 3 min, 24 s | 0.7962 | 0.5879 |
| | LSTM | 25 min, 17 s | 0.8177 | 0.5312 |
| Chi-Squared and correlation Coefficient. | CNN | 45s | 0.7984 | 0.5752 |
| | RNN | 1min,16s | 0.7931 | 0.5384 |
| | LSTM | 5min,33s | 0.7918 | 0.5685 |
| İnfo.gain, and Chi-Squared | CNN | 52s | **0.8243** | 0.5375 |
| | RNN | 1min,47s | 0.8099 | 0.4875 |
| | LSTM | 8min,27s | 0.8146 | 0.5385 |

We extended our investigation to the multilingual dataset, incorporating English, Turkish, and Arabic records. The goal was to analyze the impact of feature selection methods on training time and model performance. As showcased in Table 4.22, algorithms on Multilingual datasets, we observed significant reductions in training time when employing feature selection methods.

Table 4.22. Models Performance on Balanced Turkish Dataset with 25% Features

| | 25% of Features on Balanced Turkish datasets | | | |
|---|---|---|---|---|
| | | training Time | val_accuracy | val_loss |
| Without feature selection | CNN | 11s | **0.7016** | 0.6795 |
| | RNN | 37 s | 0.5289 | 0.6803 |
| | LSTM | 4 min, 7 s | 0.6669 | 0.8315 |
| Chi-Squared and correlation Coefficient. | CNN | 7s | 0.6523 | 0.7342 |
| | RNN | 14s | 0.5898 | 0.6805 |
| | LSTM | 1min,8s | 0.6315 | 0.9648 |
| İnfo.gain, and Chi-Squared | CNN | 8s | 0.6808 | 0.7276 |
| | RNN | 18s | 0.5505 | 0.6761 |
| | LSTM | 1min,45s | 0.6338 | 0.8276 |

We found that while accuracy experienced a slight dip, the expedited training time was particularly notable, with LSTM models exhibiting the most significant improvements. These results underscore the trade-off between accuracy and training time optimization.

In summary, our investigations into feature selection methods revealed their effectiveness in substantially reducing training times, especially in scenarios with complex models and multilingual datasets. Although some accuracy compromise was observed.

### 4.5. RoBERTa-based Offensive Language Classification

After working extensively with neural network classifiers, we start utilizing the RoBERTa language model. This model was subjected to testing on our diverse range of datasets, with each dataset being paired with an appropriate version of RoBERTa tailored to its specific language. This approach ensured that the RoBERTa classifier was finely attuned to the linguistic nuances of each dataset's language, thereby optimizing its performance.

To pave the way for effective model training, a sequence of preliminary actions was undertaken. These encompassed text data cleansing, unifying data format, formulation of preprocessing functions, determination of a maximal text length, and formulation of a dataset class to streamline the organization of training data. These preparatory steps laid a solid foundation for the subsequent phases.

For the English dataset, the following operations were conducted:

Dataset Preparation: A custom dataset class, named "TextClassificationDataset," was created to structure and tokenize the text data for efficient training. The RoBERTa tokenizer and model were loaded for English language analysis.

Model Training: The training dataset was organized into batches using a DataLoader. The model was trained through 3 epochs the same number used in our previous classifiers, and a classifier layer was added for the final classification task.

Evaluation: The model's performance was evaluated on the testing dataset, calculating test loss and accuracy.

For the Turkish dataset, similar operations were performed, but with the following differences:

Model and Tokenizer: The RoBERTa model and tokenizer used were specifically designed for the Turkish language, enhancing language-specific understanding.

For the Arabic dataset, the operations were similar, and the key difference was:

Model and Tokenizer: The XLM-RoBERTa model and tokenizer specifically tailored for Arabic were employed for better language adaptation.

For the multilingual dataset, the following operations were conducted:

Model and Tokenizer: The XLM-RoBERTa model and tokenizer designed for multilingual text genre classification were utilized. This model is intended to handle multiple languages and text genres effectively.

Important Differences:

The choice of model and tokenizer varied based on the dataset's language: RoBERTa for English and Turkish, and XLM-RoBERTa for Arabic.

The multilingual dataset employed a model designed for handling diverse languages and text genres.

While the overall process, including preprocessing, training, and evaluation, remained consistent across the datasets, the language-specific models ensured better linguistic adaptation, potentially leading to improved performance for each language.

In the context of our research, we utilized various RoBERTa-based models with distinct configurations and datasets. The following Table 4.23 summarizes the key characteristics of these models and datasets:

Table 4.23. RoBERTa Model Configurations

| | **RoBERTa Model Configuration** | | | | |
|---|---|---|---|---|---|
| | **English dataset** | **Arabic dataset** | **Turkish dataset** | **Balanced Turkish dataset** | **Multilingual dataset** |
| Model | RoBERTa | XLM-RoBERTa | RoBERTa | RoBERTa | XLM-RoBERTa |
| Initialization | 'roberta-base' (base version) | 'Davlan/xlm-roberta-base-finetuned-arabic' (XLM-RoBERTa model fine-tuned for Arabic) | burakaytan/roberta-base-turkish-uncased | burakaytan/roberta-base-turkish-uncased | classla/xlm-roberta-base-multilingual-text-genre-classifier |
| Output size | 2 | 2 | 2 | 2 | 2 |
| Learning Rate | 1e-5 | 1e-5 | 1e-5 | 1e-5 | 1e-5 |
| Epochs | 3 | 3 | 3 | 3 | 3 |
| Loss Function | Cross-Entropy Loss | Cross-Entropy Loss | Cross-Entropy Loss | Cross-Entropy Loss | Cross-Entropy Loss |
| Optimization | AdamW | AdamW | AdamW | AdamW | AdamW |
| Tokenizer | RoBERTaTokenizer | XLM-RoBERTa Tokenizer | RoBERTa Tokenizer for the Turkish language. | RoBERTa Tokenizer for the Turkish language | XLM-RoBERTa Tokenizer |
| Data Loaders | DataLoader for training and testing | DataLoader for training and testing | DataLoader for training and testing | DataLoader for training and testing | DataLoader for training and testing |
| Max Length | maximum sequence length | maximum sequence length | maximum sequence length | maximum sequence length | maximum sequence length |
| Padding and Truncation | 'max_length' parameter | 'max_length' parameter | 'max_length' parameter | 'max_length' parameter | 'max_length' parameter |

These configurations allowed us to perform experiments across various languages and datasets, providing valuable insights into the performance of RoBERTa-based models in different linguistic contexts.

