Ministry of Higher Education and Scientific Research
Carthage University - Higher School of Statistics and Information Analysis

End of studies Project Report submitted in partial fulfillment of the requirements for the degree of

National Engineering in Statistics and Data Analysis

Submitted by:

Imen Bouzidi

Depression Detection Via Users' Behavior on Social Networks Using Deep Learning

Defended on July 20, 2020 in front of the committee composed of:

Mr. Hichem Rammeh	Jury President
Mrs. Ines Abdeljaoued Tej	Reviewer
Mr. Mokhtar Kouki	Supervisor
Mr. Sayf Chagtmi	Mentor

The graduation project was made at:

datametrix
turning data into information.

Dedication

I wholeheartedly dedicate this work to my parents, who always supported me and always gave me strength and inspiration.
To my brothers, who continually provide their moral, spiritual and emotional support.
To my friends, who always were there to advise me, to listen to me and to encourage me.
To everyone who believed in me.

Acknowledgements

First, all praises and thanks are due to the Almighty Allah for guiding me to the right path, for giving me the strength, knowledge, ability and opportunity to undertake this work and to persevere and to complete it satisfactorily.

Foremost, I would like to express my sincere gratitude to my supervisor Mr. Sayf Chagtmi for his continuous support, for his patience, motivation and enthusiasm. His guidance helped me along the project.

Then I would, especially, like to express my sincere gratitude to my university supervisor Mr. Mokhtar Kouki for directing this work, for supporting me throughout the project as well as for his valuable advice.

I would also like to express my appreciation for the presence of Mrs. Ines Abdeljaoued Tej as reviewer and Mr. Hichem Rammeh as president to evaluate my work.

Imen Bouzidi

Abstract

Depressed people do not consult doctors at early stages, which often leads to a serious deterioration of their conditions that could reach suicide. Meanwhile, they are increasingly using social media to disclose their emotions and therefore generating valuable data that could be leveraged for the early detection of depression. We started by collecting visual and textual data from various social media platforms (Twitter, Pexels and Unsplash) using keywords extracted from the Diagnostic and Statistical Manual of Mental Disorders (DSM-5). Then, we trained multiple state-of-the-art models. For text classification and after preprocessing, we trained: default embedding + LSTM and Pre-Trained Global Vectors (GloVe) + BiLSTM and for image classification we trained Deep CNN, ResNets and BiT-L (Big Transfer Large). We also proposed aggregating these two models to create a multi-modal model. 300K tweets were collected from 5460 users (223014 of depressed class and 187360 of not-depressed class). Moreover, we collected 11,484 images from Unsplash and Pexels (6250 images labeled as Depressed and 5234 images labeled as Not-Depressed). After training the models, The BiT-L (ResNet101x1) classified images with an accuracy of 0.82, F1-score for depressed class equal to 0.81 and equal to 0.82 for non-depressed class. As for texts, GloVe+BiLSTM scored an accuracy of 0.69 and F1-score for depressed class equal to 0.65 and equal to 0.73 for not-depressed class. Eventually, we aggregated the two models into one model that accepts multi-modal input and we detected an amelioration of the classification, which is promising for future research on this topic. This study demonstrates the feasibility of leveraging social media data to make evidence-based mental health services more widely available to those in need.

Keywords— Mental health, Depression, Social networks, Deep learning, Multi-modal learning

Contents

List of Figures

List of Tables

Acronyms

- **DL**: Deep Learning
- **DNN**: Deep Neural Networks
- **CNN**: Convolutional Neural Networks
- **RNN**: Reccurent Neural Networks
- **ANN**: Artificial Neural Networks
- **NLP**: Natural Language Processing
- **LSTM**: Long Short-Term Memory
- **Bi-LSTM** : Bidirectional Long Short-Term Memory
- **ReLU** : Rectified Linear Unit
- **ResNets** : Residual Neural Networks
- **GloVe** : Global Vectors
- **1D** : One Dimensional
- **3D** : Three Dimensional
- **GloVe** : Global Vectors
- **BiT** : Big Transfer
- **BiT-L** : Big Transfer Large
- **GCP** : Google Cloud Platform
- **VM** : Virtual Machine
- **CPU** : Central Processing Unit
- **GPU** : Graphics Processing Unit
- **IDE** : Integrated Development Environment
- **API** : Application Program Interface
- **Lr** : Learning rate

Introduction

In recent years, social media became a host for nearly 3.5 billion people [1] who are increasingly relying on these platforms to communicate their emotions, concerns and even daily life routine. Indeed, valuable insights could be extracted from online behavior to understand and predict offline behavior. Therefore, social media was widely used by both mental health researchers and data scientists to understand the mental state of users which is reflected by the growing amount of data shared on social networks. The aim of this study is to help flatten the alarming suicide rate. In fact, detecting depressive disorder via social networks(Twitter) is proven effective[2]. However, the greatest challenge of online depression detection lies in the various signs of depression which can always variate from one person to another.

In this work, we systematically propose a study that focuses on using both texts and images to detect signs of depression through social networks. Only a few researches were made concerning this topic. As a result, in this project we will propose our method of aggregating two models. Indeed, the approach is that we will train a model on image classification for depressed users and another one on text classification and once we have good accuracy for both models, we will try combining their predictions when having a multi-modal input. In this perspective, we will build two robust models one for image classification and another one for text classification and aggregate them into one model.

The first chapter of this report introduces the hosting company, the project context and the actual situation of depression detection. In the second chapter, we will explore the theoretical and practical basis of Deep Neural Networks presenting Convolutional Neural Networks and Recurrent Neural Networks with a brief explanation of Word Embedding and text preprocessing techniques. And in the third chapter, we will look through the design of the solution from data collection to model training details.

Finally, we will present in chapter four the practical steps of data preparation and the experimentation of the chosen models as well as the aggregation of the best models and its results.

Chapter 1

General Study Context

This chapter introduces the host company where this internship was conducted and the context of the graduation project. It also explains the problematic of the project and states some studies on the current situation of detecting depression via social networks using deep learning techniques.

1.1 The Host Company: Datametrix

This internship was carried out at Datametrix Tunisia. Datametrix is a Swiss company founded in 2003 specialized in data management and bio-statistical analysis with subsidiaries in many other countries around the world. With more than 17 years of experience, Datametrix values professionalism, integrity and innovation as key elements for their success. It relies on a motivated and reliable team of 10 data scientists to timely deliver high quality services for their clients.

Moreover, Datametrix does not limit the tools used by its team. Team members can juggle between SAS, R and Python to meet their client's needs. Basically, variety is what makes working with this team eye-opening and enriching. Its mission is to deliver high quality and cost-effective insights from raw data.

Figure 1.1: Datametrix's Logo

1.2 Project's Context

Traditional methods of depression detection rely on self-reported cases that obviously lack important data that could boost the diagnosis [3]. Since Datametrix has multiple clients in the bio-statistical field, a study that combines behavioral psychology, data and target engagement will surely help in enhancing the quality of human's life by improving their mental health [4] and also will prove profitable for marketing purposes. Automating depression detection will help identify depressed and most importantly at-risk of suicide individuals and this may complement existing screening procedures in the future.

Potential clients, such as Marketing agencies, pharmaceutical companies, national health institutes and even psychologists, will certainly be interested in identifying depressed users through harvesting their social media data that are gradually increasing.

1.3 Project's problematic

Depression is one of the major mental health issues, having a destructive impact on mind's stability and life style with its diverse symptoms that may lead to suicide. Moreover, social media networks are growing to be one of the ways prospective depressed users are expressing themselves, oversharing every detail online. Therefore, inspired by the effectiveness of early depression detection via social networks[5][6], we decided to use social media data to build a model that detects early signs of depression.

In fact, textual and visual posts shared on different social network platforms such as Twitter and Facebook create new opportunities for a better understanding of linguistic and visual expression of depressed users. Consequently, in this study we will address the following questions:

1. How well does the content (components) of a posted image reflect depressive symptoms?
2. Do textual posts contain psycho-linguistic patterns that determine depressive behavior?
3. Are there contributions from a model integrating predictions from components of both visual and textual data?

To answer these questions, the main objectives and contributions of the study include:

- Build and train a powerful model to classify posted images based on an analysis of their components and their association with depressive symptoms.

- Build and train a powerful model for text classification of potentially depressive users.
- Test the performance of the two models separately and check the contribution of the aggregation of both models to assist a multi-modal detection of depression.

1.4 Current situation of depression detection via social networks

In recent years, depression detection became an interest to scientists due to the number of suicide cases that is increasing in an alarming way. Some efforts have been dedicated to study depression on social media, researchers used traditional methods (e.g. filling questionnaires) that are effective but expensive, time-consuming and based on insufficient data. Moreover, most of the studies, conducted to detect depression signs on social media using machine learning, were limited to using only one modality (visual data or textual data) [7] [8]. For the first time in 2017, in this article [9], researchers managed to propose a multi-modal depressive dictionary learning model, that aims for the detection of depression based on a predefined list of symptoms.

Conclusion

In this chapter we introduced the general context and the problematic of this project. In the following chapter we will present the theoretical tools that made working on this project feasible, we will present an easy introduction to the models we used that will facilitate understanding the design of our proposed solution.

Chapter 2

Background

The challenge of classifying users presenting signs of depression and others who do not based on multi-modal data, requires extracting a massive amount of features from raw unstructured data, principally images and texts. This leads us automatically to Deep Learning (DL) thanks to its ability to learn and perform increasingly complex tasks.

In this chapter, we will introduce the theory behind the models used to accomplish this study, starting with an introduction to the basics of deep learning. Then we will present the Convolutional Neural Network (CNN) used in image classification of users presenting signs of depression and others who do not. Thereafter, for the text classification task, we will be explaining the Recurrent Neural Network (RNN) going through some details about Long Short-Term Memory (LSTM) and Bi-directional Long Short-Term Memory (Bi-LSTM) networks, with a glimpse on both word-embedding and text pre-processing techniques. Finally, we will present the convenient classification metrics to evaluate images and text classification models.

2.1 Deep Learning (DL)

DL represents a sub-field of machine learning relying on an algorithm similar both in structure and function to the human brain called artificial neural networks. It stimulates the functions of human neurons in transmitting and processing data to turn it into information.

2.1.1 Basics of DL

2.1.1.1 The Perceptron

The perceptron is a single layer neural network model designed for supervised binary classification invented in 1957 by Frank Rosenblatt [10]. The hidden layer of this model is composed of one neuron (computational unit), therefore, its simplicity will help understand the basics of the neural network functionality. The structure of perceptron in forward propagation is presented in figure 2.1 .

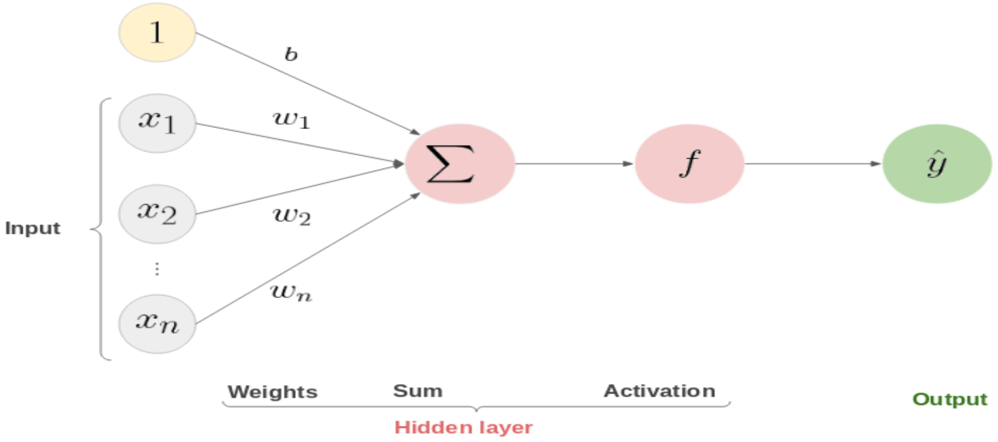


Figure 2.1: The Perceptron: forward propagation

The predicted output $\hat{y}$ from the vector of input is:

$$\hat{y} = f(b + \sum_{1 \leq i \leq n} x_i w_i) \tag{2.1}$$

Where:

- $(x_1, ..., x_n)$: the input vector
- $(w_1, ..., w_n)$: the weights vector
- f: the **activation function**
- b: the bias.

In the **forward propagation**, once we randomly initialize the weights' vector $(w_1, ..., w_n)$, the input layer takes a numerical representation of data $(x_1, ..., x_n)$ (e.g. pixels of an image, sequences of a text) then $\sum_{0 \leq i \leq n} x_i w_i$ is calculated which is the weighted sum of the input values.
An activation function f is eventually applied on this weighted sum to get the predicted output $\hat{y}$.

To improve the prediction, a loss function is calculated and optimized during the learning process of **back-propagation** that includes updating the weights on the basis of minimizing this function.

