IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 4, 2021, accepted January 19, 2021, date of publication January 22, 2021, date of current version February 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3053917

Sentence-Level Classification Using Parallel Fuzzy Deep Learning Classifier

FATIMA ES-SABERY[1], ABDELLATIF HAIR[1], JUNAID QADIR[2], BEATRIZ SAINZ-DE-ABAJO[3], BEGOÑA GARCÍA-ZAPIRAIN[4], (Member, IEEE), AND ISABEL DE LA TORRE-DÍEZ[3]

[1]Department of Computer Science, Faculty of Sciences and Technology, Sultan Moulay Slimane University, Beni Mellal 23000, Morocco
[2]Department of Electronics, Quaid-i-Azam University, Islamabad 45320, Pakistan
[3]Department of Signal Theory, Communications and Telematics Engineering, University of Valladolid, 47011 Valladolid, Spain
[4]eVIDA Research Group, University of Deusto, 48007 Bilbao, Spain

Corresponding authors: Beatriz Sainz-De-Abajo (beasai@tel.uva.es) and Fatima Es-Sabery (fatima.essabery@gmail.com)

This work was supported by the eVida Research Group, University of Deusto, Bilbao, Spain, under Grant IT 905-16.

ABSTRACT At present, with the growing number of Web 2.0 platforms such as Instagram, Facebook, and Twitter, users honestly communicate their opinions and ideas about events, services, and products. Owing to this rise in the number of social platforms and their extensive use by people, enormous amounts of data are produced hourly. However, sentiment analysis or opinion mining is considered as a useful tool that aims to extract the emotion and attitude from the user-posted data on social media platforms by using different computational methods to linguistic terms and various *Natural Language Processing* (NLP). Therefore, enhancing text sentiment classification accuracy has become feasible, and an interesting research area for many community researchers. In this study, a new *Fuzzy Deep Learning Classifier* (FDLC) is suggested for improving the performance of data-sentiment classification. Our proposed FDLC integrates *Convolutional Neural Network* (CNN) to build an effective automatic process for extracting the features from collected unstructured data and *Feedforward Neural Network* (FFNN) to compute both positive and negative sentimental scores. Then, we used the *Mamdani Fuzzy System* (MFS) as a fuzzy classifier to classify the outcomes of the two used deep (CNN+FFNN) learning models in three classes, which are: Neutral, Negative, and Positive. Also, to prevent the long execution time taking by our hybrid proposed FDLC, we have implemented our proposal under the Hadoop cluster. An experimental comparative study between our FDLC and some other suggestions from the literature is performed to demonstrate our offered classifier's effectiveness. The empirical result proved that our FDLC performs better than other classifiers in terms of true positive rate, true negative rate, false positive rate, false negative rate, error rate, precision, classification rate, kappa statistic, F1-score and time consumption, complexity, convergence, and stability.

INDEX TERMS Deep learning, convolutional neural network (CNN), sentiment analysis, feedforward neural network (FFNN), fuzzy logic, Hadoop framework, MapReduce, Hadoop Distributed File System (HDFS).

I. INTRODUCTION

Social network platforms like Instagram, Twitter, Youtube, LinkedIn, and Facebook have been considered essential and indispensable in our daily activities. Day-to-day, billions of social media users disseminate billions of personal or professional posts [1]. For example, Marketers use social media to spread professional posts that endeavor to present, promote, advertise, and market their products, services, events, and brand names. On the other hand, the customers interact with the marketers' posts by express their feelings, opinions, ideas, attitudes about the presented products or services [2]. Further, the marketers gather the customer's feedback, study, and analyze it using the sentiment analysis tool. The main objective from they are doing these operations is to improve the quality of their products and services, enhance their offerings by adding other privileges, and improve their brand performance [1], [2].

Sentiment Analysis (SA) plays a significant role in *Business Intelligence* (BI). In BI, it uses to get responses for questions such as, 'Why is product sales so low?', 'Have customer's needs are fully satisfied by utilizing our products?',

The associate editor coordinating the review of this manuscript and approving it for publication was Jing Bi.

'What did they like the most about our services?, What did they dislike the most?','Are our customers pleased by using our products/services or require more?'. We employ sentiment mining tools and techniques to find the relevant answers to these questions [3]. Such as NLP methods for data pre-processing like word stemming, word lemmatization, and effect of negation. *Machine Learning* (ML) methods for sentiment classification like *Support Vector Machines* (SVM) [4], *K-Nearest Neighbors* (KNN) [5], *Random Forest* (RF) [6], *Logistic Regression* (LR) [7], *Naive Bayesian* (NB) [8] or *Decision Tree* (DT) classifier [9]. *Deep Learning* (DL) methods like *Convolutional Neural Network* (CNN) [10], *Feedforward Neural Network* (FFNN) [11], *Long Short-Term Memory* (LSTM) [12], *Gated Recurrent Unit* (GRU) [13], or *Recurrent Neural Network* (RNN) [14] which are used for automatically extracting features from the given text and for performing the sentiment classification. and vectorization methods like word embeddings (word2vec, glove) [15], tf-idf [16], bag-of-words [17], fast-text, n-gram or character-level [18], which are used for representing the given text by numerical values.

SA is also termed opinion mining and pursues to recognize people's moods or emotions toward an entity like events, products, individuals, services, or topics. At present, SA has been mainly deemed a sentiment classification task in the context of ML, that is to say, each expressed sentiment in the given text is classified as positive, neutral, or negative. Data-driven techniques, involving ML and DL methods, are considered as one accurate and efficient solution to carry out the sentiment classification task. The application of ML methods has proved that they are a powerful tool for classifying expressed sentiments in the given text. In particular, SVM, NB, DTs, RF, and LR methods, etc. which are used extensively with high accuracy in wide application fields that include sentiment analysis, such as cyberhate detection [19] movie and product reviews [20], [21], abusive language detection [22], cyberbullying identification [23], and social media [24]. In addition to classical ML algorithms as presented earlier, there are likewise DL algorithms such as CNN, FFNN, LSTM, GRU, and RNN, which are presently preferred for sentiment classification.

Over the ages, several methods have been proposed to supply users with an effective process for classifying sentiments. These methods have evolved from dictionary-based approaches to ML techniques and presently to DL models. According to past comparative studies and analyses that are carried out on SA using both ML and DL. DL is more efficient than ML in sentiment classification issues due to massive data [25]. Fig. 1 proves that the accuracy of conventional ML algorithms is better for a lesser size of data. As the volume of data rises beyond a particular number, the accuracy of conventional ML algorithms becomes constant. In contrast, the accuracy of DL algorithms raises with respect to the rise in the volume of data.

A principal difference between ML approaches and DL models is in the manner that the features are extracted.

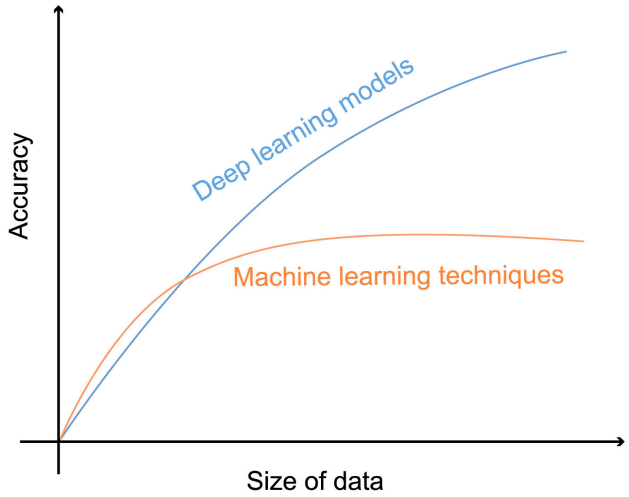


FIGURE 1. The accuracy of ML approaches compared to the accuracy of DL models with respect to data size.

As known that the accuracy of a DL or ML approach is extremely dependent on a good feature extraction process from given data. Classical ML techniques utilize hand-crafted engineering features by employing many feature extraction methods, and thence apply the learning methods. This process takes high time-consuming and extracts incomplete features. In contrast, the features are extracted automatically in the case of DL, which is the powerful point of DL models against ML techniques. Due to the fast-growing size of data in this era and to the good performance of DL models on the massive size of data, we decide to combine both CNN and FFNN in order to build an effective automatic process for extracting the features from the given social media dataset. But although the use of the most progressing DL models, there is ingrained ambiguity in NLP that needs more solutions. Based on several studies [26], [27], the fuzzy logic theory is deemed the most effective solution to deal with vagueness data. Beer [26] demonstrates in his paper that the fuzzy logic theory's strength is its resemblance to human reasoning and natural language. One substantial property of the fuzzy logic theory is the used technique for computing with terms. This technique serves to transform the terms into numerical values for reasoning, inference, and computing. Fuzzy logic supplies us with an eligible manner to handle linguistic issues at fuzziness data. Zadeh *et al*. [27] discuss in their work that no other technique resolves these linguistic issues. Accordingly, to advantages introduced in [27] about fuzzy logic theory in the area of fuzziness data, we decide to employ this theory in our work to deal with the inherent uncertainty in the social media data.

The fuzzy logic theory is deemed as an ameliorated version of the deterministic logic theory. i.e., in fuzzy logic theory, the linguistic variable takes a real value between 0 and 1 instead of, in deterministic logic, each variable takes either 0 or 1. The principal purpose of the fuzzy logic theory is transforming a white and black issue into a grey problem [28].

In the descriptions of set theory, deterministic or conventional logic assumes that the set of linguistic variables as the crisp linguistic variables, which signifies that each linguistic variable's measured membership degree in the crisp collection is equal to 1. In other words, the linguistic variable ultimately belongs to the crisp set. In contrast, fuzzy logic is regarding the collection of linguistic variables as the fuzzy collection, which indicates that each linguistic variable's membership value in a fuzzy group is varied from 0 to 1. In other words, each linguistic variable belongs partially to the fuzzy collection. The membership degree is measured by a particular *Membership Function* (MF) such as triangular MF, Gaussian MF, and trapezoidal MF [2], [3].

In order to enhance the classification rate of the sentence-level classification and to overcome the previously discussed shortcomings as best as we can, we introduce a new *Fuzzy Deep Learning Classifier* (FDLC) to recognize the polarity (negative, positive, neutral) of sentiment sentences. Our methodology mainly contains five parts: One part is the data-preprocessing steps in order to reduce the noisy data and enhance the data quality; The second part is the application of word embedding methods to convert the text-based data to numerical-based data. The third part is the proposed hybrid DL model that combines CNN and FFNN in order to build an effective automatic process for extracting the features from collected unstructured-data and compute both *Positive Sentimental Score* (PSS) and *Negative Sentiment Score* (NSS), and the four-part is the MFS which is used as a fuzzy classifier to classify the outcomes of the two used deep (CNN+FFNN) learning models into three classes Neutral, Negative and Positive. Finally, we use the Hadoop framework to parallelize our FDLC to overcome the long execution time problem. In addition–to prove the performance of our suggested FDLC– an experimental comparative study between our model and some other models from the literature is carried out, and the practical result proved that our FDLC performs better than other models in terms of true positive rate, true negative rate, false positive rate, false negative rate, error rate, precision, classification rate, kappa statistic, F1-score and time consumption, complexity, convergence and stability. The contribution of our work is fundamentally incarnated into six aspects.

1) Data preprocessing techniques are employed to enhance the text based-data quality of the given dataset by removing the existing noise data.
2) Word embeddings methods like word2vec, glove, and fast-text are applied on the given dataset to transform texts into numerical data.
3) DL methods FFNN and CNN with various parameters are employed for computing the NSS and PSS.
4) Fuzzy logic theory (MFS) is applied as the fuzzy classifier on the outcomes of the previous step (NSS and PSS) in order to classify the sentences of the used dataset into three labels negative, neutral and positive. An elevated accuracy has been accomplished because the suggested FDLC elicits more accurate features automatically and various entities from the given dataset.
5) Hadoop framework is used to parallelize our introduced FDLC in order to avoid the long execution time issue.
6) Multiple experiments are carried out to prove the effectiveness of our introduced FDLC, and this FDLC is compared with several selected classifiers from the literature. The experimental results indicate that the suggested model achieves a better classification rate than the other classifiers on the given dataset.

The rest of this work is arranged as follows: Section 2 introduces some similar works selected from the literature in detail. Section 3 explains the basic concepts of MFS, FFNN and CNN in brief. Section 4 describes our suggested FDLC. Section 5 presents the experimental outcomes and comparative study results to demonstrate the effectiveness classification of the proposed classifier, and Section 6 introduces the conclusions and future work.

II. LITERATURE REVIEW

The objective of the SA process is to extract the emotional polarity included in the given sentence. Because of that, Many researchers proposed various sentiment classification approaches based on ML, DL, or hybrid methods. This section presents recently published models in the literature.

Authors of the paper [29] proposed a new approach that combines the multi-scale CNN and LSTM, in order to raise the classification rate and decrease the error rate of the SA. They handle the review text of various commodities in a different manner, while preserving the shared features between each review data, and integrates the global extracted features and local founded features of the review data to enhance the effectiveness of the classification process. Their suggested classifier is based on two principal phases: Firstly, a multi-task training system is applied to detect private, and shared features from a review data of different kinds of commodities. The second phase is the sentence representation using the word embedding methods [30]. A comparative study is carried out by the authors to compare their proposed model with other approaches like LR, RF, NB, KNN, SVM, XGBoost, and Gradient Boosting DT. The experimental results of the comparative study prove that the SA effectiveness of their suggested approach is more accurate than other selected methods from the literature with the accuracy equal to 86.25%.

Lan *et al.* [31] introduced a new deep learning system called SRCLA that combines the cross-layer attention process and the stacked residual RNN and in order to extract more numerous linguistic features then use it for the SA task. The primary purpose of SRCLA is to construct a stacked RNN to identify and filter various kinds of semantic features and then use the new designed cross-layer attention system in order to ameliorate the filtering task. Based on SRCLA, more linguistic features can be identified, and hence the effectiveness of sentiment classification can be enhanced. The authors of this work applied both models Stacked-BiLSTM,

and their model SRCLA on four datasets TREC, SST-1, MR, and SST-2. The experimental outcomes proved that SRCLA attains 3.0% amelioration over SST-1 dataset, 2.0% refinement over SST-2 dataset, 2.5% amelioration over MR dataset, and 1.5% refinement over TREC dataset, compared with Stacked-BiLSTM.

Lin *et al.* [32] developed a new model in order to enhance the effectiveness of SA. The proposed classifier incorporates the capability of Bi-LSTM to select local series of features and the power of multi-Head Attention to identify thorough features. They use a comparative study to improve the proposed model and enhances its performance.

Liu *et al.* [33] proposed a hybrid approach that incorporates the DL models with ML methods to perform better sentiment classification. The proposed model is a bilingual sentiment classification model that is applied to Turkish and Chinese language datasets. This proposed method combines RNN, LSTM, NB, SVM, and word embedding. Their experimental results proved that the accuracy of their suggested approach could attain 89% and performs better than any other ML or DL approach individually.

Jain *et al.* [34], the tweets about renewable energy were classified to be positive, neutral, or negative using five different ML algorithms, which are SVM, KNN, NB, AdaBoost, and Bagging. The authors of this work choose the information gain function and CfsSubsetEvaluation [34] to extract the features from the given datasets. They implement their proposed approach using WEKA Tool and R-Studio. Due to the experimental outcomes, they are deduced that the SVM Algorithm outperforms other ML algorithms when integrated with the CfsSubsetEvaluation function.

Alec*et al.* [35] developed a novel deep learning classifier to sentiment classification through the combination of LSTM with CNN at the kernel level. Their incorporation of LSTM and CNN schema generated a new hybrid deep learning model with elevated accuracy when they applied it on the Internet Movie Database. In addition, the authors present in this paper multiple scheme variations of their suggested model in order to demonstrate their attempts to raise accuracy while decrease overfitting. They performed many experiments with various regularization methods, kernel size and network architectures, to design five models with high performance for comparing with other approaches in the literature. These models achieved 89% in the accuracy metric when they used to predict the polarity score of reviews from the Internet Movie dataset.

Xing *et al.* [36] applied a novel neural network architecture for SA of the market review dataset. They combined the evolving clustering method with LSTM deep learning model. Experimental results of the application of the proposed methodology on sentences from StockTwits prove that the suggested framework achieves good accuracy compared with other existing methods from the literature.

In [37], the authors propose a novel methodology for SA. This methodology combines the CNN, word embedding, RNN, and polarity lexicons. In addition, the proposed system is composed of three CNNs in order to extract high-level features from noisy word embedding representations. The outcome of these three CNNs is aggregated and employed as an entering variable of a fully-connected Multilayer Perceptron. Experimental results accomplished by their classifier were more competitive in several subtasks.

Wang *et al.* [38] proposed a growing deep belief network with transfer learning (TL-GDBN), which is an improved version of the traditional deep belief network (DBN). Their suggested model's objective is to overcome a shortcoming of DBN, which is it is difficult to determine the DBN's optimal structure fastly. Their contribution is carried out in four aspects. First, a simple DBN architecture consists of one hidden layer is implemented and then pre-trained, and the obtained weight parameters after the pre-trained process are kept. Second, They applied TL to transfer the data from the kept weight parameters in the previous step to newly attached units, and hidden layers. The learning process is repeated until achieved the stopping criterion; hence a growing DBN architecture is constructed. Third, the weight parameters obtained after the pre-training process of the proposed TL-GDBN are fine-tuned by applying layer-by-layer partial least square regression from top to bottom. Finally, The experimental results prove that their proposed model gives good performance compared to other models in the literature.

In [39], the authors developed a new hybrid model for binary SA. They combine the global pooling mechanism and one bidirectional long short-term memory (BiLSTM) layer along. They evaluate their work using three widely practiced datasets based on public opinions about movies. Their proposed model achieved an accuracy of 80.500%, 85.780%, and 90.585% on MR, SST2 and IMDb datasets, respectively.

Fuzzy logic is proposed to handle uncertainty and vagueness data. The strength of fuzzy logic is its resemblance to human reasoning and natural language. The use of fuzzy logic in the SA gives good classification accuracy in several works from the literature. Therefore, the effectiveness of fuzzy logic in the sentiment classification area is based on the methodology adopted in the classification problem by popular fuzzy logic systems. The adopted methodology deems a classification issue to be a 'degree of grey' issue rather than a 'black and white' issue [40], which is similar to the sentiment analysis problem, because the sentiment analysis is considered a 'degree of grey' problem. For that, we found various works in the literature using the fuzzy logic systems for resolving sentiment classification problems.

Wu *et al.* [41] proposed a fuzzy logic-based method for SA. Their suggested model is summarized in five stages, which are the phase in which the data is collected, manually data labelling and examination, text pre-preprocessing, The four-phase that severs to extract the essential features, and the final step is the application of the fuzzy classifier. The fuzzy classifier consists of three parts, which are the fuzzification process, IF-THEN fuzzy rules, and the defuzzification process. The authors of this work have compared their proposed

approach with the keyword search approach. The experimental result demonstrates that their fuzzy logic-based method achieved good performance compared to the keyword search techniques in terms of correctness rate and extracted tweets.

Biltawi *et al.* [3] suggested a lexicon-based method to compute the emotional score of Arabic text employing a fuzzy logic theory. The suggested method be composed of two primary parts. Firstly, Arabic sentences are given weight value. Secondly, the fuzzy logic classifier is used to determine the sentiment score to the classified Arabic review. The proposed model has experimented on a massive dataset that contains reviews about Arabic book, and which is a sentiment analysis dataset that contains over 63000 book reviews (8224 negative reviews, 12201 neutral reviews, and 42831 positive reviews). The experimental results displayed 84.04%, 94.87%, 89.13%, and 80.59% for precision, recall, F1-measure, and accuracy, respectively.

In [42], the authors have developed a novel fuzzy rule-based model for SA problems. The novelty of their contributions are i) the suggested method is unsupervised and can be applied to any dataset and to any dictionary. ii) the creation of nine fuzzy rules to classify the sentiment sentences. They have implemented their suggested approach employing three different dictionaries: AFINN, VADER, and SentiWordNet. The proposed method was tested on nine twitter datasets. The experimental result showed that the approach which employs the VADER dictionary takes the least time in execution than the process which employ the SentiWordNet dictionary. For the recall and precision measures, the methods which use the VADER or AFINN dictionary achieved better performance compared to the SentiWordNet dictionary.

Abdul-Jaleel *et al.* [43] introduced a new model to solve the SA issue. This model combines a genetic algorithm with fuzzy logic theory. The incomes to this suggested classifier are a collection of extracted features from a sentiment sentence, and the outcome of this classification model is the classification' decision for classified sentiment sentence. They compared their proposed classification system with the keyword search method by computing both accuracy and incremental rates. In terms of the accuracy, the introduced model performed better than this method, where the accuracy of the proposed system is equal to (98.75%), but the accuracy of this method is equal to (95.7%). For the incremental rate, the suggested approach is capable of extracting sentiment sentences more than the keyword search approach.