The Table 4.24 below, provides a comparison of the different datasets based on their run time and performance using the RoBERTa model with 3 training epochs, and 64 batch size the same

as in the neural network classifiers architectures that we test in this thesis. This table includes information about validation accuracy and validation loss for each dataset.

Table 4.24. Comparison Using RoBERTa on 3 Epochs with Preprocessing 64 batch size

| Dataset | Run Time | val_accuracy | val_loss |
|---|---|---|---|
| English Dataset | 15 min, 44 s | 0.9044 | 0.2375 |
| Turkish Dataset | 128 min, 17 s | 0.8070 | 0.5036 |
| Balanced Turkish Dataset | 34 min, 59 s | 0.7400 | 0.5652 |
| Arabic Dataset | 1105 min | 0.8590 | 0.3473 |
| Multilingual Dataset | 277 min | 0.8144 | 0.4153 |

By analyzing the Table 4.24, which compares the performance of different datasets using the RoBERTa model trained for 3 epochs with preprocessing and a batch size of 64:

The "English Dataset" achieved the highest validation accuracy (0. 9044) and the lowest validation loss (0.2375), indicating strong performance and effective generalization to unseen data.

The "Turkish Dataset" achieved a relatively low validation accuracy of 0.8070.

The "Balanced Turkish Dataset" had a shorter run time (21 min, 5 s) compared to the "Turkish Dataset," and it achieved a lower validation accuracy (0.7400) and higher validation loss (0.5652) but we can see although to the small number of its data, RoBERTa could be successful in achieving higher accuracy comparing the NN classifiers that we test in this thesis.

Because of being large dataset, the "Arabic Dataset" had a longer run time (1105 min) compared to the other datasets. Despite the longer training time, achieved a moderate validation accuracy of 0.8590 and a relatively lower validation loss of 0.3473.

The "Multilingual Dataset" had a run time of 277 min, and achieved a validation accuracy of 0.8144, with a validation loss of 0.4153.

In summary, the results from the RoBERTa model training on these datasets indicate that it performs well on English, Arabic, and Multilingual data, achieving relatively high validation accuracy and low validation loss. However, for the Turkish datasets, the model's performance is reasonable but may benefit from further optimization, especially in handling imbalanced data in the case of the "Balanced Turkish Dataset." Additionally, the training time varies significantly between datasets, with the Arabic dataset requiring the longest training time.

Loshchilov and Hutterin (2018) say that a smaller batch size can lead to improved generalization by diversifying mini-batches, allowing the model to better adapt to various data samples, that can improve the performance of the model. And because of the memory problem that we face using RoBERTa when set big number of batch size, I repeat the fine tuning of the model using smaller batch size that is 8, and obtain really interesting results which are different from the obtained from the previous experiment, we present the obtained results in the Table 4.25:

Table 4.25. Comparison Using RoBERTa on 3 Epochs with Preprocessing and batch size=8

| Dataset | Run Time | val_accuracy | val_loss |
|---|---|---|---|
| English Dataset | 47 min, 56 s | 0.9136 | 0.2587 |
| Turkish Dataset | 345 min, 33 s | 0.8701 | 0.3617 |
| Balanced Turkish Dataset | 21 min, 5 s | 0.7639 | 0.5528 |
| Arabic Dataset | 1411 min | 0.8625 | 0.3393 |
| Multilingual Dataset | 238 min, 9 s | 0.8232 | 0.3900 |

By analyzing the Table 4.25, we can draw several conclusions about the performance of the different datasets using the RoBERTa model:

The "English Dataset" achieved the highest validation accuracy (0.9136) and the lowest validation loss (0.2587), indicating strong performance and effective generalization to unseen data.

The "Turkish Dataset" had a longer run time (345 min, 33 s) compared to the other datasets. Despite the longer training time, it achieved a relatively good validation accuracy of 0.8701.

The "Balanced Turkish Dataset" had a shorter run time (21 min, 5 s) compared to the "Turkish Dataset," and it achieved a lower validation accuracy (0.7639) and higher validation loss (0.5528).

The "Arabic Dataset" achieved a moderate validation accuracy of 0.8625 and a relatively lower validation loss of 0.3393.

The "Multilingual Dataset" had a run time of 238 min, 9 s and achieved a validation accuracy of 0.8232, with a validation loss of 0.3900.

**Comparative Analysis of using batch sizes of 64 and 8**

We compare the results of the two experiments using batch sizes of 64 and 8:

- The batch size of 8 consistently outperformed or matched the performance of the batch size of 64 in terms of validation accuracy.
- Smaller batch size (8) seemed to be more effective in improving model generalization and achieving higher accuracies across different datasets.
- Despite longer training times for the batch size of 8, the trade-off in terms of accuracy and loss appeared to be favorable in most cases.

Overall, these comparisons suggest that a smaller batch size of 8 may be advantageous for your specific experiments, as it tends to lead to higher validation accuracies, potentially improving the model's generalization to different datasets.

**Comparative Analysis of Results using Neural networks and RoBERTa**

By analyzing the obtained results using Neural networks CNN, RNN, and LSTM beside to RoBERTa language model we can describe the results below :

1. **English Dataset:**
    - Among the NN models (CNN, RNN, LSTM), CNN achieved the highest validation accuracy of 0.9075.
    - RoBERTa achieved the highest validation accuracy of 0.9136, outperforming the NN models.
    - RoBERTa also achieved the lowest validation loss of 0.2587, indicating efficient learning.
2. **Turkish Dataset:**
    - Among the NN models, LSTM achieved the highest validation accuracy of 0. 8400.
    - RoBERTa achieved the highest validation accuracy of 0.8701, showcasing strong performance.
    - RoBERTa also achieved a relatively lower validation loss of 0.3617, indicating effective learning.
3. **Arabic Dataset:**
    - The validation accuracy for all models (CNN, RNN, LSTM) is quite close, with LSTM slightly outperforming the others with 0.8537.
    - RoBERTa achieved a validation accuracy of 0.8625, indicating solid performance.
    - RoBERTa also demonstrated a competitive validation loss of 0.3393.
4. **Balanced Turkish Dataset:**
    - Among the NN models, CNN achieved the highest validation accuracy of 0.7016.
    - RoBERTa achieved the highest validation accuracy of 0.7639, demonstrating its effectiveness.
    - RoBERTa also achieved a competitive validation loss of 0.5528.
5. **Multilingual Dataset:**
    - Among the NN models, CNN achieved the highest validation accuracy of 0.8246.
    - RoBERTa achieved a validation accuracy of 0.8232 and a validation loss of 0.3900.