In order to summarize, we present a simplified diagram 2.2 explaining the learning process of a neural network.
In the diagram, $\ell(y, \hat{y})$ represents the loss expression for the predicted value $\hat{y}$.

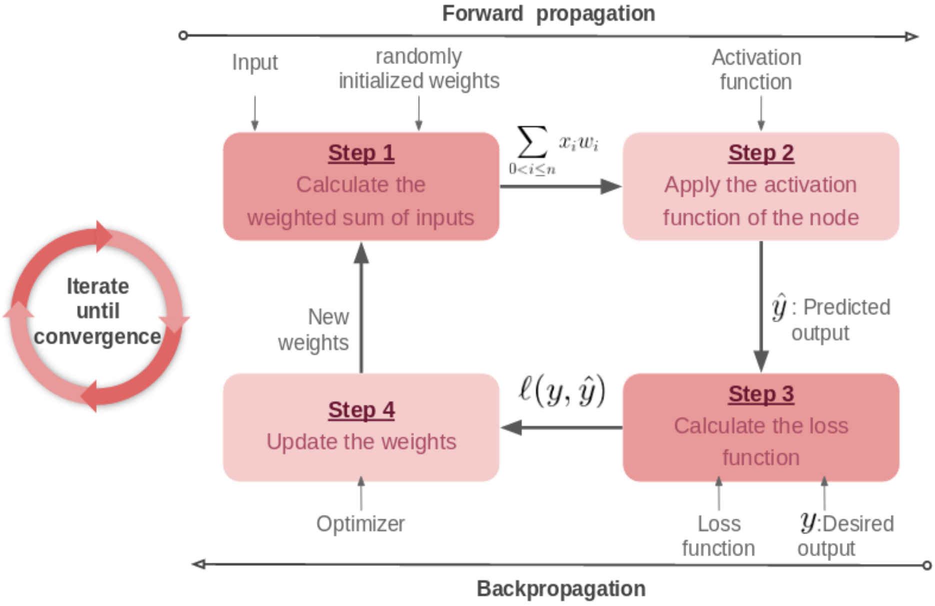

Forward propagation
Input
randomly initialized weights
Activation function
Step 1
Calculate the weighted sum of inputs
$\sum_{0<i\leq n} x_i w_i$
Step 2
Apply the activation function of the node
Iterate until convergence
New weights
$\hat{y}$: Predicted output
$\ell(y, \hat{y})$
Step 4
Update the weights
Step 3
Calculate the loss function
Optimizer
Loss function
y:Desired output
Backpropagation

Figure 2.2: Summary of a neural network process

As a conclusion, a neural network is a composition of multiple perceptrons connected and operating on different activation functions. The number of hidden layers indicates the depth of the network. Therefore, a deep neural network is composed of more than 4 layers including the input and output layers (or simply more than one hidden layer). 2.3

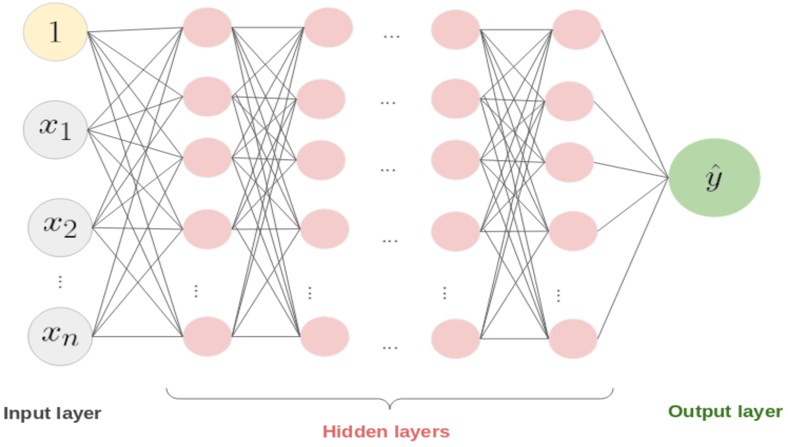


Figure 2.3: Deep neural network architecture

2.1.1.2 The activation function

It is generally a non-linear mathematical function that helps the neural network learn complex inputs, detect non-linear effects, and provide better predictions for multiple types of representations. The following are the frequently used activation functions that have particular mathematical properties such as monotony, differentiability, and convexity:

- **Rectified Linear Unit (ReLU)**: is the most used activation function in neural networks as it allows the network to converge very quickly.

  Mathematically, it is defined as:

$$\mathbb{R} \longrightarrow [0, -\infty[$$

$$f : x \mapsto max(0, x) \tag{2.2}$$

- **Sigmoid or Logistic activation function**: is generally used for binary classification problems as it outputs a value between 0 and 1. It is defined as:

$$\mathbb{R} \longrightarrow]0, 1[$$

$$f : x \mapsto \frac{1}{1 + e^{-x}} \tag{2.3}$$

- **Hyperbolic tangent (tanh)**: have almost the same properties as a sigmoid function. It outputs values that range between -1 and 1. Its mathematical expression is:

$$\mathbb{R} \longrightarrow] - 1, 1[$$

$$f : x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$

- **Softmax**: is a form of logistic regression which turns numbers into probabilities that sum to 1. Its output is a probability vector to predict the belonging to each target class. Mathematically, let N be the number of target classes to be predicted, the softmax function is defined as:

$$\mathbb{R} \longrightarrow]0, 1[$$

$$f_i : x_i \mapsto \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \quad for \;\; i = 1, ..., N \tag{2.5}$$

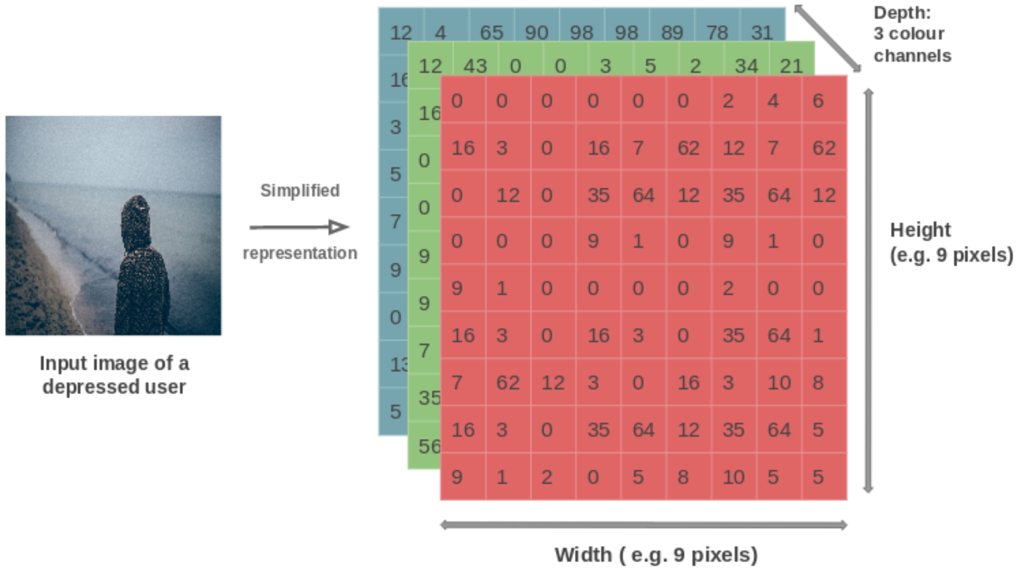
2.1.2 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN or ConvNet) is a class of DL, inspired by the human vision system [11] [12] [13] and specialized in extracting high-level features from complex inputs such as images, texts, and audio. In fact, unlike the classical neural network where features are manually engineered, CNN takes the inputs' raw data, automatically distinguishes meaningful patterns and learn based on those features instead of raw pixels. This is what makes CNN the frequently used neural network to analyze visual imagery [14].

2.1.2.1 CNN Architecture

Input Layer:

Unlike neural networks where the input is a vector of pixels, in CNN, an image is usually represented as a three dimensional (3D) matrices of pixels' values. Height and width are according to the input image dimensions, and depth is generally three channels RGB (Red-Green-Blue).

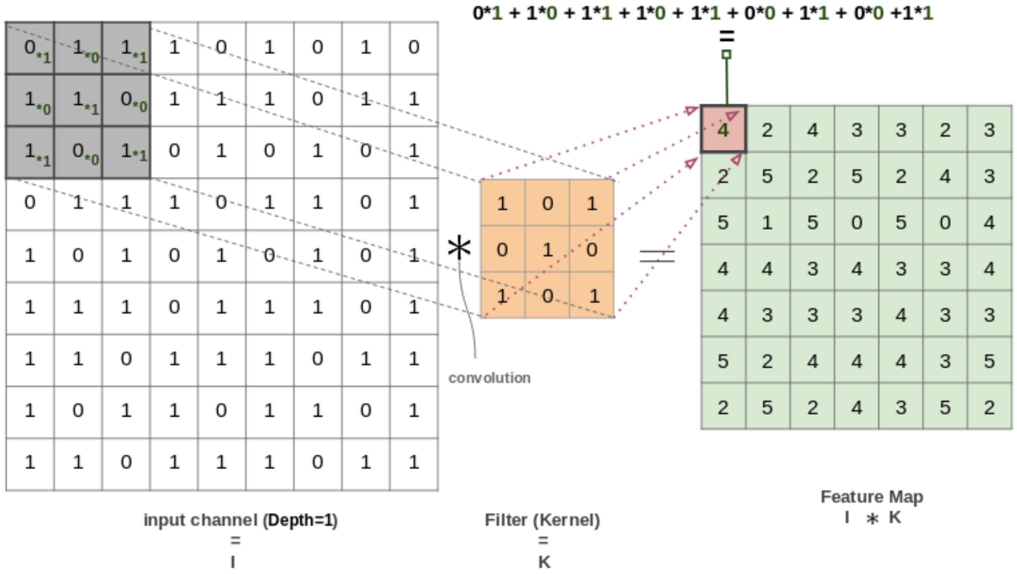


0	0	0	0	0	0	2	4	6
16	3	0	16	7	62	12	7	62
0	12	0	35	64	12	35	64	12
0	0	0	9	1	0	9	1	0
9	1	0	0	0	0	2	0	0
16	3	0	16	3	0	35	64	1
7	62	12	3	0	16	3	10	8
16	3	0	35	64	12	35	64	5
9	1	2	0	5	8	10	5	5

Figure 2.4: Example of an RGB representation of an image

Convolutional Layer:

The convolution operation is the basic component of CNN, invented by Yann LeCun[15], it consists in applying a filter on the input image to detect features related to each class.



0_{*1}	1_{*0}	1_{*1}	1	0	1	0	1	0
1_{*0}	1_{*1}	0_{*0}	1	1	1	0	1	1
1_{*1}	0_{*0}	1_{*1}	0	1	0	1	0	1
0	1	1	1	0	1	1	0	1
1	0	1	0	1	0	1	0	1
1	1	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1	1
1	0	1	1	0	1	1	0	1
1	1	0	1	1	1	0	1	1

1	0	1
0	1	0
1	0	1

4	2	4	3	3	2	3
2	5	2	5	2	4	3
5	1	5	0	5	0	4
4	4	3	4	3	3	4
4	3	3	3	4	3	3
5	2	4	4	4	3	5
2	5	2	4	3	5	2

Figure 2.5: Convolution Operation: red channel of 9 pixels image

The filter (kernel of a fixed **size** [1]) is composed of randomly initialized weights that will be updated through the back-propagation with each input. It slides through the image with a **stride** [2] (one pixel in the example of fig 2.5) multiplying the pixels' values of the images by their values. All these multiplications are summed in a number composing a pixel of the feature map.

At the end of the convolution process, the feature map obtained represents a smaller matrix containing the detected patterns of the input image.
Obviously, the more filters we apply on the image, the more features are extracted and the better the network becomes at detecting patterns in images.

To summarize, the convolution layer is composed of multiple filters of the same size that outputs various feature maps from an input of a 3D channel representation of an image 2.6.