Wang *et al.* [44] developed a new model to improve the training speed, modelling efficiency and robustness of the Deep belief network. Their proposed model incorporates Fuzzy Neural Network and Sparse Deep Belief Network. A Fuzzy Neural Network is implemented for supervised modelling in order to reduce the gradient spread. The sparse deep belief network is deemed a pre-training model to achieve weight parameter-initialisation fast and get feature vectors. The empirical outcomes reveal that their proposed model gives good performance compared to other existing techniques in the literature.

Motivated by the strengths of DL and fuzzy logic techniques in the SA field, in the present work, we develop a new hybrid fuzzy-deep learning approach, that basically integrates the CNN, FFNN deep learning networks with the MFS fuzzy logic system. In the next section, I will briefly explain the concept of CNN, FFNN and MFS.

III. BASIC CONCEPTS

In this section, firstly, we introduce the basic notion of the CNN, especially CNN, with one convolution layer [45], which is recognized as a simplified version. Subsequently, the architecture of the FNN is described in detail [46]. Finally, a general overview of the MFS [47] is introduced in brief for further accurate SA.

A. CONVOLUTIONAL NEURAL NETWORK

The CNN structure was first suggested in 1988 by Fukushima [48], which is one of the most common and effective deep convolutional learning networks. Fukushima has designed the architecture of CNN based on the conventional LeNet approach. In the past, CNN is only used for handwriting recognition and image recognition. At present, this network architecture is also used for text classification tasks (include sentiment analysis). Therefore, The use of CNN in all artificial neural networks presented previously achieved good results in terms of classification rate and execution time accordingly to multiple works from the literature. The secret behind these great successes is the structure of CNN, which is designed to become similar to the cat's visual cortex. Indeed, the cat's visual cortex is composed of a complicated arrangement of neurons. These neurons are responsible for covering small sub-areas of the visual area, named the receptive area. Then, the receptive areas are tiled to detect the overall visual area [49]. Hence, receptive areas are deemed as filters in the CNN deep learning model. In summary, the main aim behind CNNs is to innovate a solution for diminishing the total number of parameters and constructing a deeper neural network with fewer parameters. Fig. 2 depicts the overall structure of CNNs, which comprises of three fundamental layers: convolution layer, pooling layer, and fully connected layer.

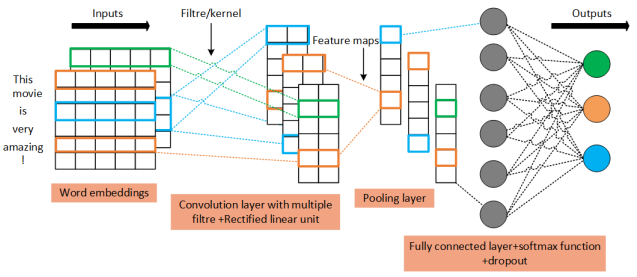


FIGURE 2. Overall structure of the CNN.

As we said previously and as clarified in Fig. 2 CNN architecture consists of three principal layers, unlike classical artificial neural networks. These layers are the Convolution

layer, Pooling layer, and Fully connected layer as described below:

1) CONVOLUTION LAYER

is the essential block in CNN and is always the premier layer in the overall structure of CNN. The major target of this layer is to detect and capture the features from an obtained matrix by applying one of the word embedding methods on the given input sentence. The convolution layer uses a slid filter over the embedding matrix and produces a convolved feature. Multiple filters are applied over the embedding matrix to obtain multiple features maps. These obtained feature maps are activated (that is to say transform the linear feature maps to non-linear feature maps) using the *Rectified Linear Unit* (ReLU) activation function in the intermediate task that linked convolution layer and the pooling layer. Finally, the obtained non-linear feature maps are passed to the pooling layer. In summary, the ReLU is the most popular activation function used with the convolution layer. It merely calculates using the following formula (1) :

$$f(y) = max(0; y) \tag{1}$$

In substance, the activation function ReLU outputs 0 if it gets a negative value as input, and if it gets a positive value as input, the ReLU will output the same positive value [50] as shown in Fig. 3.

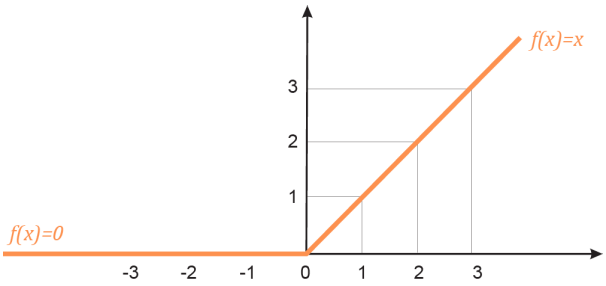


FIGURE 3. Graphic representation of the ReLU activation function.

The advantages of the ReLU function are its ability to overcome the vanishing gradient issue, its convergence is faster due to its simple math formula, and its execution time is relatively short unlike other activation methods such as tanh or sigmoid.

2) POOLING LAYER

After convolving the embedding matrix with multiple filters in the first stage (convolution layer), the second phase is the application of the pooling layer to reduce the dimensionality of obtained feature maps in the first step. Thence the total number of CNN parameters is diminished; the computational cost is decreased, and the overfitting problem is restrained. Two popular pooling functions are average and max pooling operations. The average-pooling method determines the pooling feature as the average of all values in the convolved feature map. The max-pooling method selects the maximum

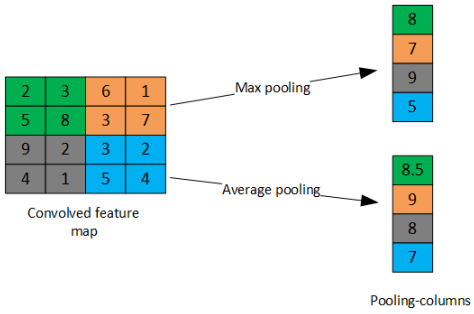


FIGURE 4. Illustration of average-pooling and max-pooling functions.

element in the convolved feature map as a pooling feature and discards the rest. These both operations are described in Fig. 4. In this work, we have applied the max-pooling method. Generally, the pooling layer transforms the convolved feature maps to a single column, which is further passed to a fully connected layer [51].

3) FULLY CONNECTED LAYER

is also termed a dense layer, which used in this work to calculate the sentimental scores of each input sentence (PSS and NSS) from the obtained single column from a pooling layer in the previous phase. We can summarize its functionality as a linear process in which each input is linked to all output by different weight [51]. The fully connected layer calculates the sentimental values using the equation (2) as follows:

$$Sv = f(Wm * Cp + Bi) \tag{2}$$

where Sv is the calculated sentimental value, f is softmax activation method, Wm is the used weight matrix, Cp is the single column obtained from the pooling layer, and Bi is the bias.

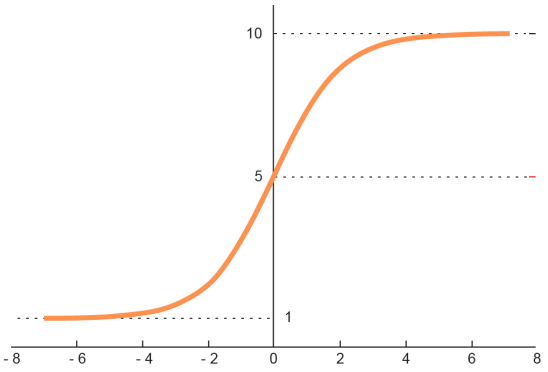


FIGURE 5. Graphic representation of the softmax activation function.

The softmax activation function (or a normalized exponential function) is applied in the intermediate learning process between the fully connected and output layers. This activation function converts the received numerical values from the fully connected layer to probable values, which are in the interval [0,1], and the sum of these probable values be equal to 1, as represented in Fig. 5.

Here, in this research work, we applied the softmax function into the received vector of z real values through the last hidden layer of FFNN to calculate two values: positive sentimental and negative sentimental scores. A softmax activation function is denoted as in equation (3):

$$f(y) = \frac{e_i^y}{\sum_{k=1}^{K} e_i^y} \tag{3}$$

where f is softmax activation method, y is the input value, e_i^y is the standard exponential function of the input value, and K is the number of classes in the given dataset.

B. FEEDFORWARD NEURAL NETWORK

FFNN is an artificial intelligence model, which is broadly used in several applications due to its capability and competence to act like the human brain. The simple version of FFNN includes three layers which are: input layer, hidden layer, and an output layer. The other versions of FFNN differ in the number of hidden layers. FFNN with two hidden layers is illustrated in the Fig. 6. In this artificial neural network, the data is only transferred in one direction, from the input neurons to the hidden neurons and from the hidden neurons to the output neurons nodes. An FFNN can decrease the error for non-linear input nodes and possesses the capability to discover the relationship that connected input nodes and the output nodes without using any complex mathematical theories. The easy application of FFNNs and its flexibility are outstanding advantages compared to other neural networks.

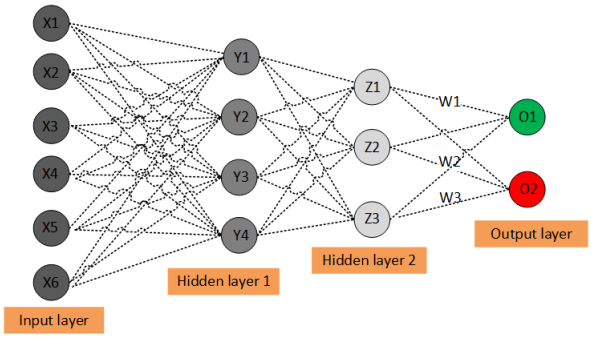


FIGURE 6. The overall architecture of the feedforward neural network.

As Fig. 6 illustrated, the FFNN contains four layers, which are: the input layer, two hidden layers, and the output layer. The former layer of the FFNN is the input layer. It is utilized to provide the input text/image data to the FFNN. This input layer is to be followed by both hidden layers. These hidden layers are utilized to augment the non-linearity and modify the representation of the received data from the input layer for good generalization over the applied activation function. The most widely applied activation method on the hidden layer is the ReLU. The last hidden layer is to be followed by the output layer, which is the latter layer in the FFNN, which outputs the class label predictions. In our work, this layer produces positive and negative sentimental scores of the input sentence. The activation function to be applied in this output layer is different for different issues. In the binary classification issue, we used the sigmoid activation function because we need the output of the layer to be either 0 or 1. For a multiclass classification issue, we applied the softmax activation function. In our work, we want the outcomes of the output layer to be in the interval [0,1]; thus, we have applied the softmax activation function. Generally, each hidden layer in the FFNN contains many nodes called neurons. Each neuron node is related to the input layer and the output layer through the connectors. And every link has multiple weights, which are modifiable. Therefore, the operations carried out at the level of each neuron are described by the schematic diagram shown in Fig. 7.

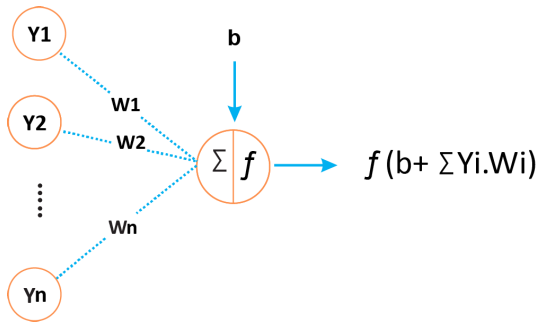


FIGURE 7. An illustration graphic of a neuron depicting the inputs ($y_1 - y_n$) which are the neurons of the former layer, their corresponding weights ($w_1 - W_n$), a bias (b) and the f is the applied activation procedure on the weighted sum of the inputs.

C. MAMDANI FUZZY SYSTEM

MFS is one of the most popular *Fuzzy Inference Systems* (FISs), which are also called fuzzy rule-based systems. The essential objective behind these kinds of systems is the decision-making process. FISs are considered as a novel version of *Classical Rule-Based Models* (CRBMs). Furthermore, in FIS, the variable's value takes a numerical value between 0 and 1. In contrast, the variable's value takes either '0' or '1' in CRBM. Therefore, the FISs serve to convert a black and white decision-making issue into a grey decision-making issue. MFS consists of a fuzzification process, a knowledge base (rule base+ database) unit, a decision-making process, and finally a defuzzification process.

1) FUZZIFICATION PROCESS

which converts the crisp set into the fuzzy set using Triangular, Trapezoidal, or Gaussian MF. In other words, each value in the crisp set is transformed into linguistic value.

2) KNOWLEDGE BASE UNIT

The knowledge base has consisted of two parts; which are a rule base and a database, wherein the rule base includes a collection of IF-THEN fuzzy rules, and the database includes

the parameters information about the employed membership function and a simple definition of the fuzzy set.

3) DECISION-MAKING UNIT OR INFERENCE ENGINE

it carries out the inference steps on the IF-THEN fuzzy rules stocked in the rule base and obtains a fuzzy output.

4) DEFUZZIFICATION PROCESS

which transforms the fuzzy outputs of the inference mechanism into a crisp output using one of these methods Max-membership procedure, Weighted average approach, Centroid technique, Mean-max membership, First of maxima or last of maxima method, Centre of largest area, and Centre of sums. The overall architecture of a FIS is illustrated in Fig. 8.

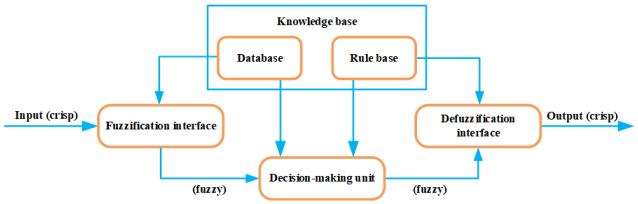


FIGURE 8. The components of the fuzzy inference systems.

In the literature, there are three well-known FISs, which are Mamdani, Sugeno, and Tsukamoto. Both Tsukamoto and Sugeno systems are applied in the case of regression issues, unlike the Mamdani, which is utilized in the case of system classification problems [28]. The major distinction between the MFS and the Sugeno fuzzy system lies in the manner that each system defines the consequent block of its fuzzy If-Then rules. Mamdani model employs fuzzy sets as a consequent block of the fuzzy If-Then rule. At the same time, the Sugeno model employs a linear equation as a consequent block of the fuzzy If-Then rule. Basically, the primary goal of our work is the resolution of a sentiment classification problem; for that, we have used the Mamdani fuzzy method as a fuzzy classifier. The foundation of the MFS is introduced as crisp output elements which are deduced from crisp input elements using a collection of fuzzy If-Then rules stocked in the fuzzy rule base and passing through the fuzzification and defuzzification processes; Therefore, in this work, to calculate the crisp output (class label of the sentiment sentence) of this MFS giving consideration to the crisp inputs, we have followed six steps as described below:

1 Defining a set of If-Then fuzzy rules.
2 Fuzzifying the crisp input variables by applying one of the membership functions, which are triangular, trapezoidal, or gaussian membership function.
3 Integrating the fuzzified input variables based on the fuzzy If-Then rules in order to create a If-Then rule strength.
4 Determining the consequence of the rule by integrating the outcome of the applied membership function and the created rule strength in the previous step.
5 Integrating all consequences obtained in step 4 to acquire an output distribution.
6 Applying the defuzzification function on the output distribution to get the crisp output.

D. PERFORMANCE METRICS

To evaluate the text classification process, we mainly calculate ten performance metrics: *True Positive Rate* (TPR), *True Negative Rate* (TNR) or Specificity, *False Positive Rate* (FPR), *False Negative Rate* (FNR), *Error Rate* (ER), *Precision* (PR), Classification Rate or *Accuracy* (AC), *Kappa Statistic* (KS), *F1-score* (FS) and *Time Consumption* (TC). These performance metrics are calculated using the confusion matrix for binary or multi-class classification as given in Fig. 9 and 10.

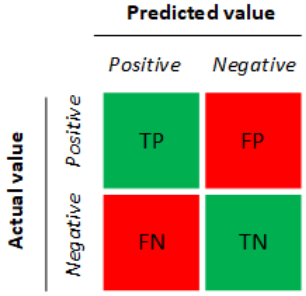


FIGURE 9. Confusion matrix for a binary classification issue.

The abbreviations *False Negative* (FN), *True Positive* (TP), *True Negative* (TN), and *False Positive* (FP) in the simple confusion matrix for binary classification in Fig. 10 are defined as follows [9], [28]:

. **True Positive (TP)**: Number of instances that are actually positive and predicted to be positive
. **False Negative (FN)**: Number of instances that are actually positive and predicted to be negative
. **True Negative (TN)**: Number of instances that are actually negative and predicted to be negative
. **False Positive (FP)**: Number of instances that are actually negative and predicted to be positive

Recall, Specificity, False Positive Rate, False Negative Rate, Error Rate, Precision Rate, Accuracy, Kappa Statistic and F1-score evaluation metrics are calculated in the case of binary classification using the confusion matrix described in Fig. 10 as follows:

Recall measures the performance and efficiency of a classifier to predict the number of instances that have a positive class label. This metric is calculated by using (4), where *tp* is the number of predicted positive instances, and *tp*+*fn* is the total number of positive instances in the dataset.

$$Recall = \frac{tp}{tp + fn} \tag{4}$$

Specificity measures how a classifier is an efficacy to identify the number of instances that have negative class labels. This metric is computed by using (5). Where *tn* corresponds

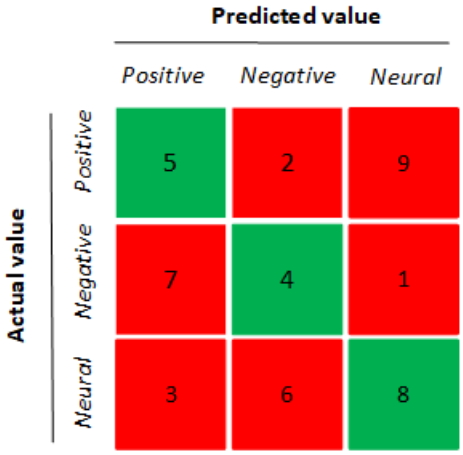


FIGURE 10. Confusion matrix for a multi-class classification issue.

to the number of samples that have negative class labels, and *tn*+*fp* is the total number of instances that are negative class labels in the used dataset.

$$Specificity = \frac{tn}{tn + fp} \tag{5}$$

False Positive Rate is the rate to detect the inefficiency and ineffectiveness of a classifier and to measure the misclassification rate by calculating the number of instances, which are actually negatives but the classifier predicted it to be positives. False Positive Rate is computed by using (6). Where *fp* corresponds to the number of instances which are actually negatives but the classifier classified it as positives, and *fp*+*tn* is the total number of negative instances.

$$FalsePositiveRate = \frac{fp}{fp + tn} \tag{6}$$

False Negative Rate is the rate to detect the inefficiency and ineffectiveness of a classifier and to measure the misclassification rate by calculating the number of instances, which are actually positives but the classifier predicted it to be negatives. This evaluation metric is calculated by using (7). Where *fn* corresponds to the number of instances which are actually positives but the classifier classified it as negatives, and *fp*+*tn* is the total number of positive instances.

$$FalseNegativeRate = \frac{fn}{fn + tp} \tag{7}$$

Error rate metric serves to measure the misclassification rate, that is to say, this metric computes the number of misclassification instances over all instances in the used dataset. Basically, its objective is to measure the classifier's ability to restrain false classification. The error rate is defined as (8) presents. Where *fp*+*fn* corresponds to the total number of incorrectly classified instances, and *tp*+*fn* +*tn*+*fp* is the total number of instances in the given dataset.

$$Error = \frac{fp + fn}{tp + fn + tn + fp} \tag{8}$$

Precision performance metric measures how many samples retrieved as positive class labels are, in fact, positives. The precision rate is useful for assessing brittle classifiers, which are applied to classify all instances of the used dataset. This evaluation metric is determined as (9) describes. Where *tp* corresponds to the number of instances which are actually positives and the classifier classified it as positives, and *tp*+*fp* is the total number of instances that are predicted as positive.

$$Precision = \frac{tp}{tp + fp} \tag{9}$$

Accuracy rate is an overall metric for estimating the effectiveness, and correctness of learning classifiers. The accuracy is computed utilizing a test set that is detached from the training set. This rate is measured using equation (10). Where *tp*+*tn* is all true classifier examples and *tp*+*fn* +*tn*+*fp* is the overall instances in the used dataset.

$$Accuracy = \frac{tp + tn}{tp + fn + tn + fp} \tag{10}$$

F1-Score or F-measure is the harmonic mean between precision rate and recall rate gives a good idea about the average. The value of F1-Score is ranged from 0 to 1. it measures how the used classifier is accurate and robust. If F1-Score increases the performance of the used classifier will be better. In other words, to get an extremely accurate performance, the precision must be higher, and the recall must be lower. This metric is calculated by using (11). Where Precision is computed using the formula (9), and Recall is calculated using the formula (4).