**Key Takeaways:**

- RoBERTa consistently achieved the highest validation accuracy across most datasets, showcasing its effectiveness for various languages and tasks.
- NN models like CNN also demonstrated competitive results, especially for certain datasets.
- Validation losses for RoBERTa were generally lower than those of the NN models, indicating better model convergence.

In summary, the results suggest that RoBERTa is a strong candidate for various language tasks due to its consistent high accuracy and relatively low validation loss across different datasets.

### 4.6. RoBERTa-Based Offensive Language Classification without Preprocessing

In an effort to explore the impact of preprocessing on the performance of RoBERTa-based offensive language classification, we conducted experiments without applying preprocessing operations to the datasets. The objective was to assess the intrinsic capabilities of the RoBERTa model in handling raw text without any additional linguistic refinement.

For this investigation, we follow a similar experimental setup as described in Section 4.5. However, instead of implementing the preprocessing steps outlined earlier, we directly input the raw text data into the RoBERTa model. This approach aimed to evaluate how well RoBERTa can adapt to the various language nuances and patterns within the unprocessed text.

The following results were obtained through experimentation with the RoBERTa model without preprocessing operations:

Here we test two experiments using batch sizes of 8 and 64, the obtained results are represented in the below two tables Table 4.25 and Table 4.26:

Table 4.26. Comparison of Datasets Using RoBERTa without Preprocessing 8 batch size

| Dataset | Run Time | val_accuracy | val_loss |
|---|---|---|---|
| English Dataset | 4 min, 55 s | 0.9275 | 0.1875 |
| Turkish Dataset | 29 min, 30 s | 0.8668 | 0.3359 |
| Balanced Turkish Dataset | 39 min, 18 s | 0.7670 | 0.5181 |
| Arabic Dataset | 2081 min, 35 s | 0.9260 | 0.1960 |
| Multilingual Dataset | 631 min, 15 s | 0.8486 | 0.3629 |

Table 4.27. Comparison of Datasets Using RoBERTa without Preprocessing 64 batch size

| Dataset | Run Time | val_accuracy | val_loss |
|---|---|---|---|
| English Dataset | 28 min, 12 s | 0.9229 | 0.2001 |
| Turkish Dataset | 79 min, 4 s | 0.8609 | 0.3408 |
| Balanced Turkish Dataset | 27 min, 49 s | 0.7446 | 0.5430 |
| Arabic Dataset | 1505 min, 19 s | 0.9196 | 0.2042 |
| Multilingual Dataset | 339 min, 3 s | 0.8285 | 0.3829 |

**Comparative Analysis with Batch Sizes 8 and 64 without Preprocessing**

1. **English Dataset:**
    - In both batch sizes (8 and 64), the English dataset achieved high validation accuracy, with a slightly higher accuracy in the batch size of 8. The validation loss was also lower in the batch size of 8.
2. **Turkish Dataset:**
    - The batch size of 8 resulted in a higher validation accuracy compared to the batch size of 64 for the Turkish dataset. However, the validation loss was lower in the batch size of 64.
3. **Balanced Turkish Dataset:**
    - The batch size of 8 outperformed the batch size of 64 in terms of validation accuracy for the Balanced Turkish Dataset.
4. **Arabic Dataset:**
    - Both batch sizes yielded high validation accuracy for the Arabic dataset, with a slightly higher accuracy in the batch size of 8. The validation loss was also lower in the batch size of 8.
5. **Multilingual Dataset:**
    - The batch size of 8 achieved a higher validation accuracy for the Multilingual dataset, while the batch size of 64 resulted in a lower validation accuracy.

- Smaller batch sizes (8) generally led to higher validation accuracies and lower validation losses across most datasets.
- The choice of batch size appears to have a substantial impact on model performance, with smaller batch sizes consistently performing better in terms of accuracy.

Because we get the higher accuracies using batch size 8, we compare the done two experiments, where the first one has preprocessing operations in contrast the another that does not have, and we can see the follow:

1. **English Dataset:**
    - With preprocessing, the English dataset achieved a slightly lower validation accuracy (0.9136) compared to without preprocessing (0.9275). However, the validation loss was slightly higher with preprocessing (0.2587 vs. 0.1875).
2. **Turkish Dataset:**
    - With preprocessing, the Turkish dataset had a higher validation accuracy (0.8701) compared to without preprocessing (0.8668). The validation loss was slightly higher with preprocessing (0.3617 vs. 0.3359).

3. **Balanced Turkish Dataset:**
    - With preprocessing, the balanced Turkish dataset had a lower validation accuracy (0.7639) compared to without preprocessing (0.7670). The validation loss was higher with preprocessing (0.5528 vs. 0.5181).
4. **Arabic Dataset:**
    - With preprocessing, the Arabic dataset had a slightly lower validation accuracy (0.8625) compared to without preprocessing (0.9260). The validation loss was higher with preprocessing (0.3393 vs. 0.1960).
5. **Multilingual Dataset:**
    - With preprocessing, the Multilingual dataset had a slightly lower validation accuracy (0.8232) compared to without preprocessing (0.8486). The validation loss was slightly higher with preprocessing (0.3900 vs. 0.3629).
    - Preprocessing seemed to have a mixed impact on the different datasets, with some datasets benefiting from it in terms of accuracy and others performing slightly better without preprocessing.
    - Preprocessing generally resulted in higher validation loss for most datasets.
    - Preprocessing significantly reduced the run time for all datasets Except the datasets in Turkish language, making the training process much faster for those languages.

### 4.7. RoBERTa-CNN Offensive Language Classification

In this thesis, we combine the optimized architectures of CNN and the used RoBERTa for each language, to create a new ensemble model. This ensemble model brings together the strengths of both architectures. And the obtained results are shown in the Table 4.28.

Table 4.28. Comparison of Datasets Using RoBERTa-CNN

| Dataset | Run Time | val_accuracy | val_loss |
|---|---|---|---|
| English Dataset | 32 min, 25 s | 0.9221 | 0.2266 |
| Turkish Dataset | 135 min, 21 s | 0.8679 | 0.3406 |
| Balanced Turkish Dataset | 27 min, 25 s | 0.7539 | 0.5345 |
| Arabic Dataset | 3010 min, 21 s | **0.9271** | 0.1926 |
| Multilingual Dataset | 663 min, 13 s | 0.8473 | 0.3795 |

By analyzing the Table 4.28, we can draw several conclusions about the performance of the different datasets using the RoBERTa-CNN model:

The "Arabic Dataset" achieved the highest validation accuracy (0.9271) and the lowest validation loss (0. 1926), indicating strong performance and effective generalization to unseen data, but it also had the longest run time (3010 min, 21 s).