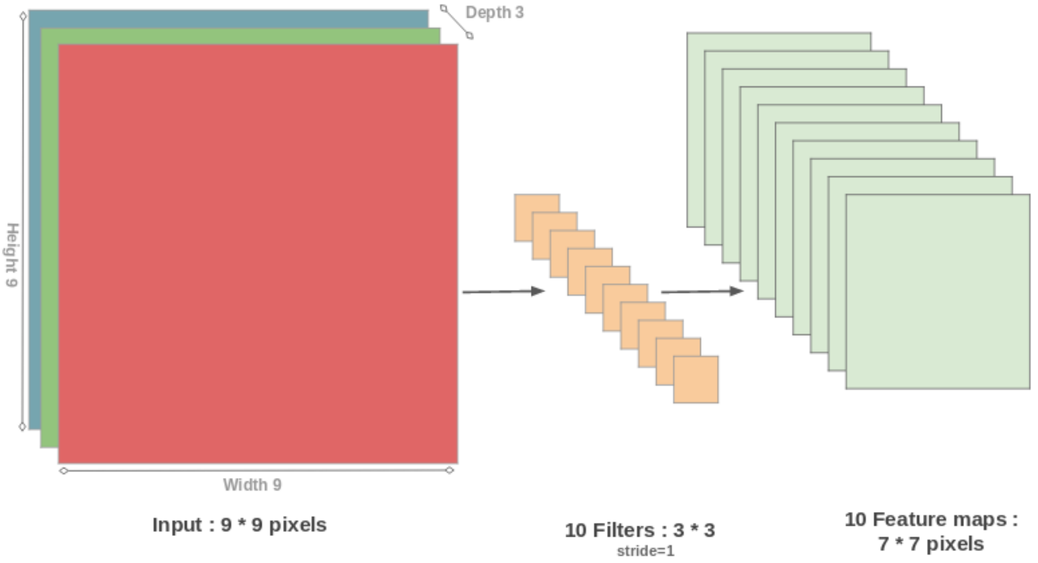


Figure 2.6: Convolutional Layer of an input image of 9*9 pixels with stride 1 and size of filter 3*3

Non-Linearity with ReLU:
Considering the non-linearity of real-world data, we must introduce a layer of activation function after each of the convolution layers, since convolution is a linear operation (multiplication and addition). Indeed by increasing the non-linearity of the network, we create a complex network that enables us to detect and distinguish numerous patterns of the input image. The most commonly used activation function for this task is ReLU [16]. Thanks to its mathematical properties, it is an element-wise operation (applied per pixel in our case) that replaces the negative pixels' values with zeros in the feature map.

[1] It is the size of the matrix considered a hyper parameter of the convolution operation.
[2] Stride is the number of pixels skipped in the convolution, it is considered a hyper parameter of the convolution operation.

Pooling Layer:

Similar to the convolutional layer, the pooling layer is used to further reduce the dimension of the previous matrices independently (dimensional reduction) to significantly decrease the amount of parameters and the computational power required to the data processing.

Furthermore, this operation extracts dominant features from the input while maintaining the process of effectively training the model as the network becomes invariant to small transformations and translations of the input image.

Pooling can be accomplished in multiple approaches: Max pooling, Min pooling, Mean pooling and Average pooling. The most common approach used is Max pooling. The process of applying Max pooling is presented in figure 2.7.

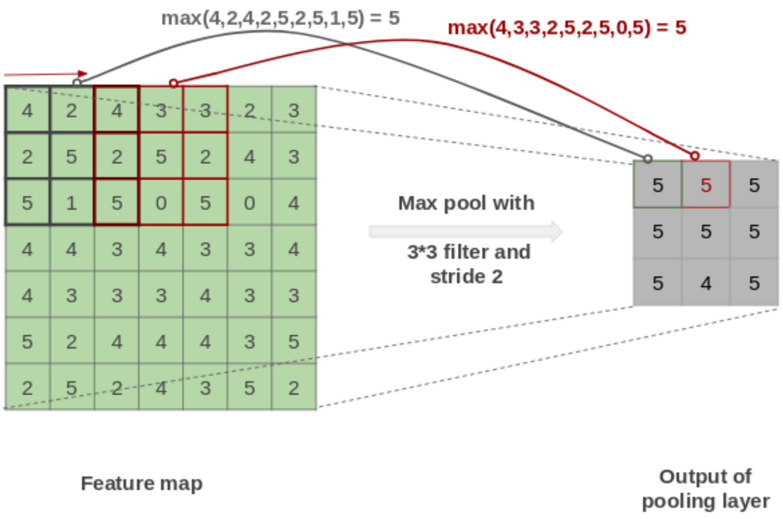


Figure 2.7: Pooling Layer of an input image of 7*7 pixels with stride 2 and size 3*3

Flatten and Fully-connected Layer:

The combination of the convolutional layer and pooling layer extracts dominant features from the input image into a number of matrices, then flatten converts it to a 1-dimensional array creating a single long feature vector suitable to be the input of the fully-connected Layer.

The fully-connected layer accomplishes the task of classification, it represents a Multi-layer Perceptron with mostly soft-max activation function. The network will then be able to learn non-linear combinations of the high-level features and distinguish each image specificity. 2.8

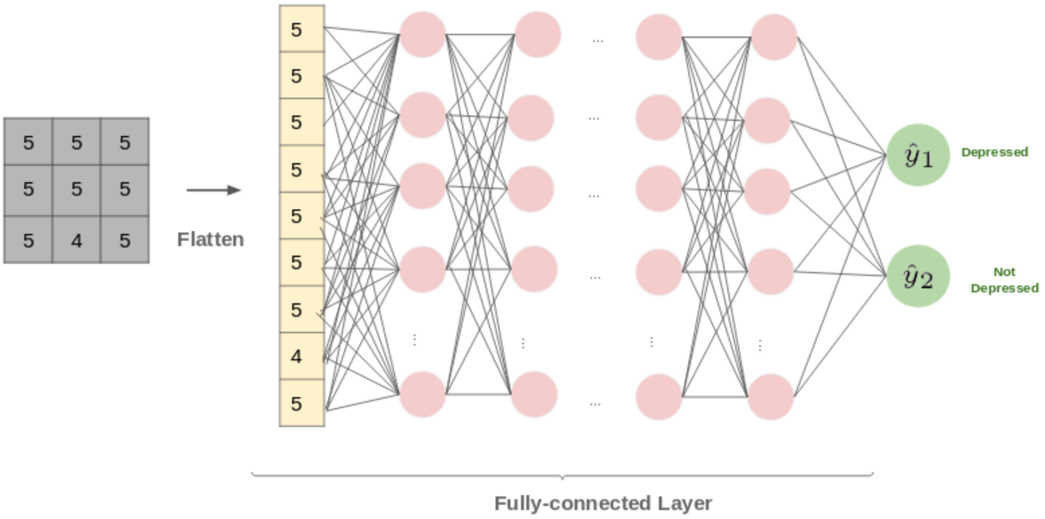


Figure 2.8: Flatten and Fully-connected Layer

To summarize, the CNN architecture performs two major tasks:

- Feature extraction: convolutional layers + pooling layers
- Classification: fully-connected layer

The figure 2.9 represent the full architecture of a CNN.

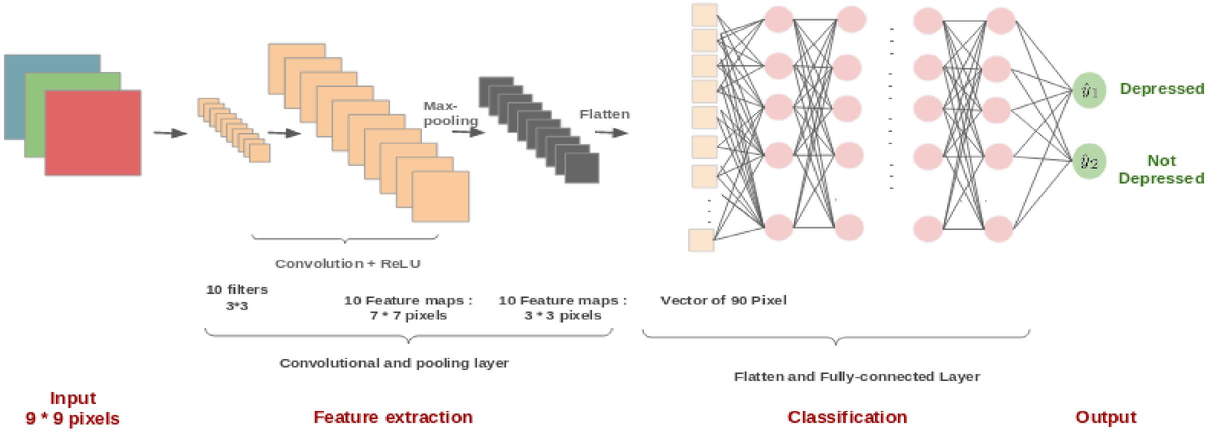


Figure 2.9: Summary of the CNN architecture for the example of an input of 9*9 pixels

In general the more convolutional layers we have, the more features the model will be able to recognize.

2.1.3 Recurrent Neural Network (RNN)

As CNN, a recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed sequential graph [17]. In fact, its idea is to make use of sequential data (e.g. Words, Sequence, Time series ...) and the dependence between the predicted outputs of the previous computations and the actual predicted output. In simple words, if you want to predict the next word in a sentence you better know which words came before it and here comes the role of RNN.

The figure 2.10 represents the architecture of an RNN.

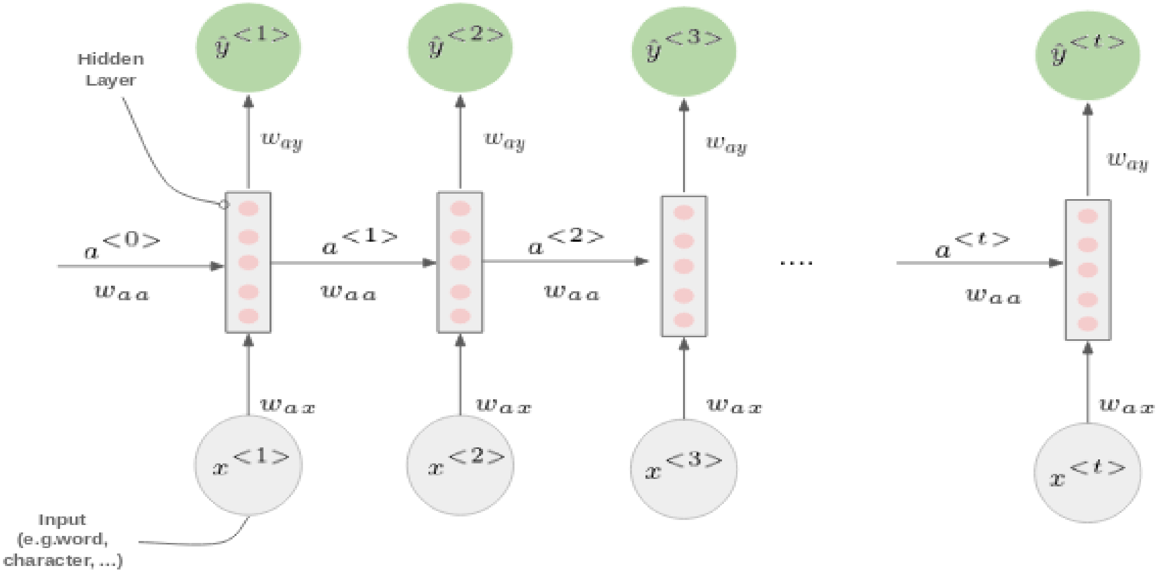


Figure 2.10: Summary of the RNN architecture

A sequential input is partitioned into minor inputs: for example a sentence will be partitioned into the composing words, these inputs are expressed in vectors using various methods such as word representation (One-hot Encoding, Bag-of-words) and word embedding 2.3.

After the process of RNN in figure 2.10, the predicted output will then be:

$$y^{<t>} = g_2(w_{aa} * a^{<t>} + by) \tag{2.6}$$

Where w_{aa}, w_{ax} and w_{ay} represent the parameters of the RNN model and:
$a^{<t>} = g_1(w_{aa} * a^{<t-1>} + w_{ax} * x^{<t>} + ba)$
g_1 is generally a ReLU 2.2 or a tanh 2.4 and g_2 is a Soft-max 2.5 or a Sigmoid (for binary classification) 2.3.

As you can see in the figure 2.10, the first hidden layer takes the $x^{<t>}$ from the sequence of input and then it outputs $a^{<t>}$ which together with $x^{<t+1>}$ is the input for the next step. This way, it keeps remembering the context while training.

As cited in [18], RNN suffers from gradient vanishing and exploding problems for complicated tasks when more context needs to be taken in consideration. As a result, Long Short-Term Memory Networks (LSTM) were designed to solve this problem[19], the following part will explain LSTM.

2.1.3.1 Long Short-Term Memory Networks (LSTM)

Long Short-Term Memory (LSTM) networks are a rectified version of RNN, which makes the task of remembering past computations easier in memory. This resolves the vanishing gradient problem [20] [21]. Therefore, LSTM is a very useful network for classifying texts.

The LSTM architecture is very similar to RNN, though LSTM architecture is composed of more inside blocks with two inputs and two outputs to keep track of the long and short-term memories. To make it easier to comprehend, the following figure represents the gates of the architecture.