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{11}$$

Kappa statistic is a performance criterion that compares an observed accuracy and an expected accuracy (random chance). It is used not only to estimate one classifier but also to inspect classifiers amongst themselves. The kappa statistic is computed by utilizing (12).

$$\text{Kappa-Statistic} = \frac{P_0 - P_e}{1 - P_e} \tag{12}$$

where: $P_0 = \frac{tp+tn}{100}$; and $P_e = [\frac{tp+fn}{100} * \frac{tp+fp}{100}] + [\frac{fp+tn}{100} * \frac{fn+tn}{100}]$.

Recall, Specificity, False Positive Rate, False Negative Rate, Error Rate, Precision Rate, Accuracy, Kappa Statistic and F1-score performance metrics are computed in the case of multi-class classification using the confusion matrix illustrated in Fig. 10. The first step to calculate these metrics is to compute the measurements TN, FN, TP, FP, as described in Fig. 9 for each class in the multi-class confusion matrix. For example, if we take the class Positive the values of these measurements are determined as follows: TP=5; TN=(4+1+6+8)=19; FN=(7+3)= 10; FP=(2+9)=11. In the case of Negative class these metrics will be TP=4; TN=(5+9+3+8) =25; FN=(2+6)=8; FP=(1+7)=8. and in the case of Neural class these measurements are computed as follows: TP=8; TN=(5+2+ 4 +7)=18; FN=(3+6)=9; FP=(1+9)=10 after the calculation of these measurements, we compute the previously evaluation metrics as follows.

Computation of the recall for multi-class classification is made according to (13). Where tp_i is the number of predicted instances are labeled class i,i is the index of the class, l is the total number of class labels, and $tp_i + fn_i$ is the total number of instances labeled class label i in the given dataset.

$$Recall = \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i+fn_i}}{l} \tag{13}$$

Specificity metric is measured for multi-class classification using (14). Where tn_i is the number of predicted instances which are not labeled class i,i is the index of the class, l is the total number of class labels, and $tn_i + fp_i$ is the total number of instances are not labeled class i in the given dataset.

$$Specificity = \frac{\sum_{i=1}^{l} \frac{tn_i}{tn_i+fp_i}}{l} \tag{14}$$

False positive rate measure is calculated as described in (15). Where fp_i is the number of instances which are not actually labeled class i but the classifier predicted it to be class label i,i is the index of the class, l is the total number of class labels, and $tn_i + fp_i$ is the total number of instances are not labeled class i in the used dataset.

$$FalsePositiveRate = \frac{\sum_{i=1}^{l} \frac{fp_i}{fp_i+tn_i}}{l} \tag{15}$$

False negative rate evaluation criterion is computed by using (16). Where fn_i is the number of instances which are actually labeled class i but the classifier predicted it not to be class label i,i is the index of the class, l is the total number of class labels, and $fn_i + tp_i$ is the total number of instances are actually labeled class i in the used dataset.

$$FalseNegativeRate = \frac{\sum_{i=1}^{l} \frac{fn_i}{fn_i+tp_i}}{l} \tag{16}$$

Error rate for multi-class classification is measured by employing (17). Where $fp_i + fn_i$ is the number of all instances which are predicted incorrectly,i is the index of the class, l is the total number of class labels, and $fn_i + tp_i + fp_i + tn_i$ is the total number of instances in the given dataset.

$$Error = \frac{\sum_{i=1}^{l} \frac{fp_i+fn_i}{tp_i+fn_i+tn_i+fp_i}}{l} \tag{17}$$

Precision measure is calculated in the case of multi-class classification as illustrated in (18). Where tp_i is the number of instances which are actually labeled the class i and predicted by the used classifier correctly,i is the index of the class, l is the total number of class labels, and $tp_i + fp_i$ is the total number of instances labeled the class i in the given dataset.

$$Precision = \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i+fp_i}}{l} \tag{18}$$

The accuracy measure in the case of multi-class classification is calculated according to (19). Where $tp_i + tn_i$ is the number of all instances which are predicted correctly,i is the index of the class, l is the total number of class labels, and $fn_i + tp_i + fp_i + tn_i$ is the total number of instances in the given dataset.

$$Accuracy = \frac{\sum_{i=1}^{l} \frac{tp_i+tn_i}{tp_i+fn_i+tn_i+fp_i}}{l} \tag{19}$$

Another metric, F1-Score, is employed to integrate the precision and recall rates in a single measure. The value of this metric is ranged from 0 to 1 as we present previously, and if the evaluated classifier properly classifies all instances, this metric will take the value 1. The F1-Score is computed by applying (20) for multi-class classification. Where $Precision_i$ is computed using the formula (18), and $Recall_i$ is calculated using the formula (13).

$$F1 - score = \frac{\sum_{i=1}^{l} \frac{2*Precision_i*Recall_i}{Precision_i+Recall_i}}{l} \tag{20}$$

Kappa statistic is computed utilizing equation (21)in the case of multi-class classification.

$$\text{Kappa-Statistic} = \frac{\sum_{i=1}^{l} \frac{P_{0i}-P_{ei}}{1-P_{ei}}}{l} \tag{21}$$

where: $P_0 = \frac{tp+tn}{100}$, and $P_e = [\frac{tp+fn}{100} * \frac{tp+fp}{100}] + [\frac{fp+tn}{100} * \frac{fn+tn}{100}]$.

IV. METHODOLOGY OF OUR PROPOSED APPROACH

In the subsequent sections, we will discuss the motivations that pushed us to develop this work proposal. The basic architecture of our suggested hybrid model is composed of the data collection phase, text pre-processing steps for reducing the noisy data, word embedding methods for transforming the text-based data into numerical-based data, CNN for extracting the features automatically, FFNN for calculating both PSS and NSS values, and MFS for classifying their input into negative or positive or neutral class.

A. MOTIVATION

As mentioned in the introduction section, our proposal endeavors to improve SA effectiveness. The primary goal of this contribution is to classify each sentence in the used dataset into a positive or negative or neutral class with high-performance and efficiency in terms of ten evaluation criteria, which are presented in the previous section, and also in terms of convergence, stability, and complexity.

In the literature, multiple categories of approaches have been applied to perform the sentiment classification. Among these techniques, we find DL models, ML methods, and dictionary-based procedures. The performance of ML and dictionary-based approaches is lower than DL models if we applied them on the enormous dataset. The studies demonstrate that classical ML methods and dictionary-based techniques are better for a lesser size of data. As the volume of data rises beyond a particular number, the accuracy of classical ML algorithms becomes constant. In contrast, the accuracy of DL algorithms raises concerning the raise in data size. This difference in performance is due to the used manner for extracting the features from the dataset. Conventional ML techniques and dictionary-based approaches adopt

hand-crafted engineering features by applying feature extractor approaches. They then apply the learning algorithms, which extract incomplete features and take high time to produce the final result. Unlike DL models that adopt automatic features extractor.

According to these studies that showed DL techniques' strengths on the massive dataset, in this work, we have been applied CNN deep learning model as an automatic feature extractor. Because CNN possesses the higher power to detect and extract relevant features at different local levels identical to a human brain and compared to the conventional deep learning models that cannot do the feature learning, another advantage of CNN is weight sharing, which makes CNN more accurate and efficient in terms of complexity and used memory than traditional neural networks. Also, the CNN is characterized by the optimized structure for handling text/image, the ability to extract the abstraction features, the absorption of shape variations by the application of the pooling layer, The number of parameters is fewer, and the lower diminishing gradient rate compared to the conventional neural network.

We also used FFNN to employ the outputs of CNN and calculates two sentimental values: PSS and NSS sentimental scores. The PSS presents the percentage of the existing positive opinion words in the given sentimental sentence, and the NSS presents the percentage of existing negative opinion words in the same sentence. Basically, FFNN is one of the most effective neural networks presently preferred in regression and classification tasks. FFNN be composed of an input fully connected layer, one or more hidden layers, and an output layer. Consequently, the changed number of hidden layers aids in processing more and more complicated functions. As presented previously in the basic concepts section, this type of network can process data by itself and generates outputs that are not restricted to the input variables provided to it.

Furthermore, it can carry out multiple operations in a parallel manner without any negative influences on the network model performance. Also, dropout regularization is applied at the level of the fully connected layer to overcome the network overfitting and enhance the generalization error. In summary, we have incorporated CNN and FFNN as the third step of our work to handle the collected unstructured data from social media networks and compute both values PSS and NSS as outputs of our deep learning model (CNN+FFNN).

As a result of social-media data holding considerable noise and unpredictable vagueness, the notion of ambiguity and uncertainty data elicits the attention of many researchers. Such vagueness assesses a big challenge on the capability to implement and to classify social-media data. First, the ability to symbolize input social-media data is restricted as variables react uncertainly. Second, CNN and FFNN deep learning models are not always powerful when training social-media data are irritated by the noise. The fuzzy logic theory has been applied to overcome deep learning shortcomings and enhance sentiment classification performance. Compared with classical logic representations, fuzzy logic representation builds a set of IF-THEN fuzzy rules for eliminating the uncertainties in social-media data and achieves higher accuracy in both data symbolization and hardiness for handling the noisy data. Motivated by the fuzzy logic theory's strengths, in the fourth step of our work, we have been used MFS as a fuzzy classifier. Simultaneously, the input variables of MFS are the PSS and NSS values, and the output variable is the class label (Positive, Neutral, Negative).

As a short conclusion, this work's essence is to increase the classification effectiveness of sentiment analysis by integrating the power of MFS to deal with uncertainty and vagueness data and the power of both deep learning models CNN and FFNN to detect and capture the features automatically from the given dataset. As depicted in Fig. 11, the overall structure of our developed hybrid model consists of six phases, which are Data collection, Data pre-processing, Word embeddings, CNN, FFNN, and MFS.

B. PHASE I: DATA COLLECTION

In this paper, we have been chosen two datasets to prove the performance of our develop FDLC approach. The first dataset called **sentiment140** dataset. Which is extracted using twitter application programming interfaces (API). It consists of 1,600,000 tweets in which the emoticons were removed. The tweets have been labeled into two class negative and positive, where (0 = negative, and 4 = positive). It includes the following six attributes:

- **Target**: is the sentimental score of the tweet (4 = positive, 0 = negative)
- **Ids**: is the identifier of the tweet (1467110309)
- **Date**: is the date when the tweet is posted (thr Mar 06 21:18:55 PDT 2007)
- **Flag**: represent The query (text of the query). If there is no query, then this attribute takes the NO_QUERY' value.
- **User**: is the username that tweeted (LionsLamb)
- **Text**: is the text posted by the user with the name LionsLamb (He's the reason for the teardrops on my guitar, the only one who has enough of me to break my heart)

In this work, we are interested in sentiment analysis. That is to say, extract the sentiment expressed by the author in the tweeted text. Thence the other attributes have not any influence on the learning process. For that, we removed the Ids, User, Flag, and Date attributes from the dataset. And we kept the Text and Target attributes. The Target distribution of the data in this dataset is balanced distribution, such as 50% of the tweets are labeled negative, which are ranged from 0 to 799999th index, and another 50% of the tweets are labeled positive, which are ranged from 800000 to 1 600 000th index. The dataset is split into testing and training subsets. Consequently, we have been used these subsets to prove the classification performance of our designed FDLC compared to other proposed methods which are selected from the literature. Fig. 12 introduces the number of tweets in every subset. Where a total of 1,440,000 tweets were utilized in the

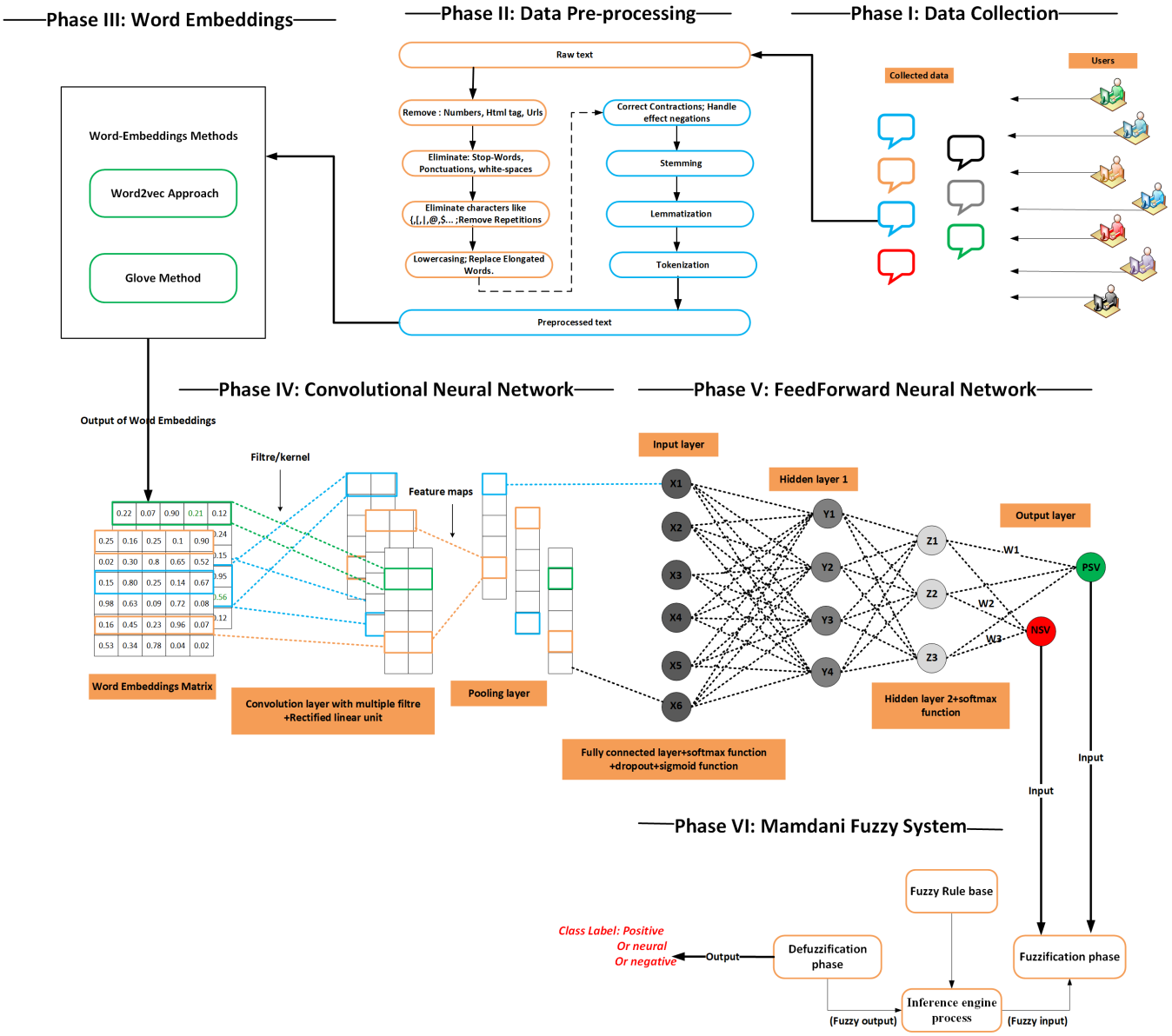


FIGURE 11. Architecture of our proposed approach.

training learning process, and 160,000 tweets were utilized in the testing learning process.

The second dataset, called **COVID-19_Sentiments**, it is also extracted using the twitter API. It consists of 637978 tweets. The tweets have been labeled into three class negative [−1,0[, neutral = 0, and positive]0,1] [52]. it contains the following attributes:

- **Target**: the sentimental score of the tweet (negative [−1,0[, neutral = 0, and positive]0,1]) [52].
- **Ids**: The identifier of the tweet (1241032866567350000)
- **Date**: the date when the tweet is posted (Sun May 31 04:52:40 +0000 2020)
- **Location**: The location where the tweet is posted (Ahmadabad City, India)
- **Text**: the text posted by the users.

The important attributes in our work are the text and sentimental score attributes. For that, we removed all other attributes. In addition, the Target distribution of the data in this dataset is an unbalanced distribution with 259458 neutral tweets, 120646 negative tweets, and 257874 positive tweets. Fig. 13 depicts the number of tweets in both testing and training subset. A total number of 574,182 tweets were employed in the training learning process, and 63,796 tweets were used in the testing learning process. In other words, the testing dataset represents 10% of the overall dataset.

C. PHASE II: DATA PRE-PROCESSING

Pre-processing tasks are deemed to be the first phase in the text classification task, and picking out the right, and

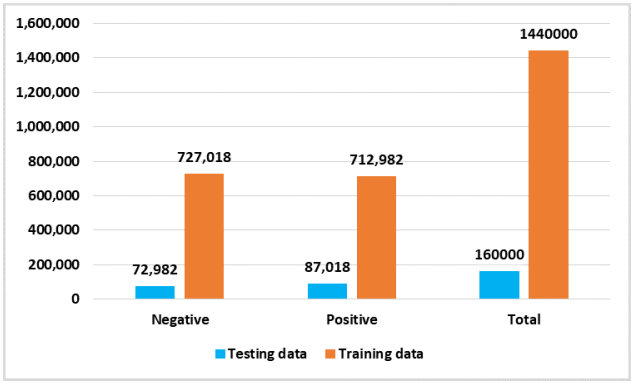


FIGURE 12. Number of negative and positive tweets for the sentiment140 dataset.

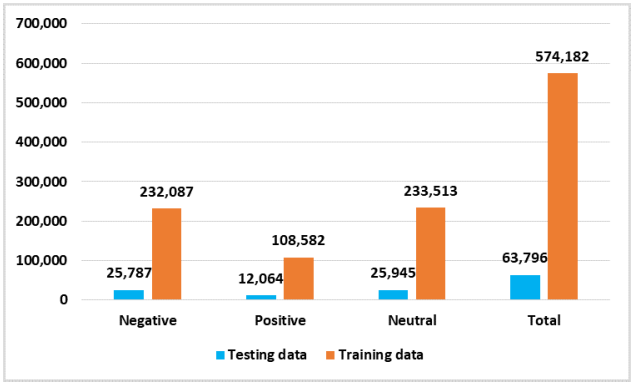


FIGURE 13. Number of negative, neutral, and positive tweets for the COVID-19_Sentiments dataset.

effective pre-processing techniques can enhance classification performance. The primary goal of the text pre-processing procedure is preparing, normalizing, removing, and cleaning the noisy data from the given dataset that is going to be classified. The noisy data is the data without any valuable information for sentiment classification. The pre-processing technique transforms the noisy data from high dimensional attributes to the low dimensional features to get as much accurate useful data as possible from the given dataset. The text pre-processing phase can consist of multiple techniques depending on the text-classification issue and the situation. In this work, our classification issue is the sentiment classification of data collected from twitter. Twitter allows its users to post only messages with 140 characters. Due to this restricted rule, Twitter users have been employed the slang, abbreviations, exclamation marks, links, repetitions, punctuation signs to assert their attitudes and emotions in a short tweet. In addition, Twitter users are vulnerable to spelling and typographical mistakes. It is not essential to include all expressions of the tweet in the learning process in our work, and multiple of them should be deleted, normalized, cleansed, or replaced with others. Thus, it emerges the need to apply the pre-processing techniques to the given data. Their free of the noise is a crucial factor to increase the sentiment classification effectiveness. The followed up pre-processing methods in this work are described below:

Remove number, URLs, hashtags, and username: It is a popular tactic to eliminate numbers, URLs, hashtags, and usernames from the pre-processing sentence because they do not hold any emotions.

Eliminate punctuation,white-spaces, and special characters: The first step to do is removed all existed white-spaces in the tweet, followed up by removing the three punctuation, which are the stop, question, and exclamation marks. All found special characters are removed because they do not have any positive or negative impact on expressed sentiment. after all these pre-processing techniques presented previously, we kept only the lowercase and uppercase letters.

Lower-casing: From the previously described steps, all special characters other than letters have been deleted. So the next step is the lower casing. In other words, all the letters kept in the tweet were transformed to lower case, which reduce the dimensionality of words.

Replace elongated words: This operation serves to remove the letter, which is repeating at least more than three times in the elongated word like the word ''haaaaaaappy''. after applying this operation, the word becomes ''haappy'' and normalized with at most two characters.

Remove stop-words: Stop-words are the words with high occurrences in the posted tweet. They are removed because they do not hold any emotions, and it is deemed needless to handle them. Therefore in our work, all found stops-words in the tweet are removed based on the stop-words list determined by the NLTK package in Python.

Correct contractions: One tactic that can be employed in the pre-processing procedure is the correction of contractions. For example, the words like 'isn't' its corrected word will be 'is not', and 'weren't' its corrected word will be 'were not'.