The "English Dataset" achieved a moderate validation accuracy of 0. 9221and a relatively lower validation loss of 0.2266.

The "Turkish Dataset" achieved a relatively good validation accuracy of 0.8679.

The "Balanced Turkish Dataset" had a shorter run time (27 min, 25 s) compared to the "Turkish Dataset," and it achieved a relatively good validation accuracy (0.7539) and higher validation loss (0.5345).

The "Multilingual Dataset" had a run time of 663 min, 13 s and achieved a validation accuracy of 0.8473, with a validation loss of 0.3795.

Table 4.29. Performance Comparison of Various Classifiers Across Multiple Datasets

| Dataset | Measure | CNN | RNN | LSTM | RoBERTa | RoBERTa -CNN |
|---|---|---|---|---|---|---|
| English Dataset | val_accuracy | 0.9075 | 0.8897 | 0.8913 | **0.9275** | 0.9221 |
| | val_loss | 0.2691 | 0.3181 | 0.3063 | 0.1875 | 0.2266 |
| Turkish Dataset | val_accuracy | 0.8373 | 0.8093 | 0.8400 | 0.8668 | **0.8679** |
| | val_loss | 0.5969 | 0.4828 | 0.6823 | 0.3359 | 0.3406 |
| Arabic Dataset | val_accuracy | 0.8487 | 0.8421 | 0.8537 | 0.9260 | **0.9271** |
| | val_loss | 0.3829 | 0.3831 | 0.3480 | 0.1960 | 0.1926 |
| Balanced Turkish Dataset | val_accuracy | 0.7016 | 0.5355 | 0.6669 | **0.7670** | 0.7539 |
| | val_loss | 0.6795 | 0.9954 | 0.8315 | 0.5181 | 0.5345 |
| Multilingual Dataset | val_accuracy | 0.8246 | 0.7962 | 0.8177 | **0.8486** | 0.8473 |
| | val_loss | 0.5219 | 0.5879 | 0.5312 | 0.3629 | 0.3795 |

The Table 4.29 compares the performance of our five classifiers, namely CNN, RNN, LSTM, RoBERTa and RoBERTa-CNN, that gave the highest accuracies in this thesis, across five diverse datasets: English, Turkish, Arabic, Balanced Turkish, and Multilingual. RoBERTa consistently outshines the other classifiers in terms of validation accuracy, with the highest accuracy observed in the English (0.9275), while RoBERTa-CNN achieved the highest accuracy in Arabic (0.9271), and Turkish (0.8679) datasets. Additionally, RoBERTa and RoBERTa based ensemble demonstrate the lowest validation loss across most datasets, highlighting RoBERTa's robustness and effectiveness in various text classification tasks. This consistent high performance suggests that RoBERTa is a

compelling choice for text classification tasks in different languages and domains, making it a noteworthy candidate for a wide range of natural language processing applications.

### 4.8. Comparative Analysis and Contributions

In this section, we elucidate the unique contributions of our study by comparing it with prior research endeavors in the realm of offensive language identification. We underscore the distinguishing features and parallels between our work and earlier studies, emphasizing the significance of our findings.

#### 4.8.1. Creation of a Diverse Arabic Dataset

Our primary contribution lies in the creation of a substantial Arabic dataset, which aggregates more than 200,000 instances from various social media platforms. This dataset encompasses a mosaic of Arabic dialects, reflecting the rich linguistic diversity present on these platforms. This initiative addresses the scarcity of resources for Arabic dialects in offensive language detection, offering a valuable resource for future research in this domain.

#### 4.8.2. Creation of a Multilingual Dataset

In tandem with our Arabic dataset, we have created a multilingual dataset that encompasses text samples from English, Turkish, and Arabic. A noteworthy feature of this dataset is its class balance across languages, ensuring a fair and equitable evaluation. This contribution facilitates cross-lingual research and provides insights into the universality of offensive language detection across different linguistic contexts.

#### 4.8.3. Exploration of Simplified Neural Network Architectures

Our study advances the field by demonstrating that competitive results can be achieved through streamlined neural network architectures. We meticulously explored and tested various configurations of CNN, RNN, and LSTM models while applying straightforward hyperparameter tuning. These simplified architectures yielded accuracy levels that are on par with or even surpassing those reported in related works.

#### 4.8.4. Feature Selection Efficiency

In an effort to optimize model efficiency, we employed three distinct feature selection methods: information gain, chi-square, and correlation coefficient. Surprisingly, we found that selecting only 25% of the dataset's features, using any of (information gain and chi-square) or (correlation coefficient and chi-square), yielded near-identical or similar accuracies. This discovery underscores the potential for resource-efficient offensive language detection systems.

#### 4.8.5. RoBERTa and XLM-RoBERTa: Batch Size and Preprocessing Insights

Our study includes an in-depth investigation into the use of RoBERTa and XLM-RoBERTa models, considering different batch sizes. Intriguingly, we observed that smaller batch sizes led to higher accuracy rates, a finding that can significantly impact model efficiency and resource allocation. Furthermore, we tested these models without preprocessing operations and consistently achieved superior results, demonstrating the effectiveness of these models in handling unprocessed data.

#### 4.8.6. RoBERTa's Dominance Over Traditional Neural Networks

One of the central outcomes of our study is the empirical evidence demonstrating the superiority of RoBERTa over traditional neural networks, particularly for languages with limited linguistic resources. RoBERTa, having been trained on extensive corpora for each language, exhibited exceptional performance across all languages in our evaluation.

In summary, our comparative analysis underscores the substantial contributions of our study in the field of offensive language identification. From the creation of diverse datasets to the optimization of neural network architectures and the successful application of advanced language models, our work not only advances the state of the art but also enriches the resources available for future research in this critical domain.

#### 4.8.7. RoBERTa-CNN Ensemble Classifier

In addition to the aforementioned contributions, this study introduces a novel ensemble classifier, RoBERTa-CNN, which combines the strengths of the RoBERTa language model and Convolutional Neural Network (CNN). The RoBERTa-CNN ensemble classifier exhibits competitive performance, achieving the highest accuracy in offensive language detection for Arabic and Turkish datasets. This innovative approach highlights the potential for further improving classification performance by leveraging the complementary abilities of different deep learning architectures. By incorporating this ensemble classifier, our research extends the repertoire of effective techniques in the realm of offensive language identification, contributing to the development of more robust and accurate models.