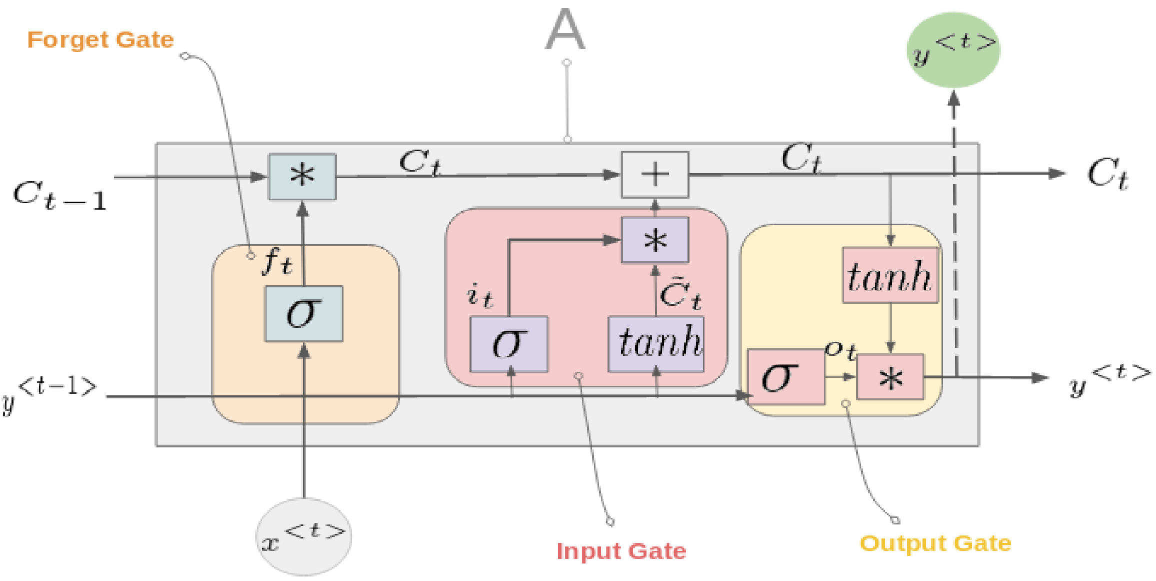


Figure 2.11: The repeating module in LSTM with its gates.

The LSTM network is composed of three gates:

- **Forget Gate:**
  This gate is composed of a sigmoid 2.3 layer that decides which information to forget. It takes the previous output $y^{<t-1>}$ and the current input $x^{<t>}$ time step t and outputs a number between 0 and 1 following the function:

  $$f_t = \sigma(W_t * [y^{<t-1>}, x^{<t>}] + bf)$$

  If f_t gives 0 as output, the information is considered as useless and will be completely forgotten. The vector that the sigmoid function outputs is multiplied to the previous cell state C_{t-1} which gives as an initial value of

the cell stat C_t.
N.b: W_t is the weight corresponding to each input and b_t is the bias.

- **Input Gate:**
  It is a two layer gate that decides which value from the input should be added to the current state.
  On one hand, the Sigmoid layer applies the function
  $$i_t = \sigma(W_i * [y^{<t-1>}, x^{<t>})] + bi)$$
  on the previous output $y^{<t-1>}$ and the current input $x^{<t>}$ to get a value of 0 or 1 deciding which value to remember.
  On the other hand, the tanh function is applied also on the same input and output to decide the level of importance of the information. The function applied is: $\tilde{C_t} = tanh(W_c * [y^{<t-1>}, x^{<t>}] + bc)$.
  The new cell state is updated with the amount of information to remember from the previous two gates to be:
  $C_t = f_t * C_{t-1} + i_t * \tilde{C_t}$

  N.b: W_i and W_c are the weights corresponding to each input of each layer while b_i and b_c are the bias.

- **Output Gate:**
  This is the final gate that decides the part of the current cell state that makes it to the output. Sigmoid function decides which values to let through applying: $o_t = \sigma(W_o * [y^{<t-1>}, x^{<t>}] + bo)$ just like the forget gate and then the tanh is applied on the current cell state C_t that is updated through the gates. So the final output of the node is:
  $y^{<t>} = o_t + tanh(C_t)$

This explication was inspired from the Colah's Blog.[22]

2.1.3.2 Bidirectional Long Short-Term Memory Networks (Bi-LSTM)

The LSTM network takes in consideration the current input and only the previous output of each node. Therefore, LSTM network takes more time to compile and understand the context of the whole sentence. Take for example in NLP, to understand a word we do not just need the previous word, but also to the next word. To solve this problem, Bi-LSTM networks was designed inspired by Bidirectional Recurrent Neural Networks (BRNN) [23].

It is composed of multiple nodes just like LSTM but it applies forward-propagation two times, one for the forward cells (like LSTM) and one for the backward cells (taking the next word in consideration). To sum up, The Bi-LSTM output takes the input of the previous cell, the current cell and the next cell. This is well-explained by the figure 2.12.

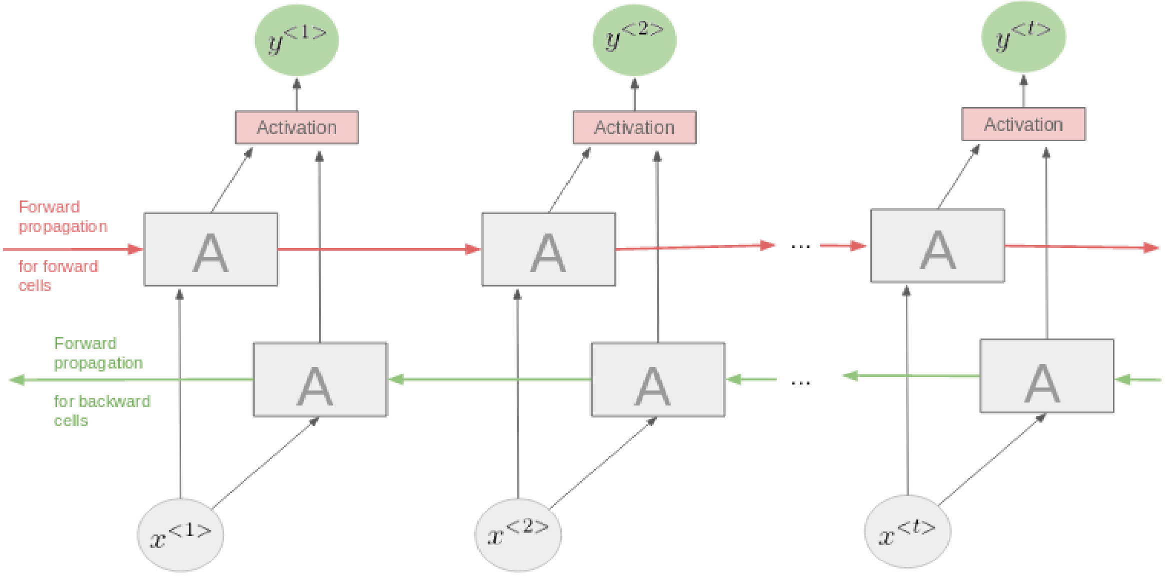


Figure 2.12: Bi-LSTM Architecture

As we can notice from the figure 2.12, Bi-LSTM network trains two LSTMs on the input sequence, the first LSTM on the sequence as it is and the other on a reversed version of the sequence.

2.2 Text Pre-processing techniques

Natural Language Processing (NLP) always requires preprocessing textual data, since in the same language we can notice major differences between written words. Take for example, the word "Like" could be written in multiple ways: ’Like’, ’Likes’, ’Liked’, ’Liiike’, ’Likeeeeeee’,...

A computer would not be able to distinguish that this is the same word. To make the task easier, we need the following steps to transfer text from human language to machine-friendly format for further processing to make text classification easier.

Some of the most used pre-processing techniques are:

- **Text Normalization:** is converting texts into a single canonical (standard) form. It includes:
  - Transforming letters to lower or upper case.
  - Removing numbers or converting it to letters.
  - Removing white spaces and punctuation.
- **Stop words removal:** as its name indicates, it consists in removing stop words that are useless for achieving the task such as: ’a’, ’and’, ’above’, ’with’, ’than’, ’about’, ...

In fact, this may seem very obvious and easy but removing certain words can affect the processing, so stop words differ with the context of the study. [24] [25]

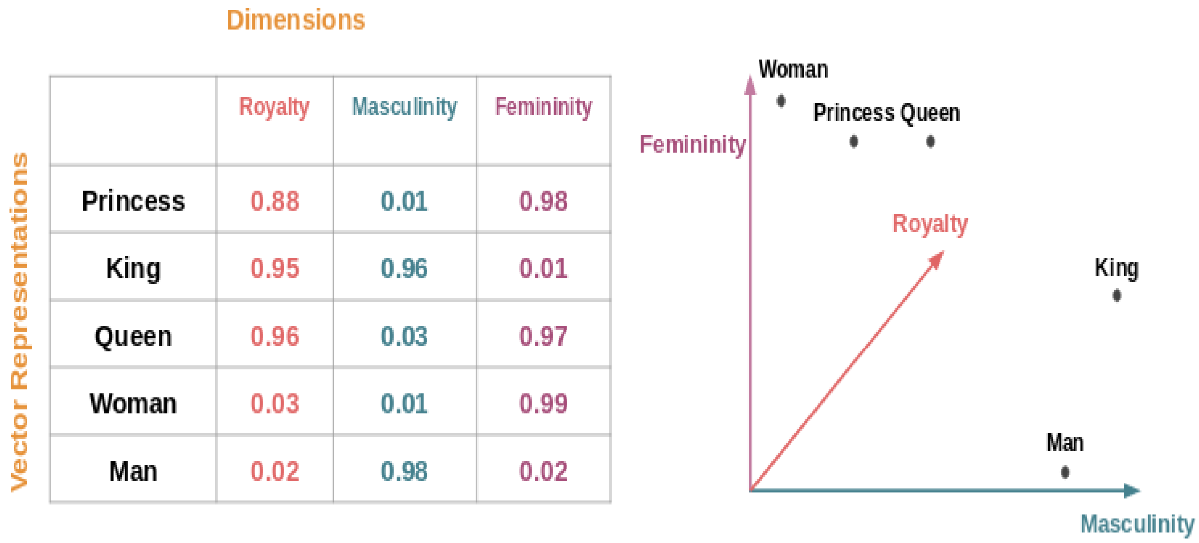
- **Stemming:** is a process of reducing words of the same root to a common form (e.g. 'Likes' and 'Liked' will be reduced to 'Like'). The main algorithms used for stemming are:
    - Porter Stemming Algorithm [26] removes common endings from words.
    - Lancaster Stemming Algorithm [27] is the most aggressive stemming algorithm, it can transform words into strange stems that have no meaning (e.g. 'having' becomes 'Hav')
- **Lemmatization:** is also a process of reducing the words having the same root into a common word. Unlike stemming, lemmatization does not simply slices of endings but uses a lexical base to get the correct form of the common word. For example: meeting becomes meet, played becomes play...
- **Text Tokenization:** consists in splitting the text after pre-processing it into smaller pieces called tokens, this can be done on character level (split words into its composing characters) or word level (split sentences into its composing words).

2.3 Word Embedding

Sequential data are represented using various methods to be a computer-friendly input, some traditional approaches such as one-hot encoding and bag-of-words are not suitable for classifying texts with a sentiments analysis (Detecting Sentiments through texts) as the dummy variables are unable to capture the word's context and meaning (semantic) by denying any relationships between words.

Word embedding, unlike the traditional approaches, represents words with real-valued vectors obtained from unlabeled large corpus. It is a very powerful tool used in modern NLP tasks including semantic analysis [28] which is the task that interests us. Each word is represented with a high quality vector disposing weights in each dimension, these dimensions do not hold a clearly defined meanings in real-word data.

We present the following figure 2.13 to simply explain how the word embedding works.



	Royalty	Masculinity	Femininity
Princess	0.88	0.01	0.98
King	0.95	0.96	0.01
Queen	0.96	0.03	0.97
Woman	0.03	0.01	0.99
Man	0.02	0.98	0.02

Figure 2.13: Example of word representation with word embedding

For the example of the figure 2.13 we suppose that the dimensions have clear meaning and we can notice that vector representation of words sharing meaning have close values, as represented in the space on the right side. We can easily apply mathematical operations to calculate similarities between words for e.g. $w_{King}{}^{-}w_{Man} + w_{Woman} = w_{Queen}$.
There are multiple methods of word embedding such as Word2Vec, GloVe, FastText, ELMo...

2.4 Classification Performance Metrics

Evaluation of the model's performance for both text classification and image classification is a necessary part of the study.

To simplify understanding the evaluation metrics, we will start with explaining the confusion matrix and then cover some efficient metrics that evaluate a classification model for binary classification problems.

2.4.1 Confusion matrix

It is a tabular visualization of the model predictions with the actual labels. Raw's values of confusion matrix are the instances in the actual class and column's values represent the instances in the predicted class.

		Predicted labels	
		Depressed	Not Depressed
Actual labels	Depressed	TP: True Positive	FN: False Negative
	Not Depressed	FP: False Positive	TN: True Negative

Table 2.1: Confusion matrix example: binary cSlassification

As we can see in the table 2.1, diagonal values of this matrix represent the correct prediction of the classes, while the off-diagonal values represent the samples which are misclassified .