Handle effect negations: This approach replaces the word preceded by NOT by its antonym. The antonym means the opposite meaning of the replaced word. The process of this approach serves to search in each tweet the word preceded by NOT, then to check if this word has an antonym in WordNet dictionary if the case, it replaces this word with its unambiguous antonym. For example, it replaces the word ''not uglify'' with ''beautify''.

Stemming: is the operation of reducing the size of words by merging several words into one. This approach deletes the endings of words to discover their word stem in a dictionary. In this work, we used the Porter Stemmer of NLTK package in Python.

Lemmatization: has the same role as Stemming. It is also another approach that serves to determine the root forms of words. The difference between both procedures is in the followed process to detect the stem or lemma words.

Tokenization: is a process that splits sentences into words called tokens. In its process, larger paragraphs of analysis data can be split into sentences. Then these sentences obtained can also be split into tokens. In this work, we used an NLTK tokenizer provided by Python.

All presented techniques previously are applied to the used dataset in this work. Besides, we construct a lookup table with 3000 words and phrases containing the words, abbreviations, and slang words to replace the abbreviation and slang words in the currently processed tweet with correct words. For example, we find these slang and abbreviation words "*ab/abt*", "*B*" and "*B4*", which respectively denote and replace by "*about*", "*be*", and "*before*". In the twitter platform, users are prone to spelling and typographical mistakes that might make the learning procedure harder. Therefore, for improving the learning process effectiveness, we used Norvig's spelling and typographical corrector, which automatically corrects them.

We carried out multiple experiments to demonstrates the effectiveness of the applied pre-processing techniques on our dataset. From the experimental results, as shown in Tables 1 and 2, we deduced that the pre-processing techniques decrease the error rate. Where the error rate in the Sentiment140 dataset decrease from 35.59% to 5.98%, and it decrease from 29.04% to 3.61% in the COVID-19 Sentiments dataset. Therefore, it is recommended to use pre-processing techniques.

TABLE 1. Error Rate (ER%) without pre-processing techniques.

Name of dataset	ER without text pre-processing
Sentiment140	35.59
COVID-19 Sentiments	29.04

TABLE 2. Error Rate (ER%) with pre-processing techniques.

Name of dataset	ER with text pre-processing
Sentiment140	5.98
COVID-19 Sentiments	3.61

After the pre-processing step that serves to remove noisy data from the used dataset, The next step is word embeddings, as described in the following subsection. In other words, then consequently, data from the application of all pre-processing techniques will be the input of word embedding methods.

D. PHASE III: WORD EMBEDDINGS

CNN deep learning model can only process the numerical data. Therefore, to make our proposal deals with text-based data obtained after the pre-processing data phase using deep learning models, these text-based data must be transformed into numerical-based data. This operation is called vectorization, which is one of the critical issues in NLP. Approaches such as word2vec, glove [15], tf-idf [16], bag-of-words [17], fast-text, n-gram or character-level [18] are the major of word vectorization techniques. The most effective methods in the case of the larger datasets are GloVe, Word2vec, and Fast-text, which are introduced by Stanford, Google, and Facebook respectively [53], [54]. Therefore–in this work–after the pre-processing data phase, the next stage is the word embeddings data using Word2vec and GloVe and Fast-text techniques. This section presents these three different kinds of word embeddings methods in detail.

1) GloVe

Global Vectors for Word Vectorization or *GlobalVectors* (GloVe) was introduced by Jeffrey Pennington *et al.* [54], and was supported by Stanford University. This learning model is an unsupervised algorithm. Its objective is computing the vector representation for distributed words. This operation is made by finding the semantic similarity between words, then generating the word-word co-occurrence count matrix. e.g., how frequently these words seem together in the corpus. For that, the GloVe was named the count-based model. Word embedding of this model is obtained by aggregating the created co-occurrence count matrix from a corpus, and the resulting word embeddings show for each word in vector space important linear substructures. In summary, This model integrates both methods, which are the local context window model and the global matrix factorization method. Experimentally, GloVe gives good results on word similarity, named entity recognition, and word analogy tasks compared to word2vec and Fast-text.

2) Word2Vec

Word2Vec was proposed by Tomas Mikolov *et al.* at Google [55]. It employs the FFNNs with one hidden layer to extract the word embeddings vector from the inputted text/image data. This method integrates the Skip-Gram model, Which predicts the current surrounding context words based on target words, and the *Continuous Bag-of-Words* (CBOW) model, Which predicts the current target words based on the surrounding context words. Because of that, it was named two-layer neural networks. Its objective is to enhance the predictive ability for word vectorization. This Word2vec takes a large corpus of text/image-based data as inputs and generates a matrix as an output. Where each row in the created matrix represents the vector with several hundred dimensions. This produced vector is the word vectorization of the one-hot vector of the input token (word or character). In summary, Word2Vec is a simple FFNN with one hidden layer. During the learning process, its main goal is to adjust their weights for minimizing the error rate by decreasing the loss function. These hidden weights are used as the word embeddings. Word2vec gives better performance in sentiment polarities prediction, and its performance is better on massive datasets.

3) FAST-TEXT

Fast-text is another word embedding techniques created by the Facebook AI Research Team for effective learning of word vectorization. This method is deemed as an extension of the Word2vec method; instead of training a set of tokens (words) directly as in the Word2vec method, Fast-text trains each token as an n-gram of characters. For example, the representation of the word '*fuzzy*' using the Fast-text

method with n-gram=2 is (*f, fu, uz, zz, zy, y*), where the brackets denote the beginning and end of the represented word. This allows to detect the sense of shorter words and helps the embeddings to learn the suffixes and prefixes of the word. So, once the inputted token has been divided by applying the character n-grams, either skip-gram or CBOW is used to learn the word embeddings. Fast-text works well with unseen words and the words out-of-vocabulary. So, even if the word is unseen in the previous training steps, this method is broken down it into n-gram characters to compute its embeddings. Word2vec and GloVe methods both fail to get vector embedding for the unseen words, unlike Fast-text that can learn the unseen words. This is the strong point of this method compared to other techniques.

As we said previously, the next step of our work after the data pre-processing step is the data vectorization using Word2vec, GloVe, or Fast-text techniques. In the data-processing process, each input sentence is split into a set of words (Word2vec, and GloVe) or n-gram characters (Fast Test). Subsequently, the word embedding methods take this set of words or n-gram characters as its inputs. These methods take as its inputs the one-hot vectors that represent the sentence's words or n-gram characters. We symbolize a sentence as $S= [W_{v1}; W_{v2}; \ldots.; W_{vn}]$, where W_{vi} is the word or character vector, which is the one-hot $vector_i$ represent the $word_i$ or $character_i$. the the one-hot vector' dimension is equal to the number of characters or words in the pre-processing sentence. In this case equals to N. a pre-trained method applied its weight matrix (W_m) to matrix S and obtained low-dimensional matrix representation $M_r= [x_1; x_2; \ldots.; x_n]$ with $x_i \in R^m$.This operation can be written as follows:

$$M_r = W_m.S \tag{22}$$

where $W_m \in R^{m.n}$ indicates the weight matrix, $M_r \in R^{m.n}$ symbolizes the low-dimensional matrix representation of a sentence, and S is the one-hot matrix.

After the Word embedding phase that aims to transform the text-based data into numerical data. It takes a pre-processing text as inputs, and it outputs an embedding matrix. The next stage is the application of CNN, as described in the following subsection.

E. PHASE IV:CONVOLUTION NEURAL NETWORK

After the Word embedding phase, our proposed system is trained to employ the CNN deep learning model, which be formed by four layers.

The first layer is called **Embedding layer or input layer**, which demands word embedding as an input, i.e., a set of vector representations of the learned sentence as explained in the previous section, where each vector $v_w \in R^{1*di}$ represent either character or word accordingly to the used word embedding method. Where d_i is the vector dimension, and it must be inferior to the size of vocabulary in the embedding dictionary. In our work, Word2vec, GloVe, and Fast-text have been used, which are eligible to discover the semantic and syntactic properties of characters and words in the used dataset. In the previous experiment, which is carried out with the used word embedding methods and Hadoop framework. These parallelized word embedding methods have been pre-trained on 90% of the used dataset. After this operation, we get a pre-trained model employed to map each word or character onto its own vector representation. We have then computed the error rate of these word embedding methods by employing (17). The computed error rate indicates that the Fast-text is the most efficient word embedding. Accordingly to these experimental results, we will use the Fast-text word embedding method in the rest of this work. So, the high-dimensional vectors set are computed for every n-gram character by computing softmax probability for every n-gram character by using (3). The produced vector representation dimension is equal to the number of hidden neuron nodes in the Skip-Gram hidden layer. The number of hidden neuron nodes has been set to 200. Each tweet is padded with a vector of zeros. The padding aims to guarantee that all the tweets in the used dataset have the same dimension. All the obtained vectors representation is the rows of the embedding matrix E_m consisting of all n-gram character in the dictionary D. These n-gram characters are noted into indices $1...|D|$ to speedily lookup the vector representation of the n-gram character in E_m Then, for each tweet with t n-gram characters, a embedding matrix $M = V_{nc1}; V_{nc2}; \ldots; V_{nci}; \ldots; V_{nc|t|}$ has been built. Where V_{nci} is the vector representation of the *ith* n-gram character.Therefore, M is passed to the convolutional layer.

The second layer is called **Convolutional layer**, which is applied to the word embedding matrix M obtained in the previous layer. In other words, each convolution operation comprises one filter matrix (F), which is applied to every n-gram character's window (CW) in the word embedding matrix M, and one feature map is generated as an outcome. Therefore, the convolutional layer consists of many convolution operations. Thus several filters with ranging window sizes are applied to M, and a set of feature maps is created. we have the $CW = [x_1; x_2; \ldots; x_n]$ with $x_i \in R^m$, a feature k_i is produced from a CW which its size is $Xi : i + v - 1$ by using the following formula:

$$k_i = ReLU(F_i.X_{i:i+l-1} + b) \tag{23}$$

where ReLU is a non-linear activation method as described in (1); $b \in R$ is the used bais and l is the length of the used filter F. Therefore, a feature map $= [k_0, k_1, \ldots, k_{i+l-1}]$ is produced by the application of (23) in all possible CW of the word embedding matrix M. Multiple filters $F_{i:1\to h}$ are applied to produce multiple feature maps $FM_{j:1\to h}$. Fig. 14 illustrates a simple example of applying a filter f into the n-gram character of the word "*Fuzzy*" to compute the feature map based on (23).

As a summarize, the convolution layer takes an embedding matrix M as input and produces a set of feature maps as output. It is known for its efficiency and capability to extract local features automatically. Then, the third layer is **max-pooling layer** which is applied over every obtained

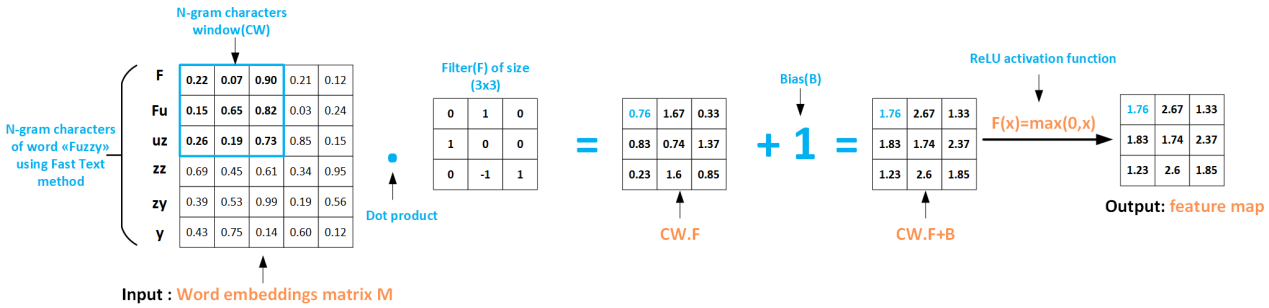


FIGURE 14. Example of the application of one filter in the n-gram character of the word fuzzy.

feature map in $FM_{j:1\rightarrow h}$ and extracts the maximum value of the feature map $pv = max[k_i]$. In this layer, the number of its output (L) will be similar to the number of its input features maps(L). Accordingly to the max-pooling layer, the size of each dimension of the input features maps will be miniaturized, and the outputs will be a set of columns, where the number of these columns equal to the number of inputted features maps. The miniaturization applied by the max-pooling operation is depending on the dimension size of the max-pooling kernel. Fig. 15 presents an example of a max-pooling operation where we used 1×3 as the max-pooling kernel.

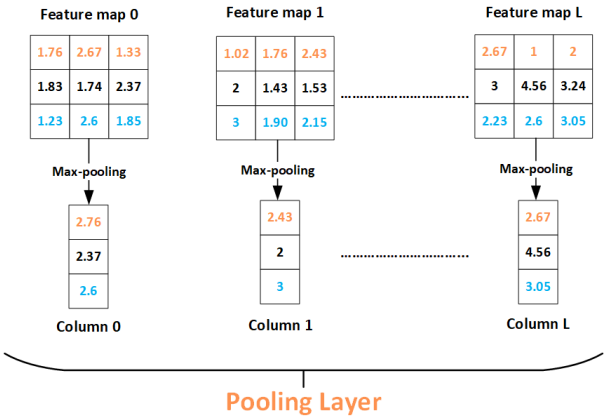


FIGURE 15. Example of the application of pooling layer.

From Fig. 15 we note that a single column of value is obtained as a result of the single max-pooling operation. Therefore, this operation aims to find and save the essential optimum feature by aggregating the data and diminishing the representation size. Finally, the fourth layer is the **fully connected layer**, which is the commune point between CNN and FFNN. At the same time, the fully connected layer is deemed as the outcome of CNN and as the income of FFNN. Furthermore, The value of each fully connected neuron is calculated in the CNN phase using the following formula (24).

$$V_n = f(W_{connector} * C_{pooling} + B) \tag{24}$$

where V_n is the computed neuron value, f is the ReLU activation method, $W_{connector}$ is the weights of the connectors that rely on pooling layer with the fully connected layer,$C_{pooling}$ is the set of the pooled column (or pooled feature maps) in the preceding layer (Pooling layer), and B is the used bias. The following algorithm (1) summarizes all CNN's steps.

The CNN deep learning model's success is due to the three factors: sparse connectivity, weight shared, and equivariant representation. So, CNN different from the classical neural networks, where the connection between the input and output neuron is determined by multiplied the neuron value into the connector weight plus the bias value, which causes the computation burden. While CNN avoids this type of computation burden based on sparse connectivity, i.e., the kernels' size is reduced to be smaller than the dimension of the inputs by using the pooling layer. These kernels are considered in the rest learning CNN process as the whole inputted text/image. The weight shared parameter allows raising the learning efficiency by diminishing the number of weights parameters being learned. The main idea behind this operation is that, in state of learning multiple set of weights parameters at each neuron as in the classical neural network, CNN learn only one set of them, which performs a good performance in terms of classification rate and consumption time. The weight shared parameters have also given the CNN deep learning model, a new property named equivariant representation. i.e., if the input alters, the output alters in an automatic manner and follows the same way as the input changed. Thanks to these three factors, CNN requires fewer weight parameters than other neural network models, which minimizes the used size memory and improves CNN efficiency.

Generally, our proposed deep learning model (CNN+ FFNN) is divided into two parts; the first part is applying CNN to word embedding matrix M obtained in the previous word embedding phase to capture and extract the most important features. In the second phase, we use an FFNN deep learning network to calculate PSS and NSS. The FFNN receives as inputs the outputs of the CNN and generates both values PSS and NSS. The following subsection introduces in detail the FFNN applied in this work.

Algorithm 1: Our Convolutional Neural Network

Input : A given word embedding Matrix M described by **R** rows and **C** columns, and the SF is the set of filter with varying size.
Output: Set of features.
==Computing operations of convolutional layer====
for $a \leftarrow 1$ **to** R **do**
  for $b \leftarrow 1$ **to** C **do**
    for $c \leftarrow 1$ **to** k **do**
      for $d \leftarrow 1$ **to** k **do**
        sum =0
        for $v \leftarrow 1$ **to** f **do**
          for $w \leftarrow 1$ **to** f **do**
            sum = sum+ F[v][w]*M[ss*(c−1)+v][ss*(d−1)+w]: where ss is the shifting stride
          end for
        end for
        FM[a][c][d] = FM[a][c][d]+sum:where FM is the feature map matrix.
        if b==C **then**
        FM[a][c][d] = F(FM[a][c][d]+B): where B is the used bias, and f is the ReLU activation function. FM[a][c][d] = max(0; (FM[a][c][d]+B));
      end for
    end for
  end for
end for
==Computing operations of pooling layer====
max-value = 0;
average-value= 0;
for $a \leftarrow 1$ **to** R **do**
  for $c \leftarrow 1$ **to** C **do**
    y = 0; **for** $d \leftarrow 1$ **to** k **do**
      x= 0; max-value = max(value,FM[a][c][d]); where the used operation is max-pooling
      average-value = + FM[a][c][d]; where the used operation is average-pooling
    end for
    $C_{pooling}[a][y][x]$ = max-value; where the used operation is max-pooling
    $C_{pooling}[a][y][x]$ = average-value/(r*c); where the used operation is average-pooling
    x++;
  end for
  y++;
end for
==Computing operations of fully connected layer====
for $a \leftarrow 1$ **to** R **do**
  vartemp=0; **for** $c \leftarrow 1$ **to** C **do**
    for $d \leftarrow 1$ **to** k **do**
      vartemp = vartemp+ $W_{connector}$[a][c][d] * $C_{pooling}[a][y][x]$; where $C_{pooling}$ is the pooling feature maps and $W_{connector}$ is the connector's weight
    end for
  end for
  Y[a][c][d] = vartemp;
end for
return *Set of features Y[a]*

F. PHASE V: FeedForward NEURAL NETWORK

After the CNN phase, the next step is the FFNN. This phase's main goal is to take the obtained set of features in the CNN phase and compute both values, which are negative and positive sentimental scores as described previously. Our simple FFNN version consists of four layers: the input fully connected layer, two hidden sigmoid layers, and the softmax output layer. The input fully connected layer is the same fully connected layer of the CNN deep learning model. So this layer consists of multiple neurons that represent the extracted features in the previous CNN phase. All neuron nodes in the fully connected layer are linked to all neuron nodes in the first hidden layer via the connections with different weights, which are adjustable. The hidden neuron value is computed using the following equation (25):

$$X_h = \sigma(\sum_{i=1}^{m} W_i * X_i) = \frac{1}{1 + e^{(\sum_{i=1}^{m} W_i * X_i)}} \tag{25}$$

where X_h is the value of the hidden neuron,W_i is the weight of the connector i, X_i is the value of every neuron in the fully connected layer, and σ is the sigmoid activation function, which is calculated using the following equation:

$$\sigma = \frac{1}{1 + e^{x}} \tag{26}$$

where σ is the sigmoid function, and e^x is the standard exponential function of the input value x.

The sigmoid function is applied to neural networks as an activation function and also known as a squashing function. i.e., this function ensures the neuron's output will be equal value in the interval [0,1]. In practice, the sigmoid function used at the level of the hidden layer is represented by the graphic representation in Fig. 16:

The second hidden layer in this work has the same function as the first hidden layer. It also uses the sigmoid activation method to compute the value of its hidden neuron nodes. The difference between both hidden layers is in the number of hidden neuron nodes. The second hidden layer has fewer hidden neuron nodes compared to the first hidden layer. Generally, In every neural network, the hidden layers are situated between the input and the output layers of the neural network model. At each hidden layer level, a weights function is applied to the inputs and passed them via an activation function as the output. i.e., the used hidden layers

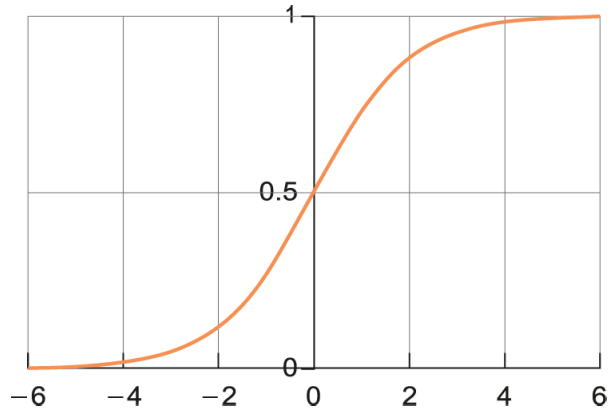


FIGURE 16. Graphic representation of the sigmoid activation function.

carry out the nonlinear conversion of the inputs that came into the neural network. Hidden layers differ from neural networks to others due to the function of every neural network. Also, the hidden layers may change, relying on their related weights. In this work, we employ these both hidden layers as squashing functions because the intended outputs of this model are probability degrees that are to say the output value will be in the interval [0,1].