## 5. CONCLUSIONS

In this study, we conducted experiments on a system aimed at classifying offensive language in social media texts using a deep-learning approaches, namely Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM), in addition we also use a language model named RoBERTa on the same datasets. The goal of the classifiers was to assign each comment into one of two categories: offensive or non-offensive.

Multiple architectures of each classifier were trained using different datasets, and the hyperparameters were carefully tuned to achieve optimal performance. We explored various combinations of hyperparameters to enhance the model's accuracy. However, it's important to note that increasing the complexity of the model architecture and using additional textual tools does not always guarantee higher accuracy in deep neural networks. The process of hyperparameter tuning requires careful consideration and experimentation to obtain a model that exhibits high performance.

RoBERTa achieved the highest validation accuracy across most datasets on the different languages, showcasing its effectiveness for various languages and tasks. But CNN also demonstrated competitive results, without the big need to the resources of memory as RoBERTa does. Furthermore, our introduction of the RoBERTa-CNN ensemble classifier resulted in competitive performance, particularly excelling in the Arabic and Turkish datasets. This ensemble approach harnesses the strengths of both RoBERTa and CNN, offering a promising avenue for further improving offensive language detection.

In our future endeavors, we plan to conduct additional experiments and expand our multilingual and Arabic datasets to gather more comprehensive and diverse data. One of our main objectives is to share the Arabic dataset we have meticulously prepared for this thesis as an open-source resource on the internet. By making our dataset freely accessible, we aim to support and contribute to other research studies in the field.

Moreover, we intend to explore the application of advanced text embedding techniques to enhance the detection of offensive language. These techniques have the potential to provide more meaningful representations of the text data, which could lead to higher accuracy in offensive language detection. Through these improvements, we aspire to contribute to the development of more effective and sophisticated methods for combating using offensive language in various languages, ultimately fostering a safer and more inclusive online environment for all users.

In conclusion, conducting multiple experiments and iterating on the model design and hyperparameter settings are essential for achieving the desired performance in offensive language classification tasks. This study highlights the importance of systematic exploration and optimization to develop an effective model.

**REFERENCES**


Activestate, 2022 What is Scikit-Learn in Python? https://www.activestate.com/resources/quick-reads/what-is-scikit-learn-in-python/

Alakrot, A., Murray, L., & Nikolov, N. S., 2018. Dataset Construction for the Detection of Anti Social Behaviour in Online Communication in Arabic. Procedia Computer Science, 142, 174-181. DOI: 10.1016/j.procs.2018.10.473. CC BY-NC-ND 4.0 License.

Al-Tufayl, Q. A. D. & Al-Ghizzy, M. J. D., 2022. A Linguistic Study of Offensive Language in Online Communication Chatgroups. International Journal of Linguistics Studies, 8(2):170 175.

Alsafari, S., Sadaoui, S., & Mouhoub, M., 2020. Deep Learning Ensembles for Hate Speech Detection. In 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 00087). DOI: 10.1109/ICTAI50040.2020.00087.

Banoula M., 2023. What is Tensorflow? Deep Learning Libraries & Program Elements https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow [accessed: 28.07.2023].

Banoula, M., 2021. Introduction to Long Short-Term Memory(LSTM). https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm [accessed: 16.08.2023].

Bharath K., 2020. Everything You Need To Know About Jupyter Notebooks https://towardsdatascience.com/everything-you-need-to-know-about-jupyter-notebooks-10770719952b [accessed: 28.07.2023].

Bozyigit, A., Utku, S., & Nasibov, E., 2018. Sanal Zorbalık İçeren Sosyal Medya Mesajlarının Tespiti. In 2018 3rd International Conference on Computer Science and Engineering (UBMK) (pp. 8566529). DOI: 10.1109/UBMK.2018.8566529. Dokuz Eylul University.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam,

Brownlee, J., 2021. How to Use Word Embedding Layers for Deep Learning with Keras. https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ [accessed: 16.08.2023].

Chowdhury, A. G., Didolkar, A., Sawhney, R., and Shah, R. R., 2019. ARHNet: Leveraging Community Interaction For Detection Of Religious Hate Speech In Arabic.

Chowdhury, S. A., Mubarak, H., Abdelali, A., Jung, S., Jansen, B. J., and Salminen, J., 2020. A Multi-Platform Arabic News Comment Dataset for Offensive Language Detection. In Proceedings of the International Conference on Language Resources and Evaluation (LREC'20).

Coban, Ö., Özel, S. A., & Inan, A., 2023. Detection and Cross-domain Evaluation of Cyberbullying in Facebook Activity Contents For Turkish. Acm Transactions On Asian And Low-Resource Language Information Processing, 22(4), 10.1145/3580393. [SCI-Expanded, Scopus].

Çelik, T., 2022. What Is a Language Model? https://www.deepset.ai/blog/what-is-a-language-model [accessed: 28.07.2023].

DataReportal., 2023. Global social media statistics. https://datareportal.com/social-media-users[accessed: 28.07.2023].

DataTechNotes., 2019. RNN Example with Keras SimpleRNN in Python https://www.datatechnotes.com/2018/12/rnn-example-with-keras-simplernn-in.html [accessed: 28.07.2023].

Devlin, J., Chang, M-W., Lee, K., & Toutanova, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 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), (pp. 4171–4186), Minneapolis, Minnesota. Association for Computational Linguistics. DOI: 10.18653/v1/N19-1423.

Elman, J. L., 1990. Finding Structure in Time. Cognitive Science, 14(2), 179-211. DOI: 10.1016/0364-0213(90)90002-E.

Elouali, A., Elberrichi, Z., & Elouali, N., 2020. Hate Speech Detection on Multilingual Twitter Using Convolutional Neural Networks. Revue d'Intelligence Artificielle, Vol. 34, No. 1, 81-88. DOI: 10.18280/ria.340111.

Essefar, K., Ait Baha, H., El Mahdaouy, A., El Mekki, A., & Berrada, I., 2023. OMCD: Offensive Moroccan Comments Dataset. Language Resources and Evaluation, June 2023. DOI: 10.1007/s10579-023-09663-2.