In simple words:

- **True Positives:** the model predicted "Depressed" and the actual output was also "Depressed".
- **False Positives:** the model predicted "Depressed" and the actual output was "Not Depressed".
- **True Negatives:** the model predicted "Not Depressed" and the actual output was "Not Depressed".
- **False Negatives:** the model predicted "Not Depressed" and the actual output was "Depressed".

2.4.2 Metrics

2.4.2.1 Classification Accuracy

Classification accuracy is the most commonly used metric, it is the fraction of the number of correct predictions divided by the total number of predictions. Formally, accuracy has the following definition:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

For binary classification, such as the case of our study, the fraction is defined as follows:

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

Classification accuracy is not an accurate indicator when we have a problem of imbalanced data. For example, if the model predicts all the samples of the most frequent class right, we would get a high accuracy rate, which does not make sense at all if the model misclass the samples of the other class.

2.4.2.2 Precision and Recall

Precision and Recall are both class specific performance metrics. Mathematically, they are defined for both classes as follows:

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

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

2.4.2.3 F1-Score

F1-score indicates the precision and robustness of the model, it ranges from 0 to 1, the greater it is the better is the performance of our model.

Mathematically, it is defined as the Harmonic Mean between precision and recall, therefore it gives importance to both recall and precision. Its expression is as follows:

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Conclusion

In this chapter, we presented the basics of deep neural networks alongside the explanations of CNN and RNN. We also went through texts' preprocessing techniques and word embedding to understand how sequential data is prepared for computer processing and finally we finished with the metrics that will allow us to evaluate our models.

On the grounds that we understood the theory of the models used in the study, we will present in the next chapter the proposed solution in response to resolve the problematic of the project.

Chapter 3

Design of the solution

In this chapter, we will go through the technical steps of building the models of text and image classification. First, we will start with the motivation of this study and follow with the data collection for both visual and textual data and how we managed to clean it and prepare it for the processing phase. Then we will present the architectures and explanations of the models that we worked with. Finally, we will finish with defining the hyper-parameters and detail the project workflow.

3.1 Motivation and Objective

Behavioral psychology is the science of detecting a mental condition through the behavior of the patient [29], therefore early depression detection via social network platforms drew the attention of data scientists and psychologists. With the increase of using both textual and visual posts as a way of expression, we were inspired by multiple studies conducted on depression detection that uses variables of various types (structured/unstructured)[9] [30] [31].

The purpose of this project is to build two state of the art models and train them respectively on visual and textual data, to get a deeper understanding of the user's behavior. After testing the performance of each model, we will finally try integrating the predictions of these two models to build a multi-modal neural network.

3.2 Data collection

Since data was not publicly available due to users' privacy, we decided to crawl it from various social network platforms. In the beginning of the internship, we thought of scraping data from Facebook, but due to the Facebook-Cambridge Analytica scandal, Facebook became more "protective" of users' data and forbid the automated collection of users' data. Therefore, we used Twitter to collect users' data that are publicly available.

3.2.1 Visual Data

To get a huge amount of images, we used two freely-usable images platforms Unsplash and Pexels. Influenced by major depressive disorder's symptoms present in the Diagnostic and Statistical Manual of Mental Disorders book [32], we used the following keywords to collect labeled images through these websites:

- Keywords for "Depressed=1" label: "stress", "depression", "suicide", "grief", "despair", "hopeless", "lonely", "anxiety" and "struggle".
- Keywords for "Not Depressed=0" label: "smile", "excited" and "happy".

Process of Crawling Data:
The overall process of scraping images from unsplash and pexels is presented as follows in figure 3.1.

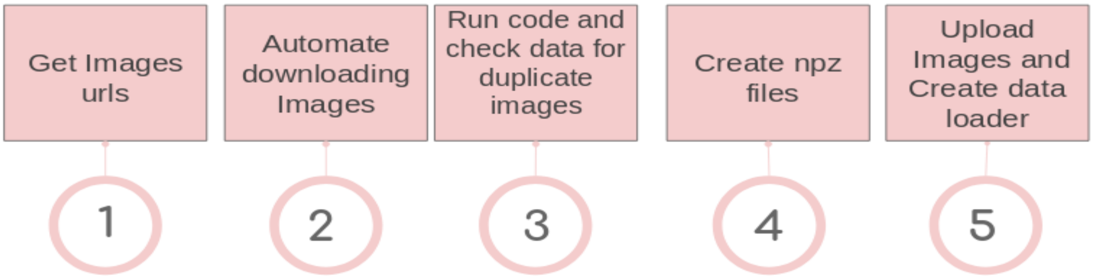

Get Images urls
Automate downloading Images
Run code and check data for duplicate images
Create npz files
Upload Images and Create data loader
1
2
3
4
5

Figure 3.1: Process of crawling images from pexels and unsplash

3.2.2 Textual Data

Since lots of studies proved the effectiveness of depression detection via twitter, we decided to scrape twitter. We crawled a data set of tweets by detecting users who self-declared having been diagnosed with depression and those who did not and we captured their profile-IDs. Then, for each user, we crawled the 100 latest tweets shared on his profile to finally create an annotated corpus with two truth labels (Depressed = 1, Not Depressed = 0).

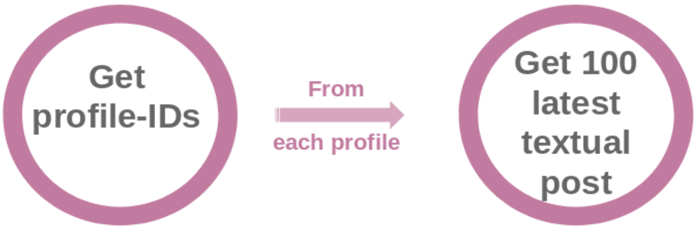


Figure 3.2: Process of crawling text from twitter

The extraction of self-declared cases was inspired from a scientific article that proves the efficiency of this method [8]. The keywords used for depressed users research were: "I am diagnosed with depression", "I am fighting depression" and "I suffer from depression". To extract "not depressed" users, we thought of an approach that would get profiles of users that are definitely not depressed. This approach consist on researching trending happiness hashtag [1] such as: "#happyme", "#lifeisgood" and "#lovemylife".

3.3 Models

3.3.1 Images Classification

To accurately classify images that depressed users are likely to post, we trained multiple models having different architectures based on CNN previously explained in chapter 2 2.1.2 to obtain the best performance. We started with the simplest architecture and finished with a state of the art model:

3.3.1.1 Deep CNN:

This model is composed of:

- 4 Convolutional layers: with a kernel (filter) of size 3 to extract a higher level of features for both depressed and not depressed users.
- 2 Max-pooling layers: with a kernel (filter) of size 2 to decrease dimensionality.
- 3 Dropout layers : dropout is one of the most used techniques to prevent over-fitting. [2]. "Dropout" refers to temporarily removing a randomly chosen unit (a neuron) from the network, along with all its incoming and outgoing connections [33].

The architecture of this model is presented in figure 3.3.

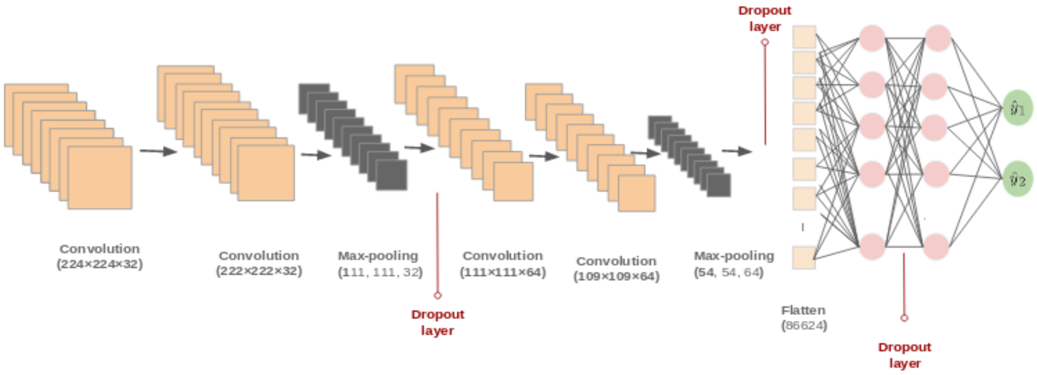


Figure 3.3: Architecture of Deep CNN model

[1] Using this website: https://hashtagify.me/hashtag/lifeisgood

[2] It is simply when the model is performing very well on the training set but poorly on the test set

3.3.1.2 Residual Neural Networks (ResNets):

ResNets learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. They take a standard feed-forward CNN and add skip connections that bypass a few convolution layers at a time.

Each bypass gives rise to a residual block in which the convolution layers predict a residual that is added to the block's input tensor. They stack residual blocks on top of each other to form a network: e.g. a ResNet-50 has fifty layers using these blocks.

Typical ResNet models are implemented with double layer skips that contain non linearities (ReLU) and batch normalization [3] in between.

Formally, denoting the desired underlying mapping as $H(x)$, we let the stacked nonlinear layers fit another mapping of $F(x) = H(x) - x$. The original mapping is recast into $F(x) + x$.

The building block of a ResNet is presented in figure 3.4:

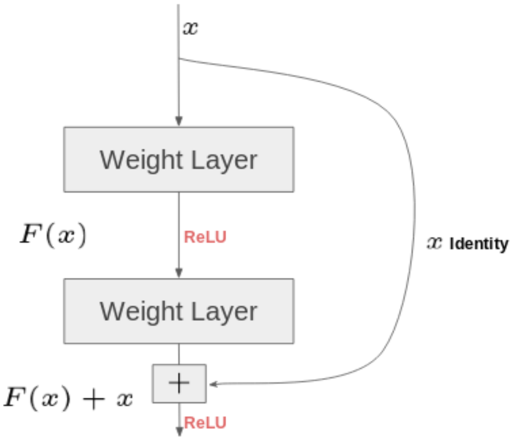


Figure 3.4: A building block of ResNets

There is empirical evidence that these types of network are easier to optimize, and can gain accuracy from considerably increased depth [34].

In our study we used two architectures of ResNets depending on the number of layers which are ResNet50 (50 layer) and ResNet101 (101 layer).

Transfer Learning:

Since training is very expensive, both in time and resources, we chose to work with a pre-trained ResNet model. Transfer of pre-trained representations improves sample efficiency and simplifies hyperparameter tuning when training deep neural networks for vision. Basically, transfer learning, as its name indicates, transfers the learning from a model trained on a specific task to a new model. In our case we chose to not fine-tune weights, so we will keep the model as it is and only train the output layer.

[3] It normalizes the input layer by adjusting and scaling the activations to allow each layer of a network to learn by itself independently of other layers.

The pre-trained model will act like a feature extractor which is previously trained on huge data-sets to become so good at this task.

Big Transfer (BiT)
To go even further and enhance the performance of the image classification model, we used BiT, a state of the art model [35] developed in 2019. Specifically, we used BiT-L (Big Transfer Large) which is trained upon the dataset JFT-300M (contains 10 to 1 billion image datasets).

We can choose the architecture of the model when importing it, we chose to try ResNet50 and ResNet101 since these two were the ones that gave the best results on other datasets.

3.3.2 Text Classification

Extracting psycho-linguistic features of depressed users is a task that requires a powerful model to establish. The challenge is that extracted texts are status that may be alike for most users belonging to different classes. For this, we tried two architectures on our extracted data labeled as "Depressed" and "Not Depressed". We started with a simple LSTM trying variant hyper-parameters and then a pre-trained Global Vectors for word embedding and BiLSTM for the model. The following sections explain the different architectures we used.

3.3.2.1 LSTM:

The used architecture of LSTM contains:

- The default embedding layer, as the first layer, that maps words onto a continuous vector space, thus creating word vectors or word embeddings with trainable parameters. The embedding layer is initialized with random weights and will learn an embedding for all of the words composing the training data-set.
- A spatial dropout 1D layer that performs the same function as dropout layer, it drops entire one dimensional (1D) feature maps instead of single neurons. This layer helps promote the independence between feature maps.
- A layer of 124 LSTM units.
- A dense layer followed with a dropout layer (for over-fitting)
- The final dense layer gives the input as two separate classes.

In table 3.5, we present the architecture of the LSTM used:

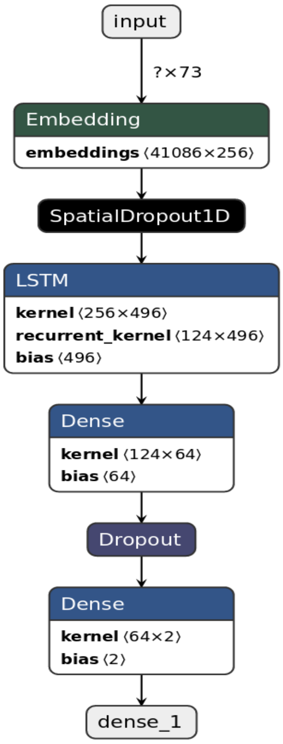


Figure 3.5: The proposed architecture of a LSTM model used.