The last layer of our used FFNN is the softmax output layer. The inputs entered into this layer are the outcomes of the second hidden layer multiplied by connections weights, and the outcome is passed through the softmax activation function at the level of the neuron nodes in the output layer. This output layer produces two values, which are PSS and NSS. The value of both is ranged between 0 no included, and 1 no included. Therefore the following algorithm (2) describes all FFNN's steps.

In this work, we also used the operation dropout, which indicates dropping out of a certain set of hidden and visible neurons in our FFNN in order to avoid the overfitting problem. We mean these randomly dropping neurons are not considered during the training phase in forwarding or backward feed as shown in Fig. 17. At each round in the training phase, every neuron is either inactive (dropout out) of the total architecture of FFNN with probability degree *1-p* or active with probability degree *p*. A question arises why do we shut down certain sets of neurons in all network layers ? the principal aim of the operation dropout is to prevent overfitting, resulting from the co-dependency establishing by neurons amongst each other at every round during the training stage. In short, dropout is a regularization method in FFNN, which leads to eliminating the interdependent training amongst the nodes.

G. PHASE VI: MAMDANI FUZZY SYSTEM

After our deep learning phase (CNN+FFNN), the next stage is the classification using the fuzzy classifier. Both outputs NSS and PSS of the CNN+FFNN deep learning model will be the inputs of our fuzzy classifier. The sentiments expressed by humans are vague and imprecision. So, it is difficult to decide if their opinions are negative, neutral, or positive about a particular topic. Our used deep learning

Algorithm 2: Our FeedForward Neural Network

Input : A given set pooled features maps $C_{pooling}$ described by **R** rows and **C** columns, and b = 1 is the used bias **Output:** Both values PSS and NSS.

Randomly initialize the weights of the neural network using the following equation $W_i^k = U_d[-\frac{1}{\sqrt{n^{k-1}}}; \frac{1}{\sqrt{n^{k-1}}}]$; where U_d is the continuous uniform distribution,n^{k-1} is the number of the neuron on the *(k-1)th* layer, and i is the *ith* connector

do

==Computing operations of the input fully connected layer====

for $a \leftarrow 1$ **to** R **do**

vartemp=0; **for** $b \leftarrow 1$ **to** C **do**

for $c \leftarrow 1$ **to** k **do**

vartemp = vartemp+ $W_{connector}$[a][b][c] * $C_{pooling}[a][b][c]$; where $C_{pooling}$ is the pooling feature maps and $W_{connector}$ is the connector's weight

end for

end for

Y[a][b][c] = vartemp;

end for

==Computing operations of the first hidden layer====

for $a \leftarrow 1$ **to** R **do**

vartemp=0; **for** $b \leftarrow 1$ **to** C **do**

for $c \leftarrow 1$ **to** k **do**

vartemp = vartemp+ $W_{connector}[a][b][c]$ * Y[a][b][c]; where Y is the set of the extracted feature by the CNN in the precedent phase, and $W_{connector}$ is the connector's weight

end for

end for

fh[a][b][c] = $\sigma(vartemp + B) = \frac{1}{1+e^{vartemp+B}}$; where σ is the sigmoid activation function.

end for

==Computing operations of the second hidden layer====

for $a \leftarrow 1$ **to** R **do**

vartemp=0; **for** $b \leftarrow 1$ **to** C **do**

for $c \leftarrow 1$ **to** k **do**

vartemp = vartemp+ $W_{connector}[a][b][c]$ * fh[a][b][c]; where *fh* is the output of the first hidden layer, and $W_{connector}$ is the connector's weight

end for

end for

sh[a][b][c] = $\sigma(vartemp + B) = \frac{1}{1+e^{vartemp+B}}$

end for

while $lf < 0.000001$

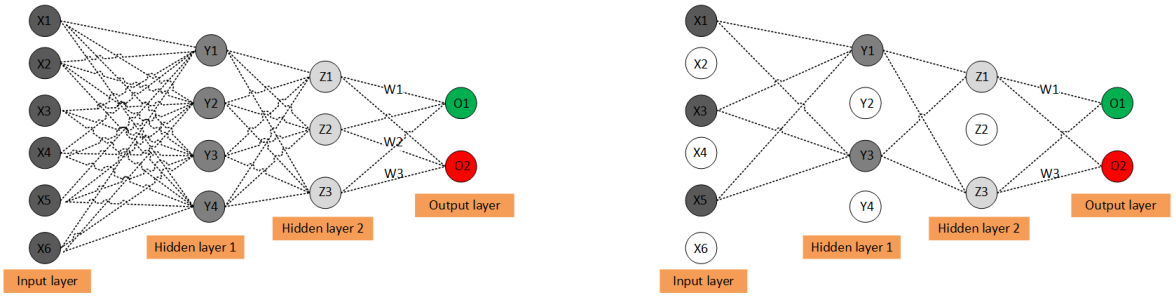


FIGURE 17. Feedforward neural network without and with dropout.

==Computing operations of the softmax output layer=====

for $a \leftarrow 1$ **to** R **do**

  vartemp=0; **for** $b \leftarrow 1$ **to** C **do**

    for $c \leftarrow 1$ **to** k **do**

      vartemp = vartemp+ $W_{connector}[a][b][c]$ * fh[a][b][c]; where *fh* is the output of the first hidden layer, and $W_{connector}$ is the connector's weight

    end for

  end for

  output[a][b][c] $= f(vartemp + B) = \frac{e^{vartemp+B}}{\sum_{i=1}^{l} e^{vartemp+B}}$

end for

oo_0 = output[0][0][0];

oo_1= output[0][0][0];

Therefore, the adaptation of this neural network can be performed by reducing (optimizing) the neural network loss function lf. The loss function is given by the following equation:

$lf(w(i)) = \frac{1}{N_c} * \sum_{i=1}^{N_c} \sum_{j=1}^{N-on} (ro - oo_j)^2$; where, lf(w(i)) is the error rate at the ith round, w(i) the actual weights of the connectors at the ith round; ro the required output neuron node; oo_j, the obtained value of the jth output neuron node; $N - on$, the number of output neuron nodes; N_c, the number of connectors. NSS = oo_0; PSS = oo_1;

return *Both values NSS and PSS*

model (CNN+FNN) is powerful for extracting the features from the given dataset but is powerless to handle with vagueness and ambiguous data. To make our proposal more accurate and more efficient, we have been applied the fuzzy set theory to the outputs of the suggested deep learning model (CNN+FFNN), mainly, we used the MFS as a fuzzy classifier.

MFS is constructed using the fuzzy set theory introduced by Zadeh [27]. The major aim of this theory is to handle the imprecise and vagueness concepts as the human brain is performing. According to multiple works in the literature, this theory proved its efficiency to deal with ambiguous data, and its ability to treat the data like the human brain. Thence, this theory is growingly applied for resolving real-life problems that cannot be solved and dealt with the application of classical set theory. Based on the Fuzzy set theory, multiple fuzzy systems are proposed. We find amongst Mamdani, Tsukamoto, and Sugeno fuzzy system [28]. Both later systems are applied in the regression problem, but the Mamdani is used in the classification issue. So our work is serving to resolve the classification issues. Therefore the suitable system is Mamdani. MFS consists of three major phases, which are the Fuzzification process, the Inference process, and the Defuzzification process, as illustrated in Fig. 18.

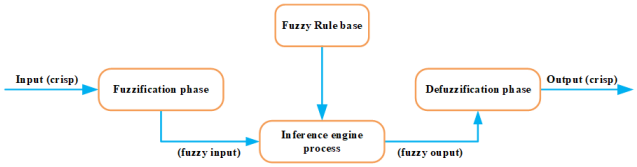


FIGURE 18. The components of the MFS.

The first stage before the fuzzification procedure is the definition of input and output linguistic variables and the definition of the linguistic terms of each linguistic variable. So, in this work, we have been applied the Mamdani as a fuzzy classifier on both outputs NSS and PSS of our deep learning model (CNN+FFNN). Thence the input linguistic variables are NSS and PSS and each variable takes five linguistic terms which are very low (is between 0.0 and 0.25), low (is between 0.0 and 0.50), moderate (is between 0.25 and 0.75), high (is between 0.50 and 1), and very high (is between 0.75 and 1). The output variable is the decision classification which has three linguistic terms neutral (is between 0.0 and 0.35), negative (is between 0.35 and 0.65), and positive (is between 0.65 and 1.0). In short, the inputs of the MFS will be the NSS and PSS, and the output will be the decision of classification as depicted in Table 3. After the definition of the linguistic variables and linguistic terms of our suggested fuzzy system. The next phase is the fuzzification process which is a substantial step in the MFS.

TABLE 3. Input and output parameters of the used fuzzy system.

Variables	Linguistic variables	Range	Linguistic terms	Parameters
Input	NSS	0-1	veryLow	0.0-0.25
			low	0.0-0.50
			moderate	0.25-0.75
			high	0.50-1
			veryHigh	0.75-1
Input	PSS	0-1	veryLow	0.0-0.25
			low	0.0-0.50
			moderate	0.25-0.75
			high	0.50-1
			veryHigh	0.75-1
Output	DC	0-1	negative	0.0-0.35
			neutral	0.35-0.65
			positive	0.65-1

1) FUZZIFICATION PROCESS

After the definition of linguistic variables and linguistic terms, the next phase is the fuzzification process. The fuzzification method is the operation that transforms the crisp input set into a fuzzy input set by computing the membership degree using one of the most popular MFs. Input variables of the MFS are represented on the fuzzy sets by the employ of MFs such as Triangular, Trapezoidal, or Gaussian, linguistic terms like very low, low, moderate, high, very high; and linguistic variables which are PSS, NSS, and DC. The linguistic variables and terms are significantly the complete phrases or the words of utilized NLP. When we are defining the linguistic terms and variables, we are convinced sufficiently that no numerical data are employed in the linguistic variables and terms. The two important points in this phase are the used MFs and the defined fuzzy sets because we are employing them to get the fuzzified values. The transformation of crisp input sets into fuzzy sets is carried out by utilizing of MFs, and this function of conversion is called fuzzification. So, in this fuzzification method, the crisp input variables are fuzzified by applying the used MF. In other words, the membership degree of belonging each input variable to each linguistic term (fuzzy set) is computed using a particular MF. The literature existing MFs are trapezoidal, triangular, Gaussian,2-D, Left-Right, Sigmoid and Generalized Bell membership functions. In this contribution, we applied the triangular, trapezoidal, Gaussian MFs, which are the most popular used membership functions in the literature. These functions are described below.

a: TRIANGULAR FUNCTION

is determined by Three parameters *ll*,*v* and *ul*. Where *ll* is the lower limit, *ul* is the upper limit, and the value *v*, where $ll < v < ul$ as shown in Fig. 19.

$$\mu_A(x) = \begin{cases} 0 & \text{if } x \leq ll \\ \dfrac{x - ll}{v - ll} & \text{if } ll \leq x \leq v \\ \dfrac{ul - x}{ul - v} & \text{if } v \leq x \leq ul \\ 0 & \text{if } c \leq x \end{cases} \tag{27}$$

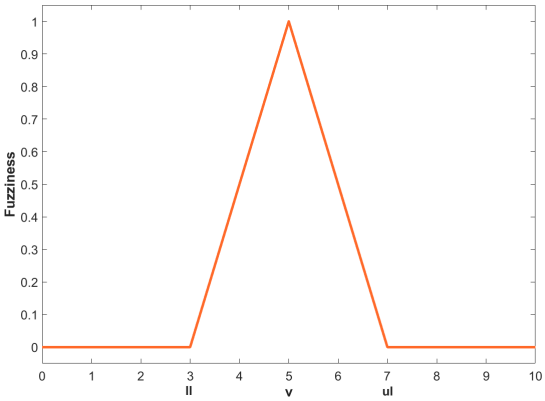


FIGURE 19. Representation graphic of Triangular function.

where $\mu_A(x)$ is the triangular MF of the input value *x*, *ll* is the lower limit, *ul* is the upper limit, and the median value *v*.

An alternative mathematical expression is obtained by applying the min and max functions on the previous equation.

$$\mu_A(x; ll, v, ul) = max(min(\frac{x - ll}{v - ll}, \frac{ul - x}{ul - v}), 0) \tag{28}$$

where $\mu_A(x; ll, v, ul)$ is the triangular MF of the input value *x*, *ll* is the lower limit, *ul* is the upper limit, and the median value *v*.

b: TRAPEZOIDAL FUNCTION

is determined by four parameters *ll*, *lsl*, *usl* and *ul*. Where *ll* is the lower limit, *lsl* is the lower support limit, *usl* is the upper support limit, and *ul* is the upper limit, and $ll < lsl < usl < ul$ as illustrated in Fig. 20.

$$\mu_A(x) = \begin{cases} 0 & \text{if } (x < ll) or (x > ul) \\ \dfrac{x - ll}{lsl - ll} & \text{if } ll \leq x \leq lsl \\ 1 & \text{if } lsl \leq x \leq usl \\ \dfrac{ul - x}{ul - usl} & \text{if } usl \leq x \leq ul \end{cases} \tag{29}$$

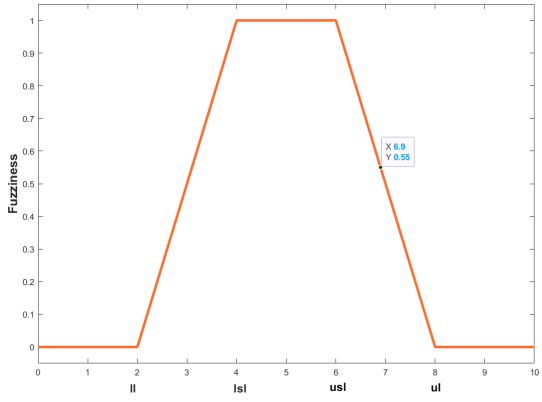


FIGURE 20. Representation graphic of Trapezoidal function.

An alternative mathematical expression is computed by applying the min and max functions on the preceding

equation:

$$\mu_A(x; ll, lsl, usl, ul) = max(min(\frac{x - ll}{lsl - ll}, 1, \frac{ul - x}{ul - usl}), 0) \tag{30}$$

where $\mu_A(x; ll, lsl, usl, ul)$ is the rapezoidal MF of the input value x, ll is the lower limit,lsl is the lower support limit, usl is the upper support limit, and $ul$is the upper limit.

c: GAUSSIAN FUNCTION

is determined by two parameters c and s. Where c is the central value, s is the standard deviation, and $s > 0$ as depicted in Fig. 21.

$$\mu_A(x) = e^{-\frac{(x-c)^2}{2.s^2}} \tag{31}$$

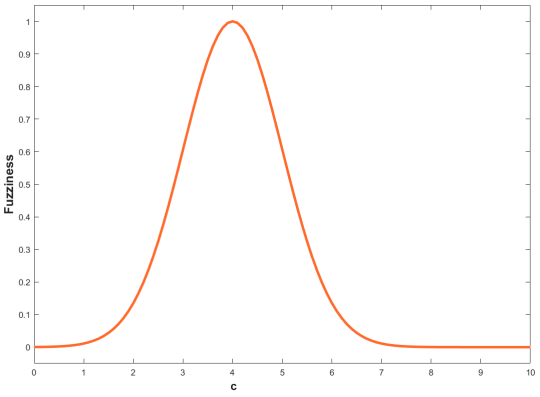


FIGURE 21. Representation graphic of Gaussian function.

2) DEFINING THE FUZZY RULES

After the fuzzification step, the next is the definition of fuzzy IF-THEN rules. For MFS, defining rules is deemed as the most important phase. Such IF-THEN fuzzy rules are usually formulated in an appropriate manner utilizing linguistic terms instead of employing numerical terms. They are mostly recognized as IF-THEN fuzzy rules, which are readily designed by harnessing vague conditional statements. IF-THEN fuzzy rules consist of two segments: a former block, which represents the inputs linguistic terms and variables, and a latter or consequent block, which represents the decision of the classification. All the Fuzzy IF-THEN rules that possess any truth in their former blocks will release and participate in the conclusion group. Each Fuzzy IF-THEN rule is released to a degree, which is a function that represents the degree to which its former block corresponds the input. This vague identification makes a foundation for fulfillment between probable input linguistic variables and aims to reduce the total number of Fuzzy IF-THEN rules in demand to determine the relationship between the input and the output. These Fuzzy IF-THEN rules have an efficient ability to resolve several real issues. Because they are similar to human knowledge, and human reasoning which is often appeared in the form of IF-THEN Fuzzy rules, at this phase, we employ empirical expert knowledge to produce a set of the IF-THEN fuzzy rules. As explained below, 25 rules are defined for the proposed fuzzy classifier.

Rule1: **IF** NSS is veryLow **AND** PSS is veryLow **THEN** DC is neutral

Rule2: **IF** NSS is veryLow **AND** PSS is low **THEN** DC is neutral

Rule3: **IF** NSS is veryLow **AND** PSS is moderate **THEN** DC is positive

Rule4: **IF** NSS is veryLow **AND** PSS is high **THEN** DC is positive

Rule5: **IF** NSS is veryLow **AND** PSS is veryHigh **THEN** DC is positive

Rule6: **IF** NSS is low **AND** PSS is veryLow **THEN** DC is neutral

Rule7: **IF** NSS is low **AND** PSS is low **THEN** DC is neutral

Rule8: **IF** NSS is low **AND** PSS is moderate **THEN** DC is positive

Rule9: **IF** NSS is low **AND** PSS is high **THEN** DC is positive

Rule10: **IF** NSS is low **AND** PSS is veryHigh **THEN** DC is positive

Rule11: **IF** NSS is moderate **AND** PSS is veryLow **THEN** DC is negative Rule12: **IF** NSS is moderate **AND** PSS is low **THEN** DC is negative

Rule13: **IF** NSS is moderate **AND** PSS is moderate **THEN** DC is neutral Rule14: **IF** NSS is moderate **AND** PSS is high **THEN** DC is positive

Rule15: **IF** NSS is moderate **AND** PSS is veryHigh **THEN** DC is positive

Rule16: **IF** NSS is high **AND** PSS is veryLow **THEN** DC is negative

Rule17: **IF** NSS is high **AND** PSS is low **THEN** DC is negative

Rule18: **IF** NSS is high **AND** PSS is moderate **THEN** DC is negative

Rule19: **IF** NSS is high **AND** PSS is high **THEN** DC is neutral

Rule20: **IF** NSS is high **AND** PSS is veryHigh **THEN** DC is neutral

Rule21: **IF** NSS is veryHigh **AND** PSS is veryLow **THEN** DC is negative Rule22: **IF** NSS is veryHigh **AND** PSS is low **THEN** DC is negative

Rule23: **IF** NSS is veryHigh **AND** PSS is moderate **THEN** DC is negative

Rule24: **IF** NSS is veryHigh **AND** PSS is high **THEN** DC is neutral

Rule25: **IF** NSS is veryHigh **AND** PSS is veryHigh **THEN** DC is neutral

H. INFERENCE ENGINE

After, we fuzzified the crisp input set to a fuzzy set using the fuzzification technique, and we defined the fuzzy IF-THEN rules. The next phase is the inference engine. The fuzzy

inference engine is exercised to incorporate the previously described fuzzy sets with taking into consideration the predefined fuzzy IF-THEN rules and the attached fuzzy zone individually. Generally, the inference engine process consists of three phases, which are application, implication, aggregation phases. The Min-Max inference technique or application technique is applied by the inference engine procedure to compute the rule conclusions employing the fuzzification results and determined Fuzzy IF-THEN rules. The outcome of this operation is known as the fuzzy conclusion. In the Mamdani inference engine, the real value of each Fuzzy IF-THEN rule is computed by the conjunction of the antecedent's blocks of the rules. With conjunction represented as *t-norm* = *minimum* in the logic connective "*AND*" case i.e. the process searches the rule with the minimum antecedent block that is considered to be the truth value of the fuzzy IF-THEN rule. This operation is expressed using the following equation (32):

$$\begin{aligned} \mu_A &= \mu_i(PSS)\ \ AND\ \ \mu_{ii}(NSS) \\ &= min(\mu_i(PSS), \mu_{ii}(NSS)) \end{aligned} \tag{32}$$

where $\mu_i(PSS)$ is the membership degree of the variable PSS, and $\mu_{ii}(NSS)$ is the membership degree of the variable NSS.

In the case of the logic connective "*OR*", the *t-norm* = *maximum*. i.e. the inference mechanism finds the rule with the maximum antecedent block that is deemed to be the real value of the fuzzy IF-THEN rule. This task is computed using the below equation (33):

$$\begin{aligned} \mu_A &= \mu_i(PSS)\ \ OR\ \ \mu_{ii}(NSS) \\ &= max(\mu_i(PSS), \mu_{ii}(NSS)) \end{aligned} \tag{33}$$

where μ_A is the membership degree obtained after the application phase,$\mu_i(PSS)$ is the membership degree of the variable PSS, and $\mu_{ii}(NSS)$ is the membership degree of the variable NSS.