Gambäck, B., & Sikdar, U. K., 2017. Using Convolutional Neural Networks to Classify Hate-Speech. In Proceedings of the First Workshop on Abusive Language Online, (pp. 85-90), Vancouver, BC, Canada. Association for Computational Linguistics. DOI: 10.18653/v1/W17-3013..

Ghelani, S., 2019. Text Classification — RNN's or CNN's? https://towardsdatascience.com/text-classification-rnns-or-cnn-s-98c86a0dd361.

Glasgow, G. P. (2006). Bad Language: Are Some Words Better than Others? (CC BY). DOI: 10.7916/salt.v6i2.1551. Kanda University of International Studies.

Goel, B., & Sharma, R., 2019. USF at SemEval-2019 Task 6: Offensive Language Detection Using LSTM With Word Embeddings. In Proceedings of the 13th International Workshop on Semantic Evaluation, (pp. 796-800), Minneapolis, Minnesota, USA. Association for Computational Linguistics. DOI: 10.18653/v1/S19-2139.

Guyon, I., & Elisseeff, A., 2003. An Introduction to Variable and Feature Selection. Journal of Machine Learning Research, 3, 1157-1182. Submitted 11/02; Published 3/03.

Haddad, B., Orabe, Z., Al-Abood, A., & Ghneim, N., 2020. Arabic Offensive Language Detection with Attention-based Deep Neural Networks. In Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection, (pp. 76-81), Marseille, France. European Language Resource Association.

Hochreiter, S., & Schmidhuber, J., 1997. Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. DOI: 10.1162/neco.1997.9.8.1735. Published: 01 November 1997.

Huynh, T. V., Nguyen, V. D., Nguyen, K. V., Nguyen, N. L-T., & Nguyen, A. G-T., 2019. Hate Speech Detection on Vietnamese Social Media Text using the Bi-GRU-LSTM-CNN Model. VLSP Workshop 2019. arXiv:1911.03644 [cs.CL]. DOI: 10.48550/arXiv.1911.03644.

IBM, (n.d.). What are recurrent neural networks? https://www.ibm.com/topics/recurrent-neural-networks [accessed: 28.07.2023].

IBM, (n.d.). What is natural language processing (NLP)? https://www.ibm.com/topics/natural-language-processing [accessed: 28.07.2023]. InfoWorld, 2023. Visual Studio Code https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensiblecode-editor.html

Janakiev, N., (n.d.). Practical Text Classification With Python and Keras. https://realpython.com/python-keras-text-classification/#convolutional-neural-networks-cnn [accessed: 01.06.2023].

Jurafsky, D., and Martin, J. H., 2019. Speech and Language Processing (2nd ed.). Pearson.

Karayigit, H., Aci, C., & Akdagli, A., 2021. Detecting Abusive Instagram Comments in Turkish Using Convolutional Neural Network and Machine Learning Methods. Expert Systems with Applications, 174(14), 114802. DOI: 10.1016/j.eswa.2021.114802.

Kalita, D., 2022. A Brief Overview of Recurrent Neural Networks (RNN). https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/#:~:text=A%20Deep%20Learning%20approach%20for,by%20a%20deep%20feedforward%20model. [accessed: 16.08.2023].

Kcsener., 2019. Recurrent Neural Network (RNN) Tutorial. [Online Tutorial]. Kaggle. URL: https://www.kaggle.com/code/kcsener/8-recurrent-neural-network-rnn-tutorial.[Accessed: 16.08.2023].

Keras, (n.d.). Embedding layer. https://keras.io/api/layers/core_layers/embedding/ [accessed: 10.07.2023].

Khezzar, R., Moursi, A., & Al Aghbari, Z., 2023. arHateDetector: Detection of Hate Speech from Standard and Dialectal Arabic Tweets. Discover Internet Things, 3(1), 1. DOI: 10.1007/s43926-023-00030-9.

Kim, Y., 2014. Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), (pp. 1746-1751), Doha, Qatar. Association for Computational Linguistics. DOI: 10.3115/v1/D14-1181.

Korde, V., 2012. Text classification and classifiers. International Journal of Artificial Intelligence & Applications, 3(2): 85- 99.

Kumari, S., 2020. Social Media Multilingual Offensive Text Identification and Categorization using Neural Network Models. Proceedings of SemEval-2020 Task 12.

Kwaik, K. A., Saad, M., Chatzikyriakidis, S., Dobnik, S., & Johansson, R., 2020. An Arabic Tweets Sentiment Analysis Dataset (ATSAD) using Distant Supervision and Self Training. In Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection, 1–8. Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020. European Language Resources Association (ELRA), licensed under CC-BY-NC.

LeCun, Y., Bengio, Y., and Hinton, G., 2015. Deep learning. Nature, 521(7553), 436-444.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278- 2324. doi:10.1109/5.726791.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D., 1989. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4), 541-551.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. DOI: 10.1109/5.726791.

Liu, H., & Setiono, R., 1995. Chi2: Feature Selection and Discretization of Numeric Attributes. Tools with Artificial Intelligence, 1995. Proceedings. Seventh International Conference on. IEEE Xplore. DOI: 10.1109/TAI.1995.479783.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., &

Lutkevich, B., 2020. BERT language model. https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model#:~:text=BERT%2C%20which%20stands%20for%20Bidirectional,calculated%20based%20upon%20their%20connection. [accessed: 16.08.2023].

Loshchilov, I., & Hutter, F., 2017. Decoupled Weight Decay Regularization. In International Conference on Learning Representations, University of Freiburg, Freiburg, Germany.

Mohaouchane, H., Mourhir, A., & Nikolov, N. S., 2019. Detecting Offensive Language on Arabic Social Media Using Deep Learning. In 2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS). DOI: 10.1109/SNAMS.2019.8931839.

Mulki, H., Haddad, H., Ali, C. B., & Alshabani, H., 2019. L-HSAB: A Levantine Twitter Dataset for Hate Speech and Abusive Language. In Proceedings of the Third Workshop on Abusive Language Online, (pp. 111-118). Florence, Italy: Association for Computational Linguistics. DOI: 10.18653/v1/W19-3512.

Narkhede, S., 2018. Understanding Confusion Matrix. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 [accessed: 16.08.2023].

NumPy, (n.d.). What is NumPy? https://numpy.org/doc/stable/user/whatisnumpy.html

P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D., 2020. Language Models are Few-Shot Learners. In 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada. arXiv:2005.14165 [cs.CL]. DOI: 10.48550/arXiv.2005.14165.