3.3.2.2 Pre-Trained Global Vectors (GloVe) + BiLSTM:

GloVe embedding:
is an unsupervised learning algorithm developed by Stanford in 2014 [36], it is one of the most commonly used word embedding approaches. It encodes co-occurrence probability ratio between two words as vector differences to mark meaningful differences between word vectors. The goal is to define a context for two words as to whether or not the two words appear in close proximity of N-word apart.
The model architecture contains 5 separate types of layers as figure 3.6 indicates, these layers are:

- The input layer
- Embedding layer : a pre-trained GloVe word embedding.
- The Bidirectional layer : which is a layer composed of 512 BiLSTM units.
- A dropout layer to avoid over-fitting.
- The dense layer is the layer that outputs the result.

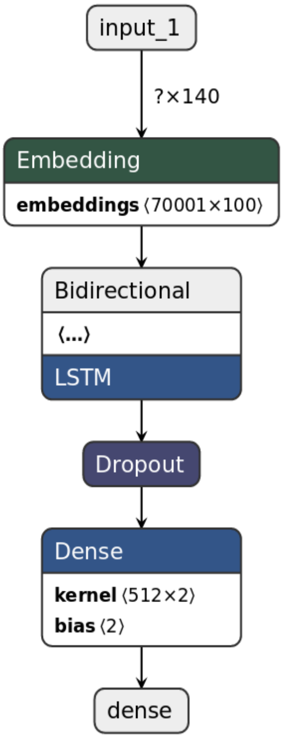


Figure 3.6: The proposed architecture of the GloVe+BiLSTM model used.

3.3.3 Models Aggregation

The approach we used for model aggregation was simple. Since we don't have enough data to grid search the coefficient of each model we made an assumption that social network users assign equal importance to expression with texts and with images. The process of the aggregation is the following:

1. Load the best saved image classification model.
2. Load the best saved text classification model.
3. Predict the output of visual input
4. Predict the output of textual input
5. calculate the predicted value for the aggregation with this formula:

$$Pred_{multimodal} = Pred_{Text} * 0.5 + Pred_{Image} * 0.5 \tag{3.1}$$

3.4 Hyper-parameters

In machine learning, a hyper-parameter is a variable whose value controls the learning process of the model. These parameters express the complexity of the model and how fast it should learn. Hyper-parameters are usually fixed before the actual training process begins.

We can distinguish two types of hyper-parameters:

1. **Optimizer's [4] hyper-parameters:** These are the variables or parameters related to the training process:
   - **Batch size:** has an effect on the resource requirements of the training, it is the number of training examples utilized in each iteration.
   - **Learning rate (Lr):** controls how much we are adjusting the weights of our neural network with respect to the gradient.
   - **Epoch:** is one complete pass forward and backward through the training data.
2. **Model's hyper-parameters:** These are the variables related to the architecture or structure of the model. It helps define the model complexity on the basis of :
   - **Number of layers:** is the number of layers that constitute the network.
   - **Hidden units:** is the number of hidden layers in a neural network.
   - **Model's specific parameters:** for the architectures like RNN: Choosing a cell type like LSTM units or BiLSTM units) and how deep the model is (number of units).

There are multiple ways to optimize the hyper-parameters of the model, but most of them take a lot of computation time.

Since we have limited resources, we chose to manually tune the hyper-parameters while testing models and evaluating results.

3.5 Project Workflow

We will present the project workflow delineated by how we organized the work to make it easier to get better results.

3.5.1 Project's folder structure:

Figure 3.7 represents the composition of the project folder:

[4] We used two types of optimizer: SDG which is Gradient descent (with momentum) optimizer and ADAM which implements the Adam algorithm.

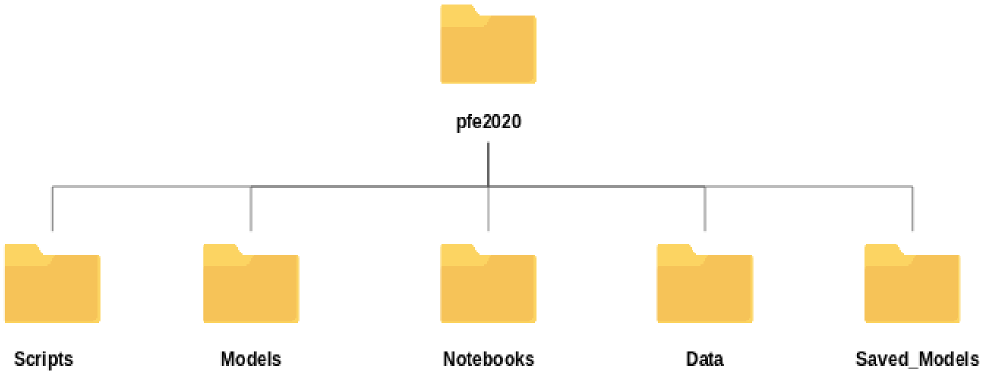


Figure 3.7: Project Folder decomposition

The project folder contains:

- **Scripts:** contains mainly the scripts used for image crawling and text crawling and some scripts for preparing image data.
- **Models:** contains the scripts of every used model for both text classification and image classification.
- **Notebook:** contains almost every experience done including pre-processing results and training models.
- **Data:** composed of a data loader for both image data and text data to simplify uploading data.
- **Saved_models:** contains saved models with the log of each model.

3.5.2 Training loop

We followed this cycle of training steps for each model:

1. **Load data:** We begin by loading train, validation and test data by simply calling loaders from the data folder and choosing the right parameters. For example, for visual data:

```
from data import Images_load
train, validation, test = Images_load.load_data()
```

2. **Load the model's script:** Next, we will call the function created for the model, which will begin by preparing data and train the model.

```
from models.Image_classifier import Deep_CNN
```

3. **Train the model:** we can train the model by simply calling the model function that we created as follows:

```
CM,CR=Deep_CNN.model(train, validation, test,  batch_size=128, lr=0.0001, epochs=10)
```

This will return the textual progress of the training on one hand, and return the confusion matrix with a confusion report on the other hand.

4. **Test the model:** we scan the results given by the fit command to check if we have problems whether with data or with the model. For example, if we notice an important difference between train accuracy and test accuracy, we have to think about over-fitting and find a solution accordingly. Also we calculate the confusion matrix and the performance metrics for the model.
5. **Save Model:** The model's architecture and weights are automatically saved while training in order to gain time and load the model when needed.

For each loop of each model, we may repeat steps three and four, while manually tuning the hyper-parameters when calling the model function.

Conclusion

In this chapter we presented the design of our solution for this study including the sources of our different data-sets and the inspiration of the idea. Then we proceeded with the architectures of the different models that we suggested for both images and texts classification and next we presented the method of aggregation of the models. Finally, we drew the big picture of the project workflow and the steps followed for each model. In the next chapter, we will present the method of crawling data and some visualization, then we will present the experimental results of the models and their aggregation.

Chapter 4

Implementation and Experimental Results

This chapter states the results of our study and the implementation of our models. We will start with the tools we used to establish this study, then we will present the process and results of data collection and preprocessing of visual and textual data-sets. Finally, we will finish with the main goal of this study which includes the results of both image classification and text classification.Moreover, we will look at the worth of aggregating these two models to better predict users suffering from depression.

4.1 Technological Tools

Working with DL requires using powerful hardware and various software. In this section we will detail the hardware and the technologies we used in our study.

4.1.1 Hardware

In the process of the implementation of our solution we used two main machines, a local machine for refactoring codes, testing models and research, and a virtual machine (VM) on Google Cloud Platform (GCP) to run models and codes that are heavy in terms of computation and time. Following are the specifications of these machines:

- **Local Machine:** Lenovo E330
    - ∗ Operating System : Kali Linux 2020.2
    - ∗ CPU: Intel Core i5-3230M 2,6GHz
    - ∗ RAM: 8 Go DDR3
    - ∗ Disk: 320 GB HDD

- **Virtual Machine:** *mastermind*
  We started working on GCP on May 2nd, it presents a variety of specifications for VM that can be modified even after creation. To minimize the cost of the VM, we worked with two main configurations:
  - For training models:
    - Operating System : Ubuntu 19.10
    - Machine Type: n1-highmem-8
    - CPU: 8 vCPUs
    - RAM: 52 Go
    - Disk: 100 GB SSD
      This configuration costs $0.6068 per hour when started and $0.1320 while stopped.
  - For tasks that are not heavy in both computation and time:
    - Operating System : Ubuntu 19.10
    - Machine Type: n1-highmem-8
    - CPU: 8 vCPUs
    - RAM: 16 Go
    - Disk: 100 GB SSD
      This configuration costs $0.2236 per hour when started and $0.0913 while stopped.

  Since we are using the 12-month free trial with $300 credit in Google Cloud Platform, we were not able to use GPUs and the limit of CPU was the type we used for training models.

4.1.2 Software and technologies:

We used various soft-wares and technologies to implement project's steps, we will state them bellow:

- **Python (version 3.8.3):** is one of the most used programming languages for machine learning and data science. It has many large and comprehensive standard libraries (packages) especially for deep learning.
- **Anaconda (Distribution 2020.02):** is an open source package management system and environment management system that runs on most of the operating systems. We chose Anaconda for its rapidity in installing, running and updating packages and their dependencies on Python.

- **TensorFlow:** is an open source end-to-end platform dedicated to machine learning, it has a comprehensive, flexible ecosystem of tools and libraries that make it easier to build machine learning models.
- **Keras:** is an open-source neural-network library written in Python. We used it while running on top of TensorFlow.
- **Git:** is a free and open source distributed version control system designed to organize from small to very large projects with speed and efficiency. We used it with GitLab (web-based DevOps lifecycle tool that provides a Git-repository manager) to organize our project workflow.
- **Jupyter Notebook:** is an open-source web application that allows us to create documents that contain live code, equations, visualizations and narrative text. This helped us in exploring data-sets and preprocessing techniques, testing built models and saving the history of the work.
- **Pycharm (Community Edition):** is used for refactoring codes of all our work. In fact, it's an Integrated Development Environment (IDE) for Python language. It provides code analysis, a graphical debugger and Git Integration plugin.

4.2 Data preparing

To prepare data for modeling, there are multiple steps that we followed to create an easy and fast pipeline for loading data both visual and textual.

We will present the process of collecting, cleaning and creating a loader for both visual and textual data separately.

4.2.1 Images

We will start by the process of crawling images, then how we cleaned these data and finish with creating the loader to make it easier when writing the script for the model.

4.2.1.1 Collecting Images

As explained in chapter three, we used keywords to search for images from two websites Unsplash and Pexels and crawled the data using two different methods. Although the process that we used in crawling images for the two websites is common (figure 3.1), the practical functions differ for the two websites. We present how data were collected for both websites:

Unsplash:
We created a function to crawl the links of all images of a specific keyword using the API of unsplash. To crawl links we can simply call this function with the proper parameters to save links in a csv file.

Pexels:
We did not use the Pexels' API to crawl links of images for a specific keyword, we created a function using Selenium Python [1] to save links in a csv file easy to implement.

After scraping the links of photos, we used a function to save these images in a local directory for further processing. This task took almost two weeks to achieve due to various problems such as interruption in internet connection, images' size (every image is almost $4.0MB$) and technical issues.

We collected a total of :

- **6250** images labeled as "Depressed"
- **5234** images labeled as "Not Depressed"

4.2.1.2 Cleaning Data

Images were collected using the same keywords, therefore, we had to think about a solution to eliminate repetitive images. Fdupes [2]for Linux was the best option we used to remove duplicate images.

After cleaning data, the number of images decreased from **6250** images labeled as "Depressed" to **4023** images and from **5234** images labeled as "Not Depressed" to **4197** images.

4.2.1.3 Creating loader

To optimize the data generation process for easier training procedure, we created a loader for crawled images following these steps:

- Resize and convert images to npy [3] format.
- Use npz [4] to zip all images to reduce its size from $63.9GB$ to $1.2GB$.
- Upload zipped files to mediafire.
- Create loader to make the generation of data easier when needed.

It returns a data-set splitted into train, validation and test.

[1] Selenium Python provides a simple API to write functional using Selenium WebDriver. Through Selenium Python API we can access all functionalities of Selenium WebDriver in an intuitive way.

[2] fdupes is a Linux utility for identifying or deleting duplicate files by comparing md5sum then running a byte-to-byte comparison

[3] This file format makes incredibly fast reading speed enhancement over reading from other types.

[4] npz is a file format by numpy that provides storage of array data using gzip compression

Results:

We may notice in the sample of images in figure 4.1 that images present different features in spite of belonging to the same class.