In the application phase, the main goal is to extract the firing strength of each activated rule via the application of the conjunction of both computed membership degrees in the previous fuzzification step respectively for both numerical variables PSS and NSS.

At every fuzzy activated IF-THEN rule, an implication operation *I* is applied between the fuzzy outcome obtaining from the application stage and the classification decision of the rule. The operation minimum is the most operation used in the implication of Mamdani operation. The following equation (34) describes this implication phase:

$$\mu_I(DC_t) = min(\mu_A, \mu_i(DC_t) = 1) \tag{34}$$

where $\mu_I(DC_t)$ the membership degree obtained after the implication operation,μ_A is the membership degree obtained after the application phase the membership degree of the variable PSS, and $\mu_i(DC_t) = 1)$ is the membership degree of the decision classification attribute.

In the implication phase, the firing strength of an IF-THEN rule obtained in the previous phase (application) is employed to define the membership degree of the decision classification attribute '*DC*' to each linguistic term 'Negative', 'Neutral' or 'Positive', based on the consequent block of the IF-THEN fuzzy rule.

The ultimate phase in the inference engine mechanism is the aggregation operation of the outcomes obtained from the implication stage. i.e. all rule has the same classification decision will be aggregated. There are multiple aggregation operators, like geometric means, arithmetic mean, Max and Min. A commonly used operator is the Max which is given by the following equation (35):

$$\mu_{Ag}(DC_t) = max(\mu_{I1}(DC_t), \mu_{I2}(DC_t), \ldots, \mu_{In}(DC_t)) \tag{35}$$

where $\mu_I(DC_t)$ the membership degree obtained after the aggregation operation,$\mu_{Ii}(DC_t))$ is the membership degree of the decision classification attribute DC_t.

In the aggregation phase, the value of the decision classification attribute 'V_{DC}' obtained from each Fuzzy IF-THEN rule requires to compute its membership degree to the identical linguistic term (Positive, Neutral, or Negative) and determines the maximum membership degree among them.

I. DEFUZZIFICATION

After the inference engine process, the next phase is the defuzzification process which is used to convert the final fuzzy set obtained in the previous aggregation step into a real number. Also, the defuzzification is the approach that produces quantifiable outcomes in crisp logic which is accomplished from defining the fuzzy sets and membership techniques with proportionate degrees There are several commonly used defuzzification methods such as *Center of Gravity Method* (CGM), *Bisector of Area Approach* (BAA), *First of Maximum Procedure* (FMP) *Last of the Maximum Technique* (LMT), *Mean of the Maximum Approach* (MMA), *weighted Average Procedure* (WAP), and the *Center of Sums* (COS) Method. In this work, we applied six defuzzification methods which are CGM, BAA, WAP, and COS. These used defuzzification methods are described below:

1) CENTER OF GRAVITY METHOD

This approach converts the fuzzified value into a crisp output value via computing the centre of gravity of the input fuzzy set. The total zone of the MF spreading employed to monitor the standard action, which is split into a certain number of sub-zones. The zone and the centroid (centre of gravity) of every sub-zone are computed and hence the integration of all these sub-zones is calculated to get the defuzzified value for a continuous input fuzzy set. Unlike the case of the discrete fuzzy set, the summation of all these sub-zones is computed to determine the defuzzified value. In this work, the fuzzy sets take discrete values. Therefore the defuzzified value d_v is calculated using the summation instead of integration as the below equation (36) describes:

$$d_v = \frac{\sum_{i=1}^{n} z_i.\mu(z_i)}{\sum_{i=1}^{n} \mu(z_i)} \tag{36}$$

where z_i indicates the instance element,$\mu(z_i)$ is the membership degree of the element z_i, and n describes the number of elements in the instance.

2) BISECTOR OF AREA APPROACH

The Center of the area defuzzification approach computes the abscissa of the perpendicular line that splits the zone of the obtained membership function into two sub-zones with an equal surface. In other words, this method serves to compute the position under the curve where the sub-zones have the same surface, which is the crisp value corresponds to defuzified value. It is one of the widely applied approaches. The defuzzified value d_v is computed using the Equation (37):

$$\int_{\alpha}^{zBOA} \mu_i(z).d_z = \int_{zBOA}^{\beta} \mu_i(z).d_z \tag{37}$$

where $\alpha = min\{z; z \in Z\}$, $\beta = max\{z; z \in Z\}$ and $z = zBOA$ is the vertical line that divides the area between $z{=}\alpha$, $z{=}\beta$ $v{=}0$ *and* $v{=}\mu_i(z)$ into two areas with the same region, $\mu_i(z)$ is the membership degree of the element z, and d_z is the derivative of the element z

3) WEIGHTED AVERAGE PROCEDURE

This approach is suitable for input fuzzy sets with identical output MFs and generates outcomes very close to the centre of area approach. This technique used less computational resources. Every membership method is weighted by its membership degree that has the maximum value. The defuzzified value d_v is determined as the below equation (38) describes:

$$d_v = \frac{\sum \mu_i(z).z}{\sum \mu_i(z)} \tag{38}$$

where $\sum$ indicates the algebraic summation, z is the element that has the maximum membership degree, and $\mu_i(z)$ is the membership function of the element z where i is the linguistic term.

4) CENTER OF SUMS (COS) METHOD

It is a widely applied defuzzification method. In this approach, the overlapping zone is computed twice. It is faster compared to other defuzzification methods. It applies algebraic sum on all output fuzzy sets. It is identical to the weighted average approach; however, in this approach, the weights are the zones, instead of membership degrees in the weighted average approach. The defuzzified value d_v is calculated using the below equation (39):

$$d_v = \frac{\sum_{ii=1}^{n} z_{ii} \cdot \sum_{j=1}^{k} \mu_{i_j}(z_{ii})}{\sum_{ii=1}^{n} \cdot \sum_{j=1}^{k} \mu_{i_j}(z_{ii})} \tag{39}$$

where n is the total number of used fuzzy sets, K is the total number of fuzzy linguistic variables, μ_{i_j} is the membership degree for the *j-th* fuzzy set.

a: OUTPUT OF THE DEFUZZIFICATION APPROACH

After the application of one of the defuzzification method on the aggregated value obtained in the aggregation phase. The applied defuzzification method transforms the fuzzy aggregated input to the crisp output.which is obtained by the application of all steps of the fuzzy inference engine process. It consists of the different kinds of the decision of the classification output variable, which are computed by the defuzzified value. Therefore we employ defuzzification rules to define the relationship between defuzzified value and decision classification.

b: DEFUZZIFICATION RULES

Here are all possible rules of defuzzification where d_v signifies the defuzzified value while d_c signifies the decision of the classification. These rules are used to determine the final crisp output, which is either negative, neutral, or positive.

if $(0.0 \leq d_v \leq 0.35)$, then $d_c =$ Negative
if $(0.35 < d_v \leq 0.65)$, then $d_c =$ Neutral
if $(0.65 < d_v \leq 1.0)$, then $d_c =$ Positive

V. PARALLELIZATION OF OUR PROPOSAL USING HADOOP FRAMEWORK

One of the most terrible shortcomings of our deep neural networks (CNN+FNN) is the long execution time. Such a time-consumption issue prevents the trained deep learning models from speedily get more accurate information and perform the required tasks. To curb this execution time problem, we have been applied the Hadoop framework [56] to our proposed approach, which is a useful framework that serves to improve the forecasting effectiveness and scalability of our proposed fuzzy deep learning model. The Hadoop platform parallelizes our FDLC between multiple computing nodes. This framework utilizing its *Hadoop Distributed File System* (HDFS) for stocking both used social-media datasets in this work (sentiment140, and COVID-19 Sentiments) to be classified and the decision of the classification, and MapReduce programming model that processes and treats our fuzzy deep learning tasks in a parallel manner using multiple mappers and reducers as illustrated in Fig. 22. Implementing our FDLC on the MapReduce programming model mostly consists of three stages: the Map phase, the Combining stage, and the Reduce stage, introduced in brief details as follows.

- **Map phase:** The map phase consists of four mappers; each mapper read one or more data chunks from HDFS as a different key-value pair's inputs data. The mapper applies the text-preprocessing to each chunk, then transforms it to numerical data based on the word embeddings phase, and passes it through our deep learning model (CNN+FFNN), finally applies the fuzzy classifier to the processed chunk. After dealing with all data chunks, The outputs obtained by using our FDLC are turned into a set of intermediate key-value pairs and write them on the local disk.

Algorithm 3: Our Fuzzy Classifier Based on Mamdani Fuzzy System

Input : A obtained both sentimental scores NSS and PSS in the deep learning phase. each variable take five linguistic terms which are very low (is between 0.0 and 0.25),low(is between 0.0 and 0.50),moderate(is between 0.25 and 0.75),high (is between 0.50 and 1),and very high(is between 0.75 and 1)

Output: Decision of classfication which is determined in three label neural,negative or positive.

Phase 1: Definition of the input linguistic and output linguistic variables and the definition of linguistic terms of every linguistic variable.

Phase 2: Fuzzification process

2.1 Use Gaussian, Triangular, or Trapezoidal membership function

2.2 Calculates the membership degree of each linguistic term using the selected membership function.

2.3 Transform every crisp set to fuzzy set

Phase 3: Generates IF-THEN fuzzy rules based on our expert knowledge

Phase 4: Inference engine process

4.1 Application phase

4.2 Implication phase

4.3 Aggregation phase

Phase 5: Defuzzification process

5.1 Use BOA, COA, WAM, or COS defuzzification methods

5.2 Transform the obtained aggregated fuzzy value in the aggregation phase to crisp or real value by the application of one defuzzification method.

5.3 From the resulted crisp value, discovery the classification decision.

if ($0.0 \leq d_v \leq 0.35$), then d_c = Negative

if ($0.35 < d_v \leq 0.65$), then d_c = Neutral

if ($0.65 < d_v \leq 1.0$), then d_c = Positive

Where d_v is the defuzzified value, and d_c is the decision of the classification.

return d_c

- **Group by keys:** The MapReduce programming model carries out this operation. Its main goal is aggregated all obtained intermediate values in the Mapper with the same intermediate key into an array list of values and passed it to the Reducer.
- **Reduce phase:** In our work, the Reduce phase consists of four Reducers; each reducer receives all intermediate array list of values from all mapper. The reducer worked on one key simultaneously and aggregated the list of values associated with that key in a smaller set [56]. Finally, all reducers outputs are combined and merged as one intermediate output and write this resulted output as output key-value pair on HDFS as depicted in Fig. 22.

The advantage of Hadoop is its ability to prohibit the problem of server failures by storing information redundantly on several compute server, which aid to back up data automatically. i.e the same piece of information is recorded on multiple computing servers. If one of those compute servers fails, the amount of data is still available on another computing server. MapReduce programming system is a software that offers scalable and reliable conditions for the process and implementation of distributed applications. To be more accurate, this programming framework broken automatically the computations into multiple parallelization tasks. Like if one task fails to accomplish its processing work, it can be refreshed without any negative influences to other running tasks. MapReduce prevents the issue of network bottlenecks by making the computation tasks close to stored data and prohibits copying data around the network, and this reduces a network bottleneck issue and leads to information and computational load balancing. MapReduce Model also supplies their users a very straightforward and straightforward model which stashes the complications of all computing tasks of its functioning.

In this contribution, we have used the Hadoop framework in our proposal to minimize the execution time and improve our FDLC. Our proposed method is involving both used massive datasets (Sentiment140, and COVID-19 Sentiments). In the first step, we have employed the HDFS to store and share the enormous dataset parallel between all the computing servers in the cluster Hadoop. After we store the dataset in HDFS, the next step is applying our FDLC. In this second step, we used the MapReduce programming model to parallelize our approach between all computing nodes of the Hadoop cluster. The input in every round of the MapReduce algorithm is a sentence to be classified, and the outcome is a classified sentence with the classification's decision. The outcome of the classification of each sentence will also be stored in HDFS. All these steps are described in Fig. 22, and algorithm 4 presents the MapReduce algorithm applied in our work to classify the sentences using our fuzzy deep learning classifier.

VI. EXAMPLE OF THE APPLICATION

This section illustrates our FDLC approach with an example i.e. in this section We will explain the followed steps to classify each sentence S, according to three class labels (Negative, Neutral, or Positive). The first step of our algorithm is to check all words in each sentence S are written in the English language. If not the case, we use the translated function to translate those words in other languages into the English language. For example, we have $S =$"*The deep neural networks are very efficiencccccccy to process data; but they are inefficiency with the ingrained ambiguity in NL que demande d'autre solutions.*"; The parts "*que demande d'autre solutions*" of the sentence S is translated to "*that needs more solutions*," Then after the application of pre-processing techniques, we obtained the sentence S as a set of token (T) like Table 4 describes.

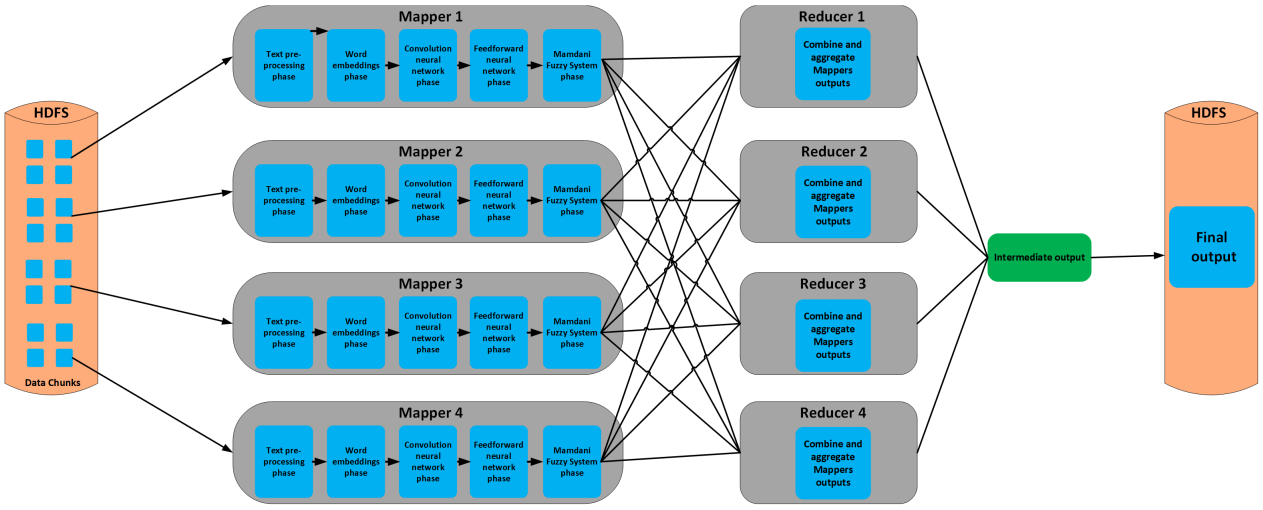


FIGURE 22. Parallel architecture of our proposal using MapReduce.

TABLE 4. Set of token represents the sentence to be classified.

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
deep	neural	networks	very	efficienccy	process	data	inefficiency	ambiguity	natural language	needs	solutions

Algorithm 4: Our MapReduce Programming Model

Input : Data Chunks
Output: Decision of classfication
if (a word in sentence not in English) then
Translate(word into English)
T = Text-preprocessing(Data chunks)
W = Word-embedding(T)
C = CNN(W)
NSS = FFNN(C)
PSS = FFNN(C)
$\mu(NSS)$ = Fuzzification(NSS)
$\mu(PSS)$ = Fuzzification(PSS)
A = Application(μ(NSS),μ(PSS),IF-THEN rules)
I = Implication(A,$\mu(DC_t)$=1)
Ag = Aggregation(I)
d_v = Defuzzification(Ag)
if ($0.0 \leq d_v \leq 0.35$), then d_c = Negative
if ($0.35 < d_v \leq 0.65$), then d_c = Neutral
if ($0.65 < d_v \leq 1.0$), then d_c = Positive
Where d_v is the defuzzified value, and d_c is the decision of the classification.
return d_c

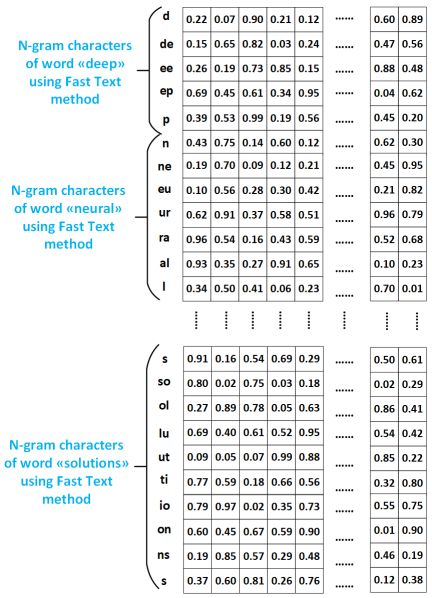


FIGURE 23. Word embedding matrix of sentence S.

After we get the set of tokens, we have applied the Fast-text word embedding approach to transform the text-based data to numerical-based data, as shown in Fig. 23.

As illustrated in Fig. 23, The Fast-text word embedding method applies the n-gram=2 characters to each word in the sentence *S*. for example the n-gram=2 for word "deep" will be (*d,de,ee,ep,p*). Then Fast-text uses either CBOW or Skip-Gram to compute the word embedding for each n-gram word and generates the word embedding matrix of the sentence *S*. After the embedding matrix is obtained, CNN automatically extracts the essential feature from this matrix. So Fig. 24 describes CNN's steps.

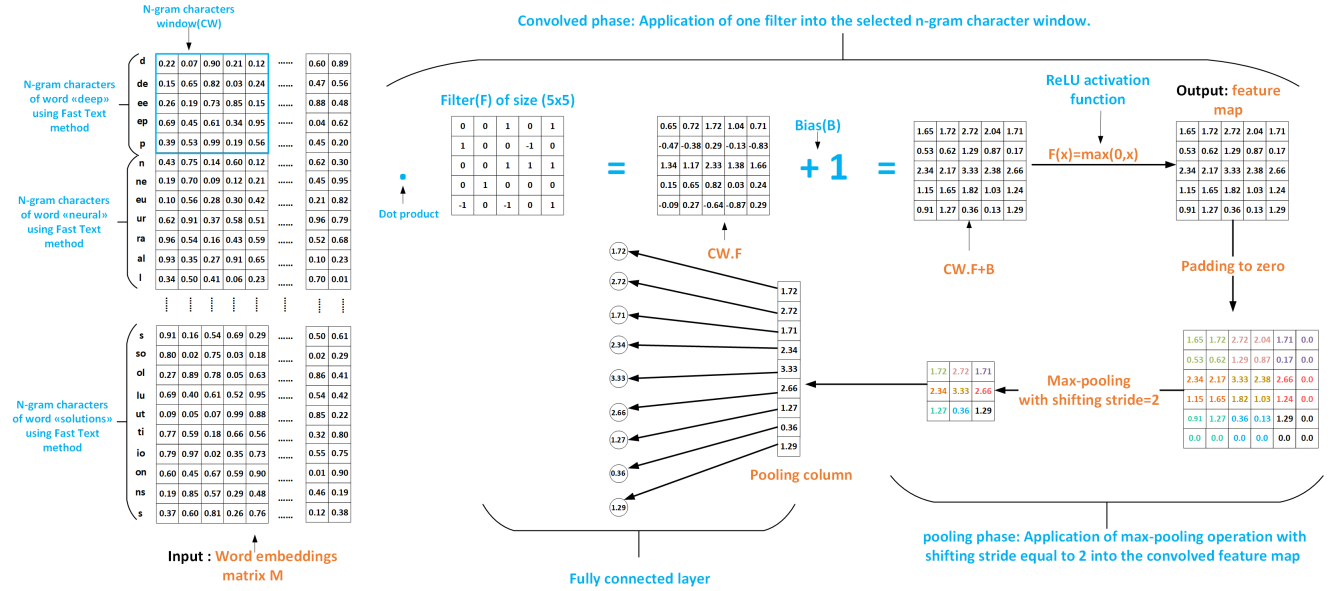


FIGURE 24. Application of CNN's steps.

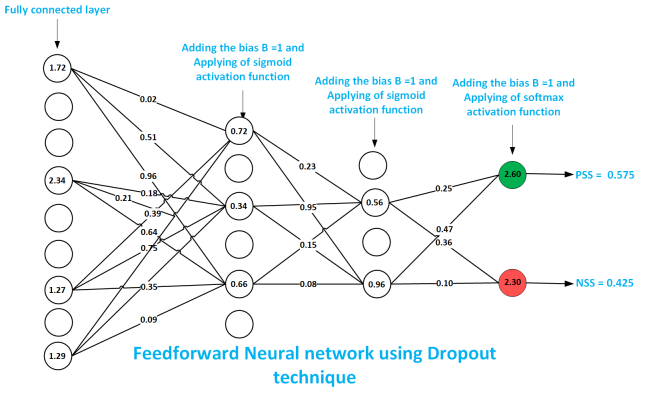


FIGURE 25. Application of FFNN's stages.