Pitsilis, G. K., Ramampiaro, H., & Langseth, H., 2018. Detecting Offensive Language in Tweets Using Deep Learning. Applied Intelligence, 48(12), 4730-4742. DOI: 10.1007/s10489-018-1242-y.

Porter, M., 2006. The Porter Stemming Algorithm. https://www.tartarus.org/~martin/PorterStemmer/ [accessed: 28.07.2023].

Porter, M., 2014. Snowball. http://snowball.tartarus.org/ [accessed: 28.07.2023].

Phung, T. M., 2021. A review of pre-trained language models: from Bert, Roberta, To Electra, Deberta, BigBird, and more https://tungmphung.com/a-review-of-pre-trained-language-models-from-bert-roberta-to- electra-deberta-bigbird-and-more/ [accessed: 01.09.2023].

Ranasinghe, T., & Hettiarachchi, H., 2020. BRUMS at SemEval-2020 Task 12: Transformer Based Multilingual Offensive Language Identification in Social Media. In Proceedings of the Fourteenth Workshop on Semantic Evaluation (pp. 1906–1915). DOI: 10.18653/v1/2020.semeval-1.251.

Stoyanov, V., 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692.

Sadiq, S., Mehmood, A., Ullah, S., Ahmad, M., Choi, G. S., & On, B. W., 2021. Aggression detection through deep neural model on Twitter. Future Generation Computer Systems, 114, 120-129. [https://doi.org/10.1016/j.future.2020.07.050].

Saxena, S., 2021. What is LSTM? Introduction to Long Short-Term Memory. https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/. [accessed: 16.08.2023].

Saxena, S., 2020. Understanding Embedding Layer in Keras. https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce. [accessed: 16.08.2023].

Tanase, M. A., Cercel, D. C., & Chiru, C., 2020. UPB at SemEval-2020 Task 12: Multilingual Offensive Language Detection on Social Media by Fine-tuning a Variety of BERT-based Models. In Proceedings of the Fourteenth Workshop on Semantic Evaluation (pp. 2222–2231). International Committee for Computational Linguistics.

Uban, A.-S., & Dinu, L. P., 2019. On Transfer Learning for Detecting Abusive Language Online. In Advances in Computational Intelligence (pp. 688-700). Lecture Notes in Computer Science, Volume 11506. International Work-Conference on Artificial Neural Networks.

United Nations. (n.d.). What is hate speech? https://www.un.org/en/hate-speech/understanding-hate-speech/what-is-hate-speech?gclid=CjwKCAjwzo2mBhAUEiwAf7wjkiiA57NQYSy1RMU9kZM25ckxT9ELvRtSHqB6B_9TOoSwfL0BMNlG0RoCXDQQAvD_BwE [accessed: 28.07.2023].

Wei, B., Li, J., Gupta, A., Umair, H., Vovor, A., & Durzynski, N., 2021. Offensive Language and Hate Speech Detection with Deep Learning and Transfer Learning. ArXiv. https://doi.org/10.48550/arXiv.2108.03305.

Witten, I. H., & Frank, E., 2005. Data Mining: Practical Machine Learning Tools and Techniques (2nd ed.). Morgan Kaufmann Publishers.

Wikipedia ,2023. pandas.https://en.wikipedia.org/wiki/Pandas_(software)

Wikipedia ,2023. Python (programming language).

https://en.wikipedia.org/wiki/Python_(programming_language)

Win, T. Z., & Kham, N. S. M., 2020. Information Gain Measured Feature Selection to Reduce High Dimensional Data. In ICCA 2019 Proceedings (pp. 79-84). University of Computer Studies, Yangon.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., & Kumar, R., 2019. SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval). In Proceedings of the 13th International Workshop on Semantic Evaluation (pp. 75–86). Minneapolis, Minnesota, USA: Association for Computational Linguistics.

Zampieri, M., Nakov, P., Rosenthal, S., Atanasova, P., Karadzhov, G., Mubarak, H., Derczynski, L., Pitenis, Z., & Çöltekin, Ç., 2020. SemEval-2020 Task 12: Multilingual Offensive Language Identification in social media (OffensEval 2020). In Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval).

Zhang, Z., Robinson, D., & Tepper, J., 2018. Detecting Hate Speech on Twitter Using a Convolution GRU Based Deep Neural Network. In The Semantic Web (pp. 745–760). European Semantic Web Conference.

**CURRICULUM VITAE**

**Name and Surname:** Mahmud BİRECİKLİ

**Education**

| **Degree** | **Institution** | **Graduation Year** |
|---|---|---|
| Undergraduate | Çukurova University | 2020 |
| High School | Musa Ben Nosair, Aleppo | 2012 |

**Experience**

| **Year** | **Work Place** | **Title** |
|---|---|---|
| 2019-Present | Söz dijital | Software Engineer |