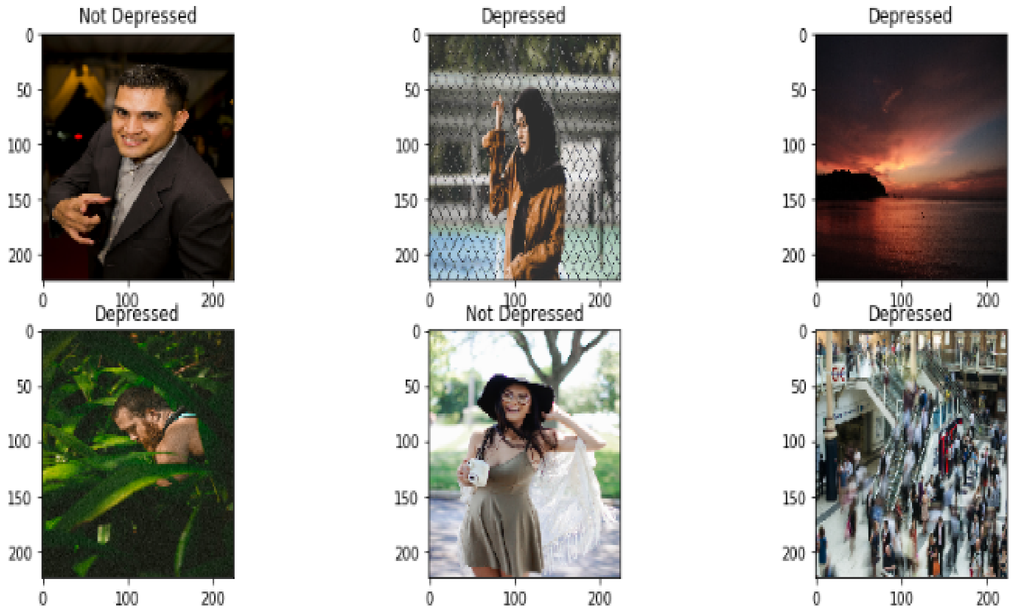


Figure 4.1: Sample of images data: Image size is 224*224 pixels

The final step of preparing data is resizing images based on the specificity of each model, converting it into a matrix of pixels and normalizing it (e.g. divide it by 255).

Now that we have images data ready to be used in training our models, we move to how we manage to get textual labeled data.

4.2.2 Textual data

As for visual data preparing, we followed the same steps in textual data preparation. First we started with collecting data from Twitter, then we cleaned it and finally created a loader to easily use it in training and other processing tasks.

4.2.2.1 Collecting and cleaning data

Tweets were collected using Twitter Intelligence Tool (TWINT)[5]. We created a function that extracted information about each tweet containing the keywords we previously stated in chapter three, then we created a function to extract timeline posts based on the labeled username, we simultaneously tried to get effective data for classification, so we investigated it for duplicate profiles (having different classes) and eliminated them.

[5] It is a package written in Python for scraping Tweets from Twitter profiles without using Twitter's API, https://github.com/twintproject/twint

4.2.2.2 Creating data loader

Crawled tweets were saved in separate csv files for each user. The data loader combines these files into one csv file having the tweets and the corresponding labels. This loader returns the train, validation and test set of textual data.

4.2.2.3 Pre-processing data

We noticed that the words are flexible and variant in the raw data of crawled social media, which causes great difficulties in word matching and semantic analysis.

Therefore, we created a customized function to pre-process our data considering the aspect of our study that engages in detecting features in texts that only depressed users use. We wanted to keep as much of the information in these tweets. For this, we created a personalized list of stop words to remove and a list of words with its substitutes to replace (such as bad words and contractions). We also decoded emoticons to emoticon alias using *emoji*[6],a package in Python. Finally, tokenizing tweets was done by *SocialTokenizer* which is a tokenizer belonging to *ekphrasis*[7] which is a text processing tool, adapted to texts from social networks.

Results:

The word clouds of tweets from both classes "Depressed" and "Not depressed" are respectively presented in figure 4.2 and figure 4.3.

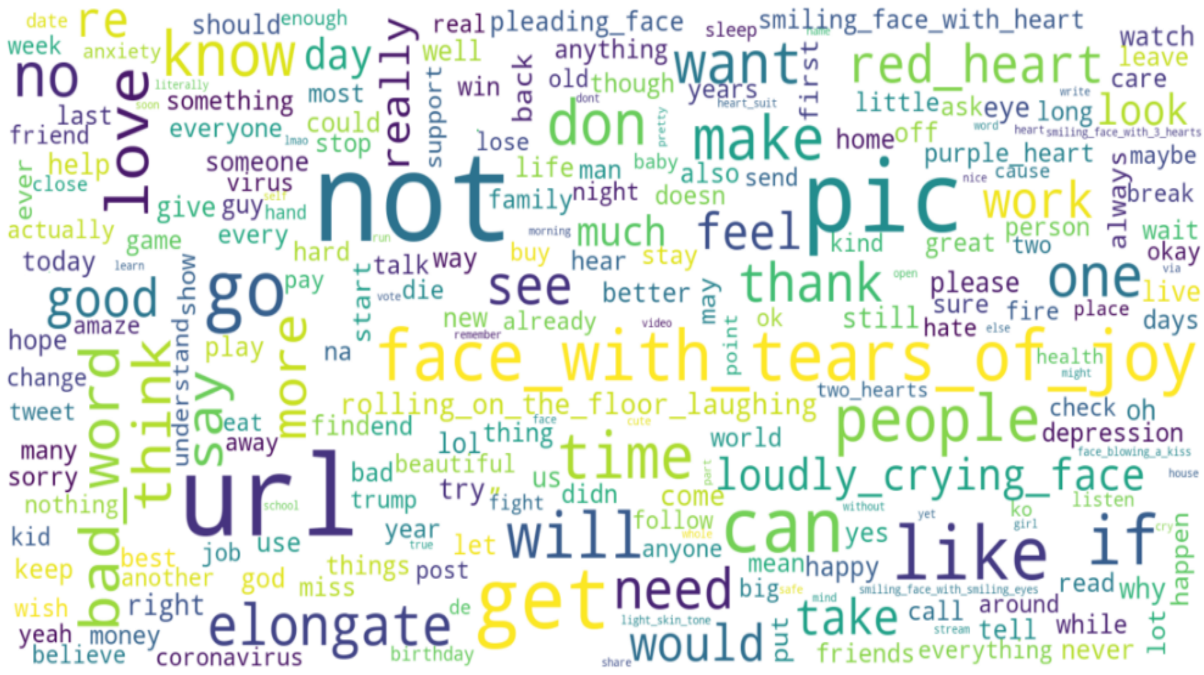


Figure 4.2: Word cloud of depressed users

[6] https://pypi.org/project/emoji/
[7] https://github.com/cbaziotis/ekphrasis

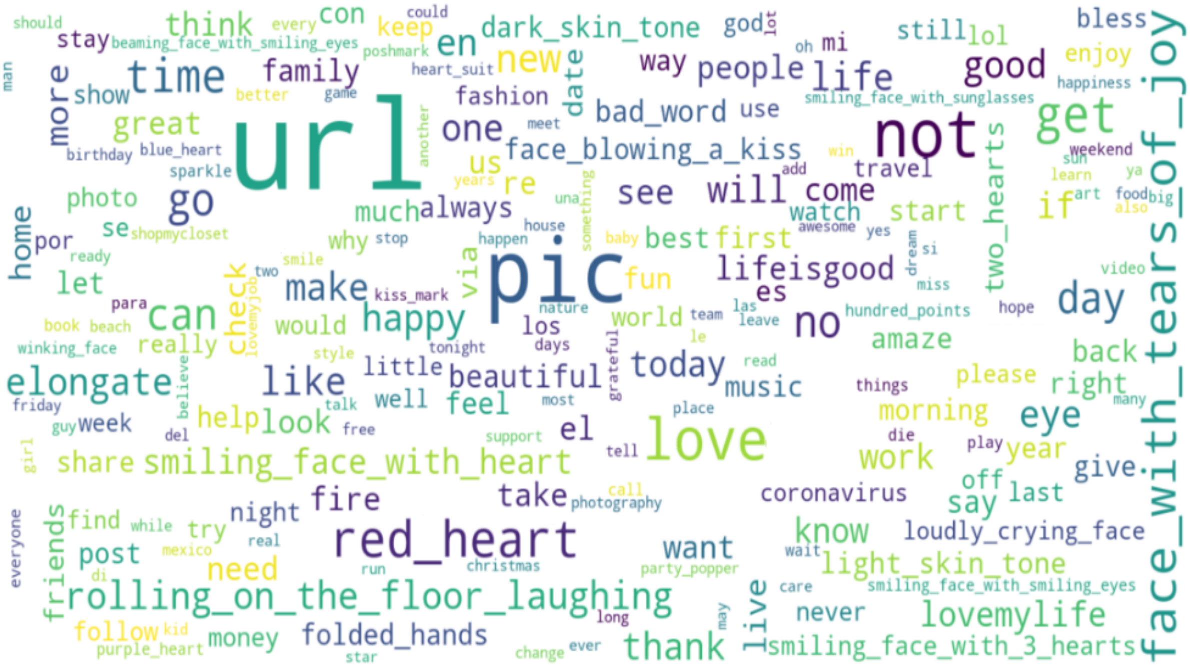


Figure 4.3: Word cloud of not depressed users

It's easy to say that the content of textual posts differs for both classes by the distinction of various words used more often by individuals labeled depressed such as bad words, "need" and emojis such as "face_with_tears_of_joy". One common word that we may notice is "pic" which means that the user shared a post containing a photo, this enhances our interest in studying the effect of detecting depression using the combination of texts and images.

4.3 Models experiments and results

We will present the results of five selected neural network models architectures, which are used to evaluate the performance of depression detection through images and texts and then we will aggregate the best two models of images and text classification.

4.3.1 Image classification

Using models as presented in chapter three, we did many experiments following the steps previously defined in 3.5.2. The results we got while manually tuning the hyper-parameters are presented in table 4.1.

Size of images	Model	Optimizer	Batch	Epochs	Lr	Train accuracy	Test accuracy	Time
224*224	**Deep CNN**	**ADAM**	**128**	**10**	**0.001**	**0.7634**	**0.7146**	**44 min**
224*224	Deep CNN	ADAM	256	10	0.001	0.7251	0.6981	1h 20min
224*224	Deep CNN	ADAM	128	20	0.0001	0.7511	0.70102	2h 20min
224*224	ResNet50	ADAM	64	10	0.001	0.69757	0.6012	3h 32min
224*224	ResNet50	ADAM	64	10	0.0001	0.4903	0.488	1h 31min
224*224	ResNet50	ADAM	128	10	0.001	0.60637	0.59560	2h 18min
224*224	**ResNet50**	**ADAM**	**128**	**20**	**0.0001**	**0.75536**	**0.7091**	**4h 30min**
224*224	ResNet50	ADAM	50	30	0.001	0.7469	0.7073	3h 56min
224*224	ResNet50	ADAM	32	30	0.001	0.73281	0.695	4h 38min
224*224	ResNet50	ADAM	50	30	0.0003	0.645196	0.64585	2h 27min
128*128	BiT-L (ResNet50x1)	SGD	None	10	Variable	0.8266	0.8117	41min
128*128	BiT-L (ResNet50x1)	SGD	None	50	Variable	0.8305	0.8126	3h
128*128	BiT-L (ResNet50x3)	SGD	None	10	Variable	0.8697	0.8180	3h 2min
128*128	BiT-L (ResNet101x1)	SGD	None	10	Variable	0.8475	0.7673	1h 11min
128*128	**BiT-L (ResNet101x1)**	**SGD**	**None**	**50**	**Variable**	**0.8889**	**0.8253**	**2h 4min**

Table 4.1: Experiments of image classification models

BiT-L (ResNet101x1), using SGD optimizer and in 50 epochs (other hyper-parameters are already fixed because it's a pre-trained model), gave us the best accuracy for test data-set which is 0.8253 for detecting depression through images. We calculated the confusion matrix and the metrics for this model using the test data-set to get better insights, and we got very impressive results shown in table 4.2 and table 4.3.

		Predicted labels	
		Depressed	**Not Depressed**
Actual labels	**Depressed**	808	193
	Not Depressed	165	884

Table 4.2: Confusion matrix of BiT-L (ResNet101x1) model

	Precision	Recall	F1-score	Support
Depressed	0.83	0.80	0.81	1001
Not depressed	0.82	0.84	0.82	1049

Table 4.3: Metrics of the BiT-L (ResNet101x1) model

Conclusion:

The BiT-L (ResNet101x1) model outperformed all the models we manipulated, therefore we saved this model using keras to further load it for the aggregation of models.

At this point we have a robust model for detecting depression using images' features, we will then move to results of text classification.