As seen in Fig. 24, CNN's first step is the convolved operation, which serves to apply multiple filters to each n-gram character windows *CW*. To clarity the convolved operation in an accurate manner, Fig. 24 illustrates only the application of one filter *F* to one n-gram character window *CW*, and we obtain *F.CW* matrix. Then, we add the bias *b* equal to 1, and we get the matrix *F.CW+1*. Finally, we applied the ReLU activation function to *F.CW+1* matrix, and we obtain the first feature map. After the convolved operation, the next stage is the pooling operation, in which we reduce the feature map dimensionality. In this example, we use the max-pooling with the shifting stride equal to 2. For that, it must pad to zero the feature map to get its dimensionalities divisible by 2. Then we apply the max-pooling to the padded matrix, and we get the pooling column. Finally, we pass the obtained pooling column to the fully connected layer. After we extract the essential feature using CNN, the next phase is the application of FFNN to compute both NSS and PSS values. Therefore Fig. 25 presents the FFNN's steps.

As presented in Fig. 25, the first layer of our FFNN simple version is the fully connected layer, followed by two hidden layers and the output layer. We used at the level of both hidden layers the sigmoid activation function, and at the level of of each neuron node of the output layer, we applied the softmax activation method. The input of this network is a fully connected layer obtained in the CNN phase, and the outputs are both PSS and NSS. The value of each hidden neuron H_{nv} is computed by multiplying the all connected weights to this hidden neuron into its value and add the bias b then calculated the sigmoid activation function. In the following an example of calculating the value of the first hidden neuron in the first hidden layer:

$$\begin{aligned} H_{nv} &= f((wi_1 * v_1 + wi_2 * v_2 + wi_3 * v_3 + wi_4 * v_4) + B) \\ &= f((1.72 * 0.02 + 2.34 * 0.15 \\ &\qquad + 1.27 * 0.39 + 1.29 * 0.75) + b) \\ &= f(0.0344{+}1{+}0.351{+}1{+}0.4953{+}1{+}0.9675 + 1) \\ &= f(5.85) = \frac{1}{1 + e^{5.85}} = \frac{1}{1 + 347.23} \\ &= \frac{1}{348.23} = 0.00287 \end{aligned}$$

The same manner is used to calculate the value of each neuron node in the output layer. The only difference is that instead of using the sigmoid function for calculating the value of each hidden neuron, we employ the softmax activation function for computing the value of each neuron node in the output layer. The following example explains how we calculate the value of both neurons in the output layer:

$$\begin{aligned} ON_{v1} &= wi_1 * v_1 + wi_2 * v_2) + B \\ &= (0.56 * 0.25 + 0.96 * 0.47) + B \\ &= 0.14 + 1 + 0.4512 + 1 = 2.60 \\ ON_{v2} &= wi_3 * v_1 + wi_4 * v_2) + B \end{aligned}$$

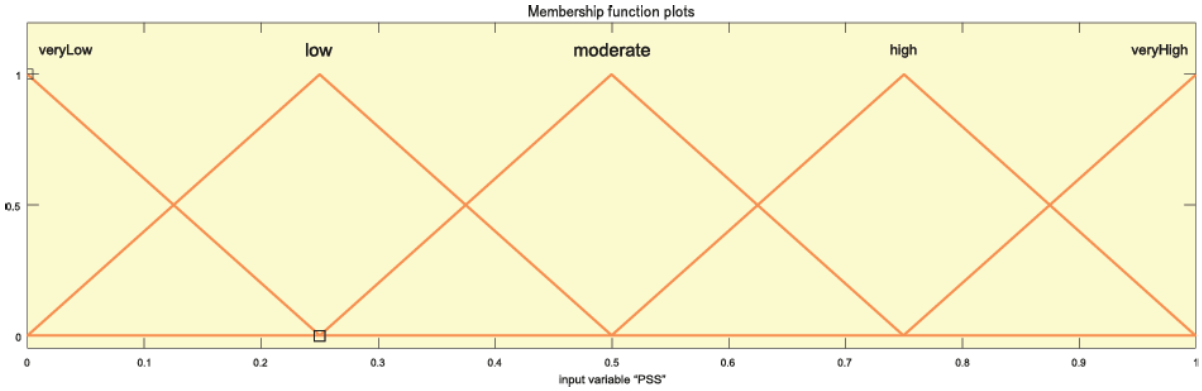


FIGURE 26. Membership degree of each linguistic term.

$$= (0.56 * 0.36 + 0.96 * 0.10) + B$$
$$= 0.2016 + 1 + 0.096 + 1 = 2.30$$

Therefore, both PSS and NSS values are calculated by applying the softmax activation function on both values ON_{v1}, and ON_{v2} as follows.

$$PSS = \frac{e^{2.60}}{e^{2.60} + e^{2.30}} = \frac{13.46}{13.46 + 9.97} = \frac{13.46}{23.43} = 0.575$$
$$NSS = \frac{e^{2.30}}{e^{2.60} + e^{2.30}} = \frac{9.97}{13.46 + 9.97} = \frac{9.97}{23.43} = 0.425$$

After we compute both **PSS=0.575** and **NSS=0.425** values using our proposed FDLC, the next phase is the application of the MFS. So, The first stage of MFS is the application of the fuzzification process into both NSS and PSS crisp values, i.e., we use the triangular (or trapezoidal, or gaussian) membership function to compute the membership degree of belonging of the NSS and PSS to veryLow, low, moderate, high, and veryHigh fuzzy sets. In this example, we have used the triangular MF represented by Equation (27) and Fig. 19. The calculating process is introduced as follows based on Fig. 26.

For the linguistic term veryLow, and the optimal scalar parameters are ll=0; v=0.125; and ul=0.25; then, we used these parameters to calculate the membership degrees of both linguistic variables NSS and PSS of belonging to the fuzzy set veryLow. The results like the following:

- We have PSS=0.575 $\geq$ ul=0.25 Therefore; $\mu_{veryLow}(PSS) = \mathbf{0}$
- We have NSS=0.425 $\geq$ ul=0.25 Therefore; $\mu_{veryLow}(NSS) = \mathbf{0}$

Therefore, the values of each used membership function's parameters were determined experimental, and we take the optimal values of these parameters that provide better classification performance.

For the linguistic term low, the optimal scalar parameters are ll=0; v=0.25; and ul=0.5; then, we used these parameters to calculate the membership degrees of both linguistic variables NSS and PSS of belonging to the fuzzy set low. The results like the following:

- We have PSS=0.575 $\geq$ ul=0.5 Therefore; $\mu_{low}(PSS) = \mathbf{0}$
- We have v=0.25 $\leq$ NSS=0.425 $\leq$ ul=0.5 Therefore; $\mu_{low}(NSS) = \frac{ul-NSS}{ul-v} = \frac{0.5-0.425}{0.5-0.25} = \mathbf{0.3}$

For the linguistic term moderate, the optimal scalar parameters are ll=0.25; v=0.5; and ul=0.75; then, we used these parameters to calculate the membership degrees of both linguistic variables NSS and PSS of belonging to the fuzzy set moderate. The results like the following:

- We have v=0.5$\leq$ PSS=0.575$\leq$ ul=0.75 Therefore; $\mu_{moderate}(PSS) = \frac{ul-PSS}{ul-v} = \frac{0.75-0.575}{0.75-0.5} = \mathbf{0.7}$
- We have ll=0.25 $\leq$ NSS=0.425 $\leq$ v=0.5 Therefore; $\mu_{moderate}(NSS) = \frac{NSS-ll}{v-ll} = \frac{0.425-0.25}{0.5-0.25} = \mathbf{0.7}$

For the linguistic term high, the optimal scalar parameters are ll=0.5; v=0.75; and ul=1; then, we used these parameters to calculate the membership degrees of both linguistic variables NSS and PSS of belonging to the fuzzy set high. The results like the following:

- We have ll=0.5$\leq$ PSS=0.575 $\leq$ v=0.75 Therefore; $\mu_{high}(PSS) = \frac{PSS-ll}{v-ll} = \frac{0.575-0.50}{0.75-0.50} = \mathbf{0.30}$
- We have NSS=0.425 $\leq$ ll=0.5 Therefore; $\mu_{high}(NSS)$ **0**

For the linguistic term veryHigh, the optimal scalar parameters are ll=0.75; v=0.875; and ul=1; then, we used these parameters to calculate the membership degrees of both linguistic variables NSS and PSS of belonging to the fuzzy set high. The results like the following:

- We have PSS=0.575 $\leq$ ll=0.75 Therefore; $\mu_{veryHigh}(PSS) = \mathbf{0}$
- We have NSS=0.425 $\leq$ ll=0.75 Therefore; $\mu_{veryHigh}(NSS) = \mathbf{0}$

As we said earlier, the next step after the fuzzification process is the application process of the 25 generated IF-Then fuzzy rules. The primary objective of this operation is to discover the firing strength of each activated fuzzy IF-THEN rule via the application of the conjunction of both computed

numerical variables PSS and NSS in the fuzzification phase. The computing steps of this process are presented as follows:

- Rule1: **IF** (NSS is veryLow) = 0 **AND** (PSS is veryLow) = 0 **THEN** (DC is neutral)=Min(0,0)=0
- Rule2: **IF** (NSS is veryLow) = 0 **AND** (PSS is low)= 0 **THEN** (DC is neutral) = min(0,0)= 0
- Rule3: **IF** (NSS is veryLow) = 0 **AND** (PSS is moderate) = 0.7**THEN** (DC is positive)= min(0,0.7) = 0
- Rule4: **IF** (NSS is veryLow) = 0 **AND** (PSS is high) = 0.30 **THEN** (DC is positive)= min(0,0.30) = 0
- Rule5: **IF** (NSS is veryLow) = 0 **AND** (PSS is veryHigh) = 0 **THEN** (DC is positive) = min(0,0) = 0
- Rule6: **IF** (NSS is low)=0.3 **AND** (PSS is veryLow) = 0**THEN** (DC is neutral) = min(0.3,0)=0
- Rule7: **IF** (NSS is low)=0.3 **AND** (PSS is low)= 0 **THEN** (DC is neutral)= min(0.3,0) =0
- Rule8: **IF** (NSS is low)=0.3 **AND** (PSS is moderate)=0.7 **THEN** (DC is positive) = min(0.3,0.7)= 0.3
- Rule9: **IF** (NSS is low)=0.3 **AND** (PSS is high)=0.3 **THEN** (DC is positive) = min(0.3,0.3)= 0.3
- Rule10: **IF** (NSS is low)=0.3 **AND** (PSS is veryHigh)=0 **THEN** (DC is positive) = min(0.3,0) =0
- Rule11: **IF** (NSS is moderate)=0.7**AND** (PSS is veryLow)=0 **THEN** (DC is negative)=min(0.7,0)=0
- Rule12: **IF** (NSS is moderate)=0.7 **AND** (PSS is low)= 0**THEN** (DC is negative)=min(0.7,0)=0
- Rule13: **IF** (NSS is moderate)=0.7 **AND** (PSS is moderate)=0.7 **THEN** (DC is neutral)=min(0.7,0.7) =0.7
- Rule14: **IF** (NSS is moderate)=0.7 **AND** (PSS is high)=0.375 **THEN** (DC is positive)=min(0.7,0.3) =0.3
- Rule15: **IF** (NSS is moderate)=0.7 **AND** (PSS is veryHigh)=0 **THEN** (DC is positive)==min(0.7,0)=0
- Rule16: **IF** (NSS is high)= 0**AND** (PSS is veryLow)= 0**THEN** (DC is negative)=min(0,0)=0
- Rule17: **IF** (NSS is high)= 0 **AND** (PSS is low)= 0**THEN** (DC is negative)=min(0,0)=0
- Rule18: **IF** (NSS is high)= 0 **AND** (PSS is moderate)= 0.7**THEN** (DC is negative)=min(0,0.7)=0
- Rule19: **IF** (NSS is high)= 0 **AND** (PSS is high)=0.3 **THEN** (DC is neutral)=min(0,0.3)=0
- Rule20: **IF** (NSS is high)= 0 **AND** (PSS is veryHigh)=0 **THEN** (DC is neutral)=min(0,0)=0
- Rule21: **IF** (NSS is veryHigh)=0 **AND** (PSS is very low)=0 **THEN** (DC is negative)=min(0,0)=0
- Rule22: **IF** (NSS is veryHigh)=0 **AND** (PSS is low)=0 **THEN** (DC is negative)=min(0,0)=0
- Rule23: **IF** (NSS is veryHigh)=0 **AND** (PSS is moderate)=0.7 **THEN** (DC is negative)=min(0,0.7)=0
- Rule24: **IF** (NSS is veryHigh)=0 **AND** (PSS is high)=0.3 **THEN** (DC is neutral)=min(0,0.30)=0
- Rule25: **IF** (NSS is veryHigh)=0 **AND** (PSS is veryHigh)=0 **THEN** (DC is neutral)= min(0,0)=0

The following stage is the implication process. The main goal of the implication phase, as we presented previously, is the calculation of the membership degree of the decision classification attribute 'DC' to each linguistic term 'Negative,' 'Neutral' or 'Positive,' based on the firing strength of an IF-THEN rule obtained in the previous application phase, and on the consequent block of the IF-THEN fuzzy rule. the computing steps are presented below:

- Rule1:μ_1(neutral)=min(0,1)=0
- Rule2:μ_2(neutral)=min(0,1)=0
- Rule3:μ_3(positive)=min(0,1)=0
- Rule4:μ_4(positive)=min(0,1)=0
- Rule5:μ_5(positive)=min(0,1)=0
- Rule6:μ_6(neutral)=min(0,1)=0
- Rule7:μ_7(neutral)=min(0,1)=0
- Rule8:μ_8(positive)=min(0.3,1)=0.3
- Rule9:μ_9(positive)=min(0.3,1)=0.3
- Rule10:μ_{10}(positive)=min(0,1)=0
- Rule11:μ_{11}(negative)=min(0,1)=0
- Rule12:μ_{12}(negative)=min(0,1)=0
- Rule13:μ_{13}(neutral)=min(0.625,1)=0.7
- Rule14:μ_{14}(positive)=min(0.375,1)=0.3
- Rule15:μ_{15}(positive)=min(0.625,1)=0
- Rule16:μ_{16}(negative)=min(0,1)=0
- Rule17:μ_{17}(negative)=min(0,1)=0
- Rule18:μ_{18}(negative)=min(0,1)=0
- Rule19:μ_{19}(neutral)=min(0,1)=0
- Rule20:μ_{20}(neutral)=min(0,1)=0
- Rule21:μ_{21}(negative)=min(0,1)=0
- Rule22:μ_{22}(negative)=min(0,1)=0
- Rule23:μ_{23}(negative)=min(0,1)=0
- Rule24:μ_{24}(neutral)=min(0,1)=0
- Rule25:μ_{25}(neutral)=min(0,1)=0

After the implication process, the next process is the aggregation process, which aims to aggregate each class label(Positive, Neutral, or Negative) among them and find the maximum membership degree among the aggregated values. Also, the computing steps are carried out as follows:

- μ(**positive**)= μ_3(positive) $\vee$ μ_4(positive) $\vee$ μ_5(positive) $\vee$ μ_8(positive) $\vee$ μ_9(positive) $\vee$ μ_{10}(positive) $\vee$ μ_{14}(positive) $\vee$ μ_{15}(positive) = **max(0,0,0,0.3,0.3,0, 0.3,0) = 0.3**
- μ(**negative**)= μ_{11}(negative) $\vee$ μ_{12} (negative) $\vee$ μ_{16}(negative) $\vee$ μ_{17}(negative) $\vee$ μ_{18}(negative) $\vee$ μ_{21}(negative) $\vee$ μ_{22}(negative) $\vee$ μ_{23}(negative) = **max(0,0,0,0,0,0,0,0) = 0**
- μ(**neutral**)= μ_1(neutral) $\vee$ μ_2(neutral) $\vee$ μ_6(neutral) $\vee$ μ_7(neutral) $\vee$ μ_{13}(neutral) $\vee$ μ_{19}(neutral) $\vee$ μ_{20}(neutral) $\vee$ μ_{24}(neutral) $\vee$ μ_{25}(neutral) = **max(0,0, 0,0,0.7,0,0,0,0) = 0.7**

Finally, and after the aggregation process, we use the Centroid of Area defuzzification method to defuzzify the fuzzy aggregated outputs to obtain one crisp result that indicates the class label. This process is performed by following these described steps below:

Step 1: Accordingly to Fig. 27 we compute the Centroid of negative label that will be equal to (0.0+0.35)/ 2 = **0.175**, the Centroid of neutral label that will be equal to

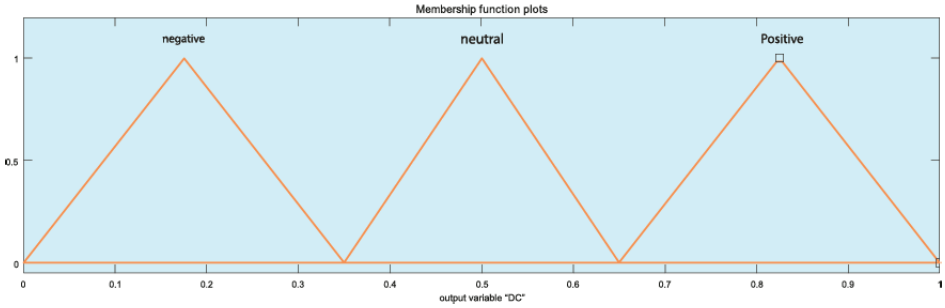


FIGURE 27. Membership degree of each class label.

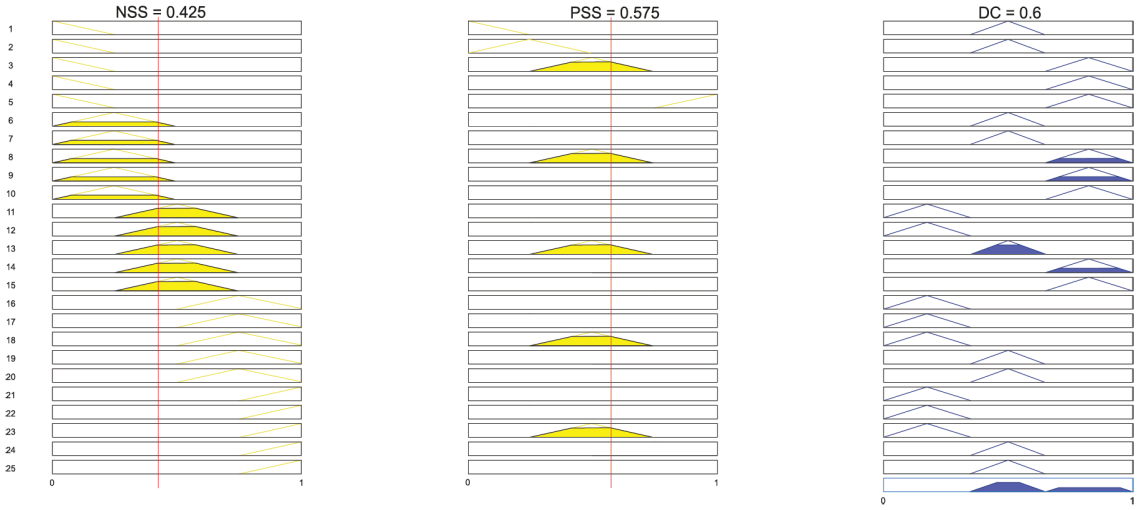


FIGURE 28. Defuzzified value computed using rules review in Matlab.

(0.35+0.65)/2 = **0.5**, and the Centroid of positive label that will be equal to (0.65+1)/2 = **0.825**

Step 2: Accordingly to Centroid of Area defuzzification method introduced in equation (36). The defuzzfied value d_v is calculated using the computed Centroid in the first step and the membership degree of each label as follows: $d_v = \frac{0*0.175+0.7*0.5+0.825*0.3}{0+0.7+0.3} = \mathbf{0.60}$

Step 3: Accordingly to step 2 the defuzzified value $d_v = 0.60$ is between 0.35 and 0.65. Therefore; the decision of classification of the sentence **S** is **neutral**

Accordingly to Fig. 28 we remark that the defuzzified value is equal to 0.60 for PSS=0.575 and NSS=0.425. This defuzzified value is is between 0.35 and 0.65. Therefore; the decision of classification of the sentence *S* is *neutral*. So we get the same conclusion as the performed manually computing.