# APPENDICES

| Authors | Dataset | Lang. | Subject of Study | Methods | Classification Methods | Obtained Accuracies |
|---|---|---|---|---|---|---|
| Gambäck and Sikdar (2017) | English Twitter hate-speech dataset | Eng | Hate speech classification using CNN models | Feature embeddings, character n-grams, and CNN | CNN model utilizing word2vec embeddings | 78.3% F-score |
| Sadiq et al (2020) | Cyber-troll dataset | Eng | Aggression detection using CNN-LSTM and CNN-BiLSTM models | Feature extraction, feature selection, CNN-LSTM, and CNN-BiLSTM | CNN-LSTM and CNN-BiLSTM models | 92% accuracy |
| Pitsilis (2018) | tweets that have been labeled as either neutral, sexist, or racist | Eng | Effectiveness of LSTM for hate speech detection | LSTM model | LSTM | 84.2% accuracy |
| Wei et al. (2021) | public tweet dataset of 24,783 tweets | Eng | Toxic online speech detection | Experiments with BI-LSTM models using empty embeddings and pre-trained Glove embeddings, transfer learning with models like BERT, DistilBert, and GPT-2 | BI-LSTM | Over 92% accuracy |
| Goel et al. (2019) | OffensEval dataset | Eng | Offensive language detection using LSTM with Word Embeddings | Offensive/Profane Word List, LSTM with Word Embeddings | LSTM | Approximately 88.75% accuracy |
| Mohaouchane et al (2019) | Arabic YouTube comments | AR | Effectiveness of different neural network models for hate speech detection | CNN, Bi-LSTM, Bi-LSTM with attention mechanism, and a combined CNN-LSTM architecture | CNN, Bi-LSTM, Bi-LSTM with attention, combined CNN-LSTM | recall rates: 82.24%, 80.97%, 81.51%, and 83.46% Respectively |
| Alsafari et al. (2020) | fine-grained labeled textual Arabic corpus | AR | Hate speech and offensive speech detection using DNNs | CNN and Bi-GRU models, ensemble with FastText, MBert, and AraBert | CNN, Bi-GRU models, ensemble with FastText, MBert, AraBert | F1 score of 0.859, 0.75 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Haddad et al. (2020) | OSACT4 shared task on offensive language detection | AR | Offensive language and hate speech detection using DNNs | CNN and Bi-GRU models, both enhanced with attention layers | CNN, Bi-GRU models, enhanced with attention layers | F1 score of 0.859 for the task of offensive language detection and an F1 score of 0.75 for the task of hate speech detection |
| Essefar (2023) | Offensive Moroccan Comments Dataset (OMCD) | AR | Automatic detection of offensive tweets in the Arabic language | Various machine learning and deep learning models, including MARBERT, LSTM, and BiLSTM | MARBERT LSTM BiLSTM | MARBERT: 84.17%, LSTM: 78.82%, BiLSTM: 77.63% |
| Chowdhury et al (2019) | Twitter dataset | AR | Detecting religious hate speech in Arabic social media | ARHNet, a novel approach that combines Arabic Word Embeddings and Social Network Graphs | LSTM + CNN + NODE2VEC | F1-score of 0.78 |
| Khezzar et al. (2023) | arHateDataset | AR | Identifying hate speech in Arabic tweets | arHateDetector framework, which uses multiple machine learning (ML) and deep learning (DL) models, including AraBERT | AraBERT | AraBERT: 93% |
| Karayiğit et al. (2021) | Abusive Turkish Comments (ATC) | TR | Abusive comments detection on Instagram | CNN, Naive Bayes, SVM, Decision Tree, Random Forest, Logistic Regression, AdaBoost, XGBoost | Convolutional Neural Network (CNN), Naive Bayes, Support Vector Machine, Decision Tree, Random Forest, Logistic Regression, Adaptive Boosting (AdaBoost), and eXtreme Gradient Boosting (XGBoost) | Micro-averaged F1-score: 0.974, Macro-averaged F1-score: 0.973, Kappa-value: 0.946 |
| Coban et al. (2023) | Turkish users on the Facebook social media platform. | TR | Cyberbullying Detection using Machine Learning (ML) and Deep Learning | Logistic Regression, Multinomial Naive Bayes, Support Vector Machines, Convolutional Neural Networks, | Logistic Regression, Naive Bayes, Support Vector Machines, CNN, RNN, Bert | F1-score: 0.928 on CF-B dataset. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | (DL) methods | Recurrent Neural Networks, BERT | | |
| Elouali et al. (2019) | Multiling. Twitter dataset | Multi. | Hate speech detection on a multilingual Twitter dataset | CNN and Character level representation | CNN | 0.8893 (5-language dataset) and 0.8300 (7-language dataset) |
| Zhang and Tepper (2018) | Multiling. Twitter dataset | Multi. | Hate speech detection on a multilingual Twitter dataset | Convolutional and Gated Recurrent Networks (CNN-GRU) And Dropout, pooling layers, and elastic net regularization | Convolutional and Gated Recurrent Networks (CNN-GRU) | 0.912 (7-language dataset) |
| Kumari (2020) | offensive tweets | Multi. | Multilingual offensive language identification using models | BERT, BiLSTM with attention mechanism (Attn-BiLSTM), ensemble models (UML_BiLSTM) | BERT, Attn-BiLSTM, UML_BiLSTM | F1-scores for each lang:<br><br>BERT:<br><br>Arabic-A: 0.8172 Danish-A: 0.7672 English-A: 0.9060 English-B 0.6711 English-C: 0.6054 Greek-A 0.8304 Turkish-A 0.7481<br><br>BiLSTM :<br><br>Arabic-A: 0.8447 Danish-A: 0.7000 English-A: 0.8970 English-B 0.6200 English-C: 0.5774 Greek-A of 0.8984 Turkish-A 0.7425 |
| Tanase (2020) | Offenseval 2020 subsets and other datasets | Multi | Offensive language detection across various languages | Transformer-based models (BERT, mBERT, RoBERTa, XLM-RoBERTa, and ALBERT) | Transformer-based models (BERT, mBERT, RoBERTa, XLM-RoBERTa, and ALBERT) | Offenseval 2020: English (91.05%), Arabic (82.19%), Danish (73.80%), Greek (81.40%), Turkish (77.89%). |

| | | | | | | |
|---|---|---|---|---|---|---|
| Huynh et al. (2019) | 20,345 for train and 5,086 for test | Vietnamese | Detecting hate speech in Vietnamese | Neural networks | BiGRU-LSTM-CNN | F1-score of 70.576% |
| This Thesis Study | Arabic: New dataset created, English: Combined SOLID and OLID English datasets. Turkish: OffensEval 2020 Turkish offensive language corpus. Balanced: Balanced subset extracted from the Turkish dataset.<br><br>Multilingual: Merged data from Arabic, English, and Turkish datasets. | Multi: Ar, Eng, Tr, Multi | Detecting Multilingual Offensive Language in Social Media Using Deep Neural Networks | Neural Networks CNN, RNN, and LSTM with Feature selection methods. Transformer-based language model RoBERTa, and new ensemble using combination of RoBERTa-CNN | CNN, RNN, LSTM, RoBERTa, RoBERTa-CNN | **English**: CNN: 0.9075, RNN: 0.8897, LSTM: 0.8913, RoBERTa: 0.9275, RoBERTa-CNN: 0.9221. **Arabic**: CNN: 0.8487, RNN: 0.8421, LSTM: 0.8537, RoBERTa: 0.9260, RoBERTa-CNN: 0.9271.<br><br>**Turkish**: CNN: 0.8373, RNN:0.8093, LSTM: 0.8400, RoBERTa: 0.8668, RoBERTa-CNN: 0.8679.<br><br>**Balanced Tr**: CNN: 0.7016, RNN: 0.5355, LSTM: 0.6669, RoBERTa: 0.7670, RoBERTa-CNN: 0.7539.<br><br>**Multilingual**: CNN: 0.8246, RNN: 0.7962, LSTM: 0.8177, RoBERTa: 0.8486, RoBERTa-CNN: 0.8473. |