4.3.2 Text classification

We conducted experiments on the data-set crawled from Twitter, where depressed class has 223014 samples and not depressed class has 187360 samples. Using LSTM model and the GloVe embedding plus BiLSTM model gave us results shown in table 4.4 with a variation of hyper-parameters:

Size of data sample	Model	Batch	Epochs	Lr	Train accuracy	Test accuracy	Time
10000 (not pre-processed)	LSTM	128	10	0.001	0.8721	0.5241	1h 30min
10000	LSTM	264	10	0.001	0.8678	0.5365	1h 49min
10000	LSTM	128	10	0.0001	0.9691	0.6101	1h 12min
20000	LSTM	128	20	0.00001	0.5891	0.554	7h 2min
10000	GloVe+BiLSTM	32	20	0.001	0.7298	0.6321	9h 5min
10000	**GloVe+BiLSTM**	**32**	**20**	**0.0001**	**0.7003**	**0.6957**	**12h 50min**
10000	**GloVe+BiLSTM**	**128**	**20**	**0.0001**	**0.7362**	**0.7219**	**14h 20min**

Table 4.4: Experiments of text classification models

According to the table 4.4, we could notice that the GloVe+BiLSTM model outperformed the LSTM model. Since we calculated the confusion matrix and the metrics for each model, we will present results for the models that gave us the best accuracy (0.6957 and 0.7219) to decide which one to choose:

	Precision	Recall	F1-score	Support
Depressed	0.70	0.61	0.65	635
Not depressed	0.70	0.77	0.73	739

Table 4.5: Metrics of model with 0.6957 accuracy

	Precision	Recall	F1-score	Support
Depressed	0.75	0.54	0.63	635
Not depressed	0.71	0.86	0.78	739

Table 4.6: Metrics of model with 0.7219 accuracy

Conclusion:
Judging by the recall and precision of depressed users' class which is the main focus of our study, we will choose the model with 0.6957 for the aggregation and further prediction.

At this stage, we have a robust model for image classification and a good model for text classification, we will then pass to the aggregation of these models.

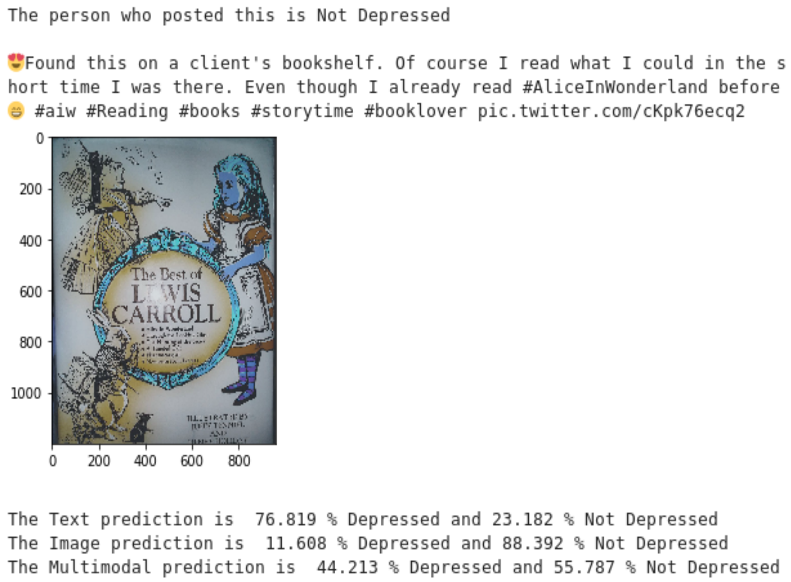
4.3.3 Aggregation of models

The aggregation of the models was done following these steps:

1. Load best saved image classification model which is BiT-L (ResNet101x1).

2. Load best saved text classification model which is GloVe+BiLSMT.
3. Create function for classifying new data with both models.
4. Create function that applies the equation previously defined 3.1.

Unfortunately due to the limited time of the project, we did not have enough data that allow us to effectively test this multi-modal model that takes two types of inputs, but based on a sample of six labeled posts that include both images and texts, we were able to detect an amelioration of the model that surely incite us to continue future research on this topic. In figures 4.4 and 4.5, we present a sample of the prediction that the multi-modal model gave us.

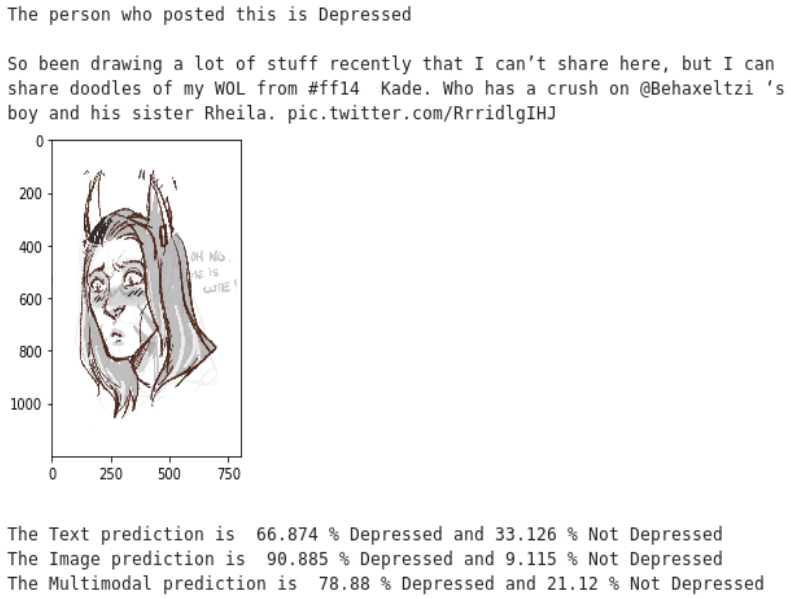


```
The person who posted this is Not Depressed

😍Found this on a client's bookshelf. Of course I read what I could in the s
hort time I was there. Even though I already read #AliceInWonderland before
😄 #aiw #Reading #books #storytime #booklover pic.twitter.com/cKpk76ecq2
```

```
The Text prediction is  76.819 % Depressed and 23.182 % Not Depressed
The Image prediction is  11.608 % Depressed and 88.392 % Not Depressed
The Multimodal prediction is  44.213 % Depressed and 55.787 % Not Depressed
```

Figure 4.4: Post of a user who is labeled not depressed and the results of prediction of each model



```
The person who posted this is Depressed

So been drawing a lot of stuff recently that I can’t share here, but I can
share doodles of my WOL from #ff14  Kade. Who has a crush on @Behaxeltzi ‘s
boy and his sister Rheila. pic.twitter.com/RrridlgIHJ
```

```
The Text prediction is  66.874 % Depressed and 33.126 % Not Depressed
The Image prediction is  90.885 % Depressed and 9.115 % Not Depressed
The Multimodal prediction is  78.88 % Depressed and 21.12 % Not Depressed
```

Figure 4.5: Post of a user who is labeled depressed and the results of prediction of each model

Conclusion

In this final chapter, we went through the work environment setup of this graduation project. Then, we presented the steps we followed to collect and clean textual and visual data for processing.

Subsequently, based on the models' performances, we decided which models to keep for both textual and visual data and finally we established the aggregation of the best two models that we got from the conducted experiments.

Conclusion and future work

Throughout this project, we designed and implemented a novel approach to detect depression through social networks aggregating two effective models to form a model that accepts a multi-modal input. This project consisted of a research work where we tried new approaches for depression detection using Deep Learning based models.

The first step of our project was a research work on current studies done on early detection of mental diseases through social networks. Research work included the understanding of the process followed by researchers to build datasets and models for this kind of subject. The second step was to collect data from various social networks platforms to gather sufficient data for training. The final steps, were the implementation, the test and the discussion of the results of each model trained on both images and texts.

The performance results of these models showed two models, the first one that was able to distinguish images posted by depressed people and the second one that was able to extract psycho-linguistic features from texts to identify people having signs of depression.

After having two robust models, we tried aggregating them into one model that accepts multi-modal input, and this gave us some promising results that unfortunately could not be effectively tested.

Future studies could fruitfully explore the effectiveness of this aggregation further by collecting data and grid searching the appropriate coefficients of image and text prediction. Also, due to a problem of hard-ware, the text classification model was not able to train on the whole data, so future research might enhance the performance of the models that we trained by increasing the capacities of used hard-ware or optimizing the hyper-parameters using approaches such as grid search or Bayesian optimization.

'The important thing is to not stop questioning. Curiosity has its own reason for existing.'

Albert Einstein

Bibliography

[1] J. Clement. Number of global social network users 2010-2023, April 2020.

[2] Michael Gamon, Munmun Choudhury, Scott Counts, and Eric Horvitz. Predicting depression via social media. *Association for the Advancement of Artificial Intelligence*, 07 2013.

[3] J Myers and Myrna Weissman. Use of self-report symptom scale to detect depression in a community sample. *The American journal of psychiatry*, 137:1081–4, 10 1980.

[4] Aron Halfin. Depression: The benefits of early and appropriate treatment. *The American journal of managed care*, 13:S92–7, 12 2007.

[5] Luca Avena, Fabienne Castell, Alexandre Gaudillière, and Clothilde Melot. Random forests and networks analysis. *Journal of Statistical Physics*, 11 2017.

[6] Md Rafiqul Islam, Ashad Kabir, Ashir Ahmed, Abu Kamal, Hua Wang, and Anwaar Ulhaq. Depression detection from social network data using machine learning techniques. *Health Information Science and Systems*, 6:8, 12 2018.

[7] Michael Gamon, Munmun Choudhury, Scott Counts, and Eric Horvitz. Predicting depression via social media. *Association for the Advancement of Artificial Intelligence*, 07 2013.

[8] Sharath Chandra Guntuku, David Yaden, Margaret Kern, Lyle Ungar, and Johannes Eichstaedt. Detecting depression and mental illness on social media: an integrative review. *Current Opinion in Behavioral Sciences*, 18:43–49, 12 2017.

[9] Tiancheng Shen, Jia Jia, Guangyao Shen, Fuli Feng, Xiangnan He, Huanbo Luan, Jie Tang, Thanassis Tiropanis, Tat-Seng Chua, and Wendy Hall. Cross-domain depression detection via harvesting social media. pages 1611–1617, 07 2018.

[10] Marvin Papert Minsky and Seymour. *Perceptrons*, volume 1. M.I.T. Press, 1969.

[11] Kunihiko Fukushima. A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Neocognitron*, 1980.

[12] K. Fukushima. Neocognitron. *Scholarpedia*, 2(1):1717, 2007.

[13] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

[14] Maria Valueva, Nikolay Nagornov, Pavel Lyakhov, G.V. Valuev, and N.I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177, 05 2020.

[15] Y. Bengio and Yann Lecun. Convolutional networks for images, speech, and time-series. 11 1997.

[16] Hidenori Ide. and Takio Kurita. Improvement of learning for cnn with relu activation by sparse regularization. *IEEE*, 07 2017.

[17] Robert DiPietro and Gregory D. Hager. *Deep learning: RNNs and LSTM*, volume 1. Handbook of Medical Image Computing and Computer Assisted Intervention, 2020.

[18] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.

[19] S. Hochreiter and J. Schmidhuber. Tlong short-term memory. *Neural computation*, 9:1735–1780, 04 1997.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. pages 473–479, 01 1996.

[21] Nir Arbel. How lstm networks solve the problem of vanishing gradients, December 2018.

[22] Oinkina and Hakyll. Understanding lstm networks, August 2015.

[23] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997.

[24] Hassan Saif, Miriam Fernandez, and Harith Alani. On stopwords, filtering and data sparsity for sentiment analysis of twitter. 05 2014.

[25] Jianqiang Zhao and Xiaolin Gui. Comparison research on text preprocessing methods on twitter sentiment analysis. *IEEE Access*, PP:1–1, 02 2017.

[26] MF Porter. An algorithm for suffix stripping. *Program: Electronic Library and Information Systems*, 14, 03 1980.

[27] Chris Paice. Another stemmer. *SIGIR Forum*, 24:56–61, 11 1990.

[28] Liang-Chih Yu, Jin Wang, K. Lai, and Xuejie Zhang. Refining word embeddings using intensity scores for sentiment analysis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, pages 1–1, 12 2017.

[29] John Watson. Psychology as a behaviorist views it. *Psychological Review - PSYCHOL REV*, 20:158–177, 01 1913.

[30] Ahmed Husseini Orabi, Prasadith Buddhitha, Mahmoud Husseini Orabi, and Diana Inkpen. Deep learning for depression detection of twitter users. pages 88–97, 01 2018.

[31] Guangyao Shen, Jia Jia, Liqiang Nie, Fuli Feng, Cunjun Zhang, Tianrui Hu, Tat-Seng Chua, and Wenwu Zhu. Depression detection via harvesting social media: A multimodal dictionary learning solution. pages 3838–3844, 08 2017.

[32] American Psychiatric Association. *Diagnostic and Statistical Manual of Mental Disorders*, volume 5. American Psychiatric Publishing, 2013.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 06 2016.

[35] Xiaohua Zhai Joan Puigcerver Jessica Yung Sylvain Gelly Alexander Kolesnikov, Lucas Beyer and Neil Houlsby. Big transfer (bit): General visual representation learning. pages 1–28, 12 2019.

[36] Jeffrey Pennington, Richard Socher, and Christoper Manning. Glove: Global vectors for word representation. *EMNLP*, 14:1532–1543, 01 2014.