VII. THE EXPERIMENT AND THE RESULTS

This section describes the experimental results of our fuzzy deep learning classifier (CNN + FFNN + MFS). These experimental results are provided by applying our fuzzy deep learning classifier and other literature methods on both used datasets, as presented in the data-collection subsection. Generally, in this study, we split the given dataset into a training dataset, which represents 90% of the overall dataset, and a testing dataset representing only 10% of the comprehensive dataset. After that, we store both obtained datasets(training and testing) into HDFS. Once the storage is finished, we apply multiple text pre-processing techniques on training and testing datasets to reduce and remove the noisy data. Then we implement the most efficient word embedding approach to transform the text-based data into numerical-based data. Besides, we apply our proposed fuzzy deep learning classifier on the testing dataset, and we stock the classified data into HDFS. Also, we carry out the classification based on our proposed classifier in a parallel manner by applying the Hadoop framework with its HDFS and its MapReduce programming framework. In this work, the Hadoop cluster consists of five computing nodes; one master computing node and four salve computing nodes. For assessing the experimental results, we have calculated ten evaluation metrics exhibited in the performance metrics subsection. The ten metrics comprise TPR, TNR, FPR, FNR, ER, PR, AC, KS, FS, and TC.

In this work, we performed the experiments in five steps. First, we kept the same parameters of our suggested

TABLE 5. Parameters settings of our FDLC that we used to assess the word embedding approaches.

Parameters settings of CNN		Parameters settings of FFNN	
Parameters	**value**	**Parameters**	**Value**
Vocabulary size	40,000	N. of fully connected layers	1
Input embedding matrix	500x500	N. of hidden layers	2
N. of Convolutional layer	1	Output layer	1
N. of Pooling layer	1	N. of neuron nodes in input layer	250
Function of Pooling layer	Max-pooling	N. of neuron nodes in output layer	2
Number of filter	15	N. of neuron nodes in first hidden layer	125
Size of filter	4,7	N. of neuron nodes in second hidden layer	63
Padding	0	Activation function	sigmoid,softmax
Activation function	ReLU	dropout	0.5
Regularizer	L2	Regularizer	L2

TABLE 6. ER and TC of each word embedding methods without applying the Hadoop framework.

Evaluation criteria	ER(%)			TC(s)		
Word embedding	Word2vec	GolVe	FastText	Word2vec	GolVe	FastText
sentiment140	20.59	35.68	11.02	13.43	7.32	20.15
COVID-19 Sentiments	15.54	21.39	8.66	08.30	4.52	12.46

fuzzy deep learning classifier. Then, we switched the word embedding adopted approaches(Word2vec, GloVe, Fast-text) to conclude the most effective approach to transform the text-based data into numerical-based data. After we obtained the most performant word embedding procedure in terms of accuracy, we have utilized it in the rest of this work.Second, we applied the efficiency word embedding technique obtained in the preceding step, and we preserved the same parameters of our proposed deep learning model(CNN+FFNN). Then, we changed the adopted fuzzification approach and the used defuzzification methods. This experiment's main objective is to determine the most effective fuzzification approach and the most efficient defuzzification method among all used methods in terms of accuracy. Third, after determining the most efficient word embedding technique, the better fuzzification method, and the most effective defuzzification approach. We formed several fuzzy deep learning classifiers (FDLC) using different parameters for each layer. This experiment's primary goal is to define the set of parameters, which made our FDLC more accurate. Four, we parallelize our proposed FDLC employing the Hadoop framework with its HDFS and MapReduce programming framework. Finally, we compared the most effective model of our proposed FDLC with similar literature approaches and demonstrating our suggested FDLC's performance.

A. EXPERIMENT 1

In this experiment, we evaluated the effectiveness of the Word2vec, GloVe, Fast-text in terms of the ER, and the TC. We merged these studies word embedding methods with our suggested FDLC to prove and to verify their performance. This experiment serves to discover the more efficient one among all employed word embedding processes in the case of our work, in which we used both large datasets. So, we kept the same parameters of our proposed FDLC, and we switched the adopted word embedding method (Word2vec, GloVe, and Fast-text), Then we applied each combination on both used massive datasets as presented in Table 5.

Table 5 introduces the parameter settings of both used deep learning model CNN and FFNN. Also, for the MFS classifier, we used the Triangular membership function for performing the fuzzification and the Centroid of the Area for carrying out the defuzzification. Besides, the Hadoop framework is implemented in our work, which parallelizes the learning word embedding tasks between five machines; one master node and four slave nodes. The Hadoop framework uses its HDFS for stocking the dataset to be embedding and the set of representation vectors (the obtained result by applying the word embedding method), and MapReduce programming framework for processing and treating our work [56]. The Hadoop framework's primary goal is to parallelize our embedding process multiple machines to improve the AC and reduce the TC. Table 6 shows the ER and TC of each word embedding methods without applying the Hadoop framework.

From our experimental consequences, as revealed in Table 6, we remarked that the GolVe method takes less execution time than other techniques, which is equal to 7.32s, and 4.52s in the case of sentiment140 and COVID-19 Sentiments datasets, respectively. But it has a higher error rate, which is equal to 35.68%, and 21.39% in the case of sentiment140 and COVID-19 Sentiments datasets, respectively. Fast-text has less error rate compared to Word2vec and GloVe techniques, which is equal to 11.02%, and 8.66% in the case of sentiment140 and COVID-19 Sentiments datasets,

TABLE 7. ER and TC of each word embedding methods with applying the Hadoop framework.

Evaluation criteria	ER(%)			TC(s)		
Word embedding	Word2vec	GolVe	FastText	Word2vec	GolVe	FastText
sentiment140	15.16	27.44	8.28	1.68	0.46	3.03
COVID-19 Sentiments	10.68	14.08	5.51	0.66	0.096	1.49

TABLE 8. ER, AC, and TC of fuzzification method/defuzzification approaches without using Hadoop framework.

Fuzzification method	Defuzzification method	ER(%)	AC(%)	TC(s)
Trapezoidal MF	Centroid of Area	16.33	83.67	29.06
	Bisector of Area	21.95	78.05	31.56
	Weighted Average	24.08	75.92	25.37
	Center of Sums	12.64	87.36	35.09
Triangular MF	Centroid of Area	18.02	81.98	21.64
	Bisector of Area	25.41	74.59	28.03
	Weighted Average	24.98	75.02	22.18
	Center of Sums	14.36	85.64	27.92
Gaussian MF	Centroid of Area	13.56	86.44	23.46
	Bisector of Area	20.03	79.97	25.84
	Weighted Average	23.45	76.55	24.39
	Center of Sums	10.25	89.75	22.01

respectively. But it has a higher execution time, which is equal to 20.15s, and 12.46s in the case of sentiment140 and COVID-19 Sentiments datasets, respectively. In summary, the Fast-text embedding method is the most efficient in terms of the learning rate. And in order to overcome its execution time shortcoming, we have employed the Hadoop framework to parallelize the used embedding methods to minimize the execution time and raise the learning rate. Table 7 depicts the obtained results after the incorporation of the Hadoop framework with word embedding methods.

From Tables 6 and 7, we deduce that the Hadoop framework can decrease the execution time and raise the learning rate. For example, it reduces the Fast-text method's execution time from (20.15s,12.46s) to (3.03s,1.49s) for both datasets sentiment140, and COVID-19 Sentiments, respectively. Almost, we can say that Fast-text is the most efficient method. Therefore, we will use only the Fast-text word embedding method in the rest of this work.

B. EXPERIMENT 2

This second experiment's main goal is to find the most efficient fuzzification approach and better defuzzification methods. In this experiment, the adopted word embedding is the Fast-text, and the parameter settings of both used deep learning model CNN and FFNN are the same as those presented in Table 5. The renovated part of our proposal is in the applied fuzzy classifier. As explained earlier, In our fuzzy classifier, we use three MFs for the fuzzification process, which are Triangular, Trapezoidal, and Gaussian MFs, and four defuzzification approaches, which are Centroid of Area, Bisector of Area, Weighted Average, and Center of Sums. Therefore in this experiment, we combine each fuzzification method with different defuzzification methods, as illustrated in Table 8. This aggregation generates 12 combinations, which are Triangular MF/Centroid of Area, Triangular MF/Bisector of Area, Triangular MF/Weighted Average, Triangular MF/Center of Sums, Trapezoidal MF/Centroid of Area, Trapezoidal MF/Bisector of Area, Trapezoidal MF/Weighted Average, Trapezoidal MF/Center of Sums, Gaussian MF/Centroid of Area, Trapezoidal MF/Gaussian MF, Trapezoidal MF/Gaussian MF, and Trapezoidal MF/Gaussian MF. Table 8 presents the obtained ER, AC, and TC after applying our proposal on the Sentiment140 dataset without using the Hadoop framework.

As presented previously, in this experiment, we kept the same parameters for the deep learning model, as detailed in Table 5, and we applied the Fast-text word embedding. Therefore at each time, we changed the fuzzification/defuzzification methods in order to discover the most efficient fuzzification method /defuzzification approaches. From Table 8, we perceive that the better fuzzification method is the Gaussian MF, and the most efficient defuzzification is the Center of Sum approach compared to other approaches. Furthermore, we used the Gaussian MF to fuzzify both NSS and PSS values into veryLow, low, moderate, high, and veryHigh fuzzy sets. After we get the output of the inference engine process, we defuzzified this fuzzy output into the crisp output using the Center of Sum method. This Gaussian MF/Center of Sums aggregation raises the AC to 89.75% and decrease the ER to 10.25%; also, this combination is better in terms of consumption time that equal to 22.01s.

Table 9 describes the obtained result for the ER, AC, and TC of applying fuzzification approaches and defuzzification methods using the Hadoop framework. This step's primary

TABLE 9. ER, AC, and TC of fuzzification method/defuzzification approaches using Hadoop framework.

Fuzzification method	Defuzzification method	ER(%)	AC(%)	TC(s)
Trapezoidal MF	Centroid of Area	12.26	87.74	5.812
	Bisector of Area	17.02	82.98	6.312
	Weighted Average	18.91	81.09	5.074
	Center of Sums	8.79	91.21	7.018
Triangular MF	Centroid of Area	14.56	85.44	4.328
	Bisector of Area	19.62	80.38	5.606
	Weighted Average	17.34	82.66	4.436
	Center of Sums	9.86	90.14	5.584
Gaussian MF	Centroid of Area	7.49	92.51	4.692
	Bisector of Area	16.07	83.93	5.168
	Weighted Average	18.05	81.95	4.878
	Center of Sums	5.13	94.87	4.402

TABLE 10. Parameters settings of our proposed (CNN+FNN) deep learning model.

Parameters	value
N. of convolutional layers	1, 2, 3, 4, 5, 6, 7
N. of pooling layers	1, 2, 3, 4, 5, 6, 7
Input embedding matrix	500x500
N. of fully connected layers	1
Activation method	sigmoid,softmax,ReLU
Filter size	3, 4, 5, 7, 9, 10, 11, 12
Regularizer	L2
N. of filter	15, 45, 90, 180, 360, 270, 225, 200, 190, 183, 185, 187, 184
N. of the hidden layers	3, 4, 5, 6, 7, 8
Dropout options	0.5
Window size	256
Vocabulary size	40,000
N. of epochs	15

purpose is to demonstrate the effectiveness of applying the Hadoop framework on the fuzzification techniques and defuzzification process. Similarly, in this phase, the deep learning hybrid model keeps the same parameters as displayed in Table 5; the Fast-text approach is used as the word embedding technique, and at each time, the fuzzification approach /defuzzification methods are changed.

From Table 9, we remark that the Hadoop framework reduces the execution time of all 12 fuzzification method/ defuzzification approach. Also, it increases the AC and decreases the ER. For example, in the case of Gaussian MF/Center of Sums incorporation, the TC is reduced from 22.01s, as shown in Table 8, to 4.402s, as exhibited in Table 9. The AC is increased from 89.75%, as illustrated in Table 8, into 94.87%, as presented in Table 9. The ER is decreased from 10.25%, as described in Table 8, to 5.13%, as depicted in Table 9. Therefore, in this experiment, we have learned two remarks. First, the Hadoop framework possesses a significant ability to improve our proposed FDLC's performance. Second, the Gaussian MF/Center of Sums combination is proven its effectiveness compared to eleven other varieties. For that, we have decided to use this Gaussian MF/Center of Sums as a fuzzification approach/defuzzification method in our proposed FDLC.

C. EXPERIMENT 3

In this experiment, several FDLCs have been constructed harnessing different parameters for each layer. The parameter settings employed for our proposed deep learning (CNN+FFNN) model such as the number of convolutional layers, number of pooling layers, number of fully connected layers, number of the hidden layers, the used activation functions, filter size, regularizer, number of filters, dropout options, window size, vocabulary size and number of epochs are described in Table 10.

For evaluation of our proposal, in this experiment, we take either 1,2,3,4,5,6, or 7 convolution layers, also we take either 1,2,3,4,5,6,or 7 pooling layers, then we vary the sizes of filter such as 3×3,4×4,5×5,7×7,9×9,10×10,11×11,12×12. The number of these used filters is varied from 15 to 135. Also, we take either 3,4,5,6,7 or 8 hidden layers. For the other parameters such as embedding input matrix, dropout option, the used activation functions, regularizer, window size, vocabulary size, and the number of epochs have been made steady along with the performed changes because these parameters have not demonstrated any enhancement in the accuracy of our proposal. In all constructed FDLCs, the number of convolutional layers, number of pooling layers has been changed along with other used parameters such

TABLE 11. Parameters settings for our parallel FDLCs.

Name	N. of convolutional layers	N. of pooling layers	N. of hidden layers	N. of filters	Size of filters
FDLC1	1	1	3	15	4.5
FDLC2	1	1	4	15	5, 7
FDLC3	1	1	3	15	5, 7
FDLC4	1	1	4	15	4, 5
FDLC5	1	1	4	15	5, 7, 9
FDLC6	1	1	4	45	5, 7, 9
FDLC7	1	1	4	45	5, 7, 9, 11
FDLC8	1	1	4	90	5, 7, 9, 11
FDLC9	1	1	4	90	5, 7, 9, 10
FDLC10	1	1	4	90	5, 7, 9, 12
FDLC11	1	1	5	90	5, 7, 9, 10
FDLC12	1	1	4	180	5, 7, 9, 10
FDLC13	1	1	5	180	5, 7, 9, 10
FDLC14	1	1	6	180	5, 7, 9, 10
FDLC15	1	1	5	360	5, 7, 9, 10
FDLC16	1	1	5	270	5, 7, 9, 10
FDLC17	1	1	5	225	5, 7, 9, 10
FDLC18	1	1	5	200	5, 7, 9, 10
FDLC19	1	1	5	190	5, 7, 9, 10
FDLC20	1	1	5	183	5, 7, 9, 10
FDLC21	1	1	5	185	5, 7, 9, 10
FDLC22	1	1	5	187	5, 7, 9, 10
FDLC23	1	1	5	184	5, 7, 9, 10
FDLC24	1	1	7	183	5, 7, 9, 10
FDLC25	2	2	5	183	5, 7, 9, 10
FDLC26	2	2	7	183	5, 7, 9, 10
FDLC27	2	2	8	183	5, 7, 9, 10
FDLC28	3	3	7	183	5, 7, 9, 10
FDLC29	4	4	7	183	5, 7, 9, 10
FDLC30	5	5	7	183	5, 7, 9, 10
FDLC31	6	6	7	183	5, 7, 9, 10
FDLC32	7	7	7	183	5, 7, 9, 10

as the size of filters, number of hidden layers, number of filters, the word embedding is set to Fast-text, and the fuzzification method/defuzzification approach is set to Gaussian MF/Center of Sums method. The configuration of parameter settings of all the 18 parallel FDLCs is illustrated in Table 11.

As presented in Table 11, the difference between FDLC1 and FDLC2 is in the number of hidden layers, such as the FDLC1 has three hidden layers and FDLC2 has four hidden layers, and in the size of filters such as the FDLC1 used two filters of size 4×4 and 5×5, and FDLC2 applied two filters of size 5×5 and 7×7. From Table 12 we remark that the accuracy of FDLC1 is equal to 91.30% and those of FDLC2 is equal to 94.04% So, there is a significant amelioration for passing from FDLC1 into FDLC2.

Based on FDLC1 and FDLC2, we have been built two novel models, which are called FDLC3 and FDLC4, in which we fixed the number of hidden layers, and we varied the size of the used filters to find the causes that make FDLC2 better than FDLC1. From FDLC2 and FDLC4, we note that the filter 7×7 increases the classification rate by 0.74%. Also, From FDLC1 and FDLC4, we remark that using four hidden layers in FDLC4 instead of three hidden layers in FDLC1 raises the classification by 4.74%.

At this point, the most efficient model among the four constructed models is the FDLC2. So, based on this FDLC 2, we will build two other models, which are called FDLC5 and FDLC6. In FDLC5, we add a new filter size 9×9, and in FDLC6, we boost the number of used filters to 45, and we also append the filter size 9×9 to other used filters size. Consequently, The filter 9×9 grows the classification rate by 0.94%, and the increase in the number of used filters to 45 leads to an increase in the classification rate by 3.69%.

At this actual moment, the FDLC6 is better than all previously suggested models; likewise, we build two other models based on FDLC6, which are called FDLC7 and FDLC8. In FDLC7, we attach the filter 11×11 to different adopted filters size in FDCL6, and in FDLC8, we also add the filter 11×11, and we boost the number of used filters to 90. According to a comparative study between FDLC7 and FDLC8, we perceive that the classification rate is decreased by 0.09% when we add the filter 11×11. Therefore in the next experiment, we will use the filter 10×10 and filter 12×12 to discover the optimal filter among 9×9,10×10,11×11 and12×12. In FDLC8, the classification rate is increased by 5.24%. We conclude that the raise in the number of used filters leads to an increase in the classification rate.

TABLE 12. ER, AC and TC of different proposed parallel FDLCs.

Name	AC(%)	ER(%)	TC(s)
FDLC1	79.30	20.7	4.446
FDLC2	84.78	15.22	4.960
FDLC3	82.04	17.96	6.009
FDLC4	84.04	15.96	5.750
FDLC5	85.72	14.28	7.120
FDLC6	89.41	10.59	7.978
FDLC7	89.32	10.68	10.862
FDLC8	94.56	5.44	11.970
FDLC9	95.59	4.41	12.621
FDLC10	93.20	23.29	12.968
FDLC11	96.45	3.55	13.200
FDLC12	95.98	4.02	15.157
FDLC13	96.89	3.11	16.402
FDLC14	97.25	2.75	19.586
FDLC15	78.29	21.71	41.170
FDLC16	80.17	19.83	32.865
FDLC17	82.75	17.25	29.145
FDLC18	85.34	14.66	25.007
FDLC19	87.91	12.09	21.18
FDLC20	98.06	1.94	18.145
FDLC21	91.08	8.92	19.03
FDLC22	89.71	10.29	19.35
FDLC23	94.52	5.48	19.11
FDLC24	98.84	1.16	19.445
FDLC25	98.75	1.25	20.92
FDLC26	99.17	0.83	22.03
FDLC27	98.39	1.61	24.59
FDLC28	99.28	0.72	25.12
FDLC29	99.56	0.44	25.94
FDLC30	99.72	0.28	26.64
FDLC31	99.83	0.17	25.97
FDLC32	98.40	3.98	27.58

Presently, the most powerful model is FDLC8. Therefore, based on this FDLC8 model, we have been constructed both models FDLC9 and FDLC10. These novel models keep the same configuration as FDLC8. The only difference is the added filter size, such as in FDLC9, we remove the filter 11×11, and we adopt the filter with size 10×10, and in FDLC10, we eliminate the filter with size 11×11, and we add the filter with size 12×12. The filter 11×11 is removed because it gives rise to a decreasing in the classification performance, as presented in FDLC7. According to the obtained result, as explained in Table 12, the filter 10×10 used in FDLC9 increases the classification performance by 1.03%, and the filter 12×12 used in the FDLC10 decreases the accuracy by 0.97% compared to the FDLC8. In this experiment, we observe that the filter size, which starts by 3×3 to 10×10, gives rise to an increase in the classification rate. Still, once we arrive at a filter size greater or equal to 11×11, the classification performance starts to decrease. Thence, we deemed the size filter 10×10 as an optimal size value of our suggested FDLC.

Accordingly, to all previously performed experiments, FDLC9 is the most efficient model. Thus, based on this FDLC9 model, we form two different models: FDLC11 and FDLC12. In FDLC11, we vary the number of hidden layers, and for FDLC12, we raise the number of filters. From the empirical results, we notice that the augmentation in the number of hidden layers leads to 0.86% augmentation in the classification rate for the FDLC11 model. Also, for the model 12, we observe that the rise in the number of used filters increases the accuracy by 0.39%. Based on these results, we build a novel FDLC13 that augments the number of hidden layers and the number of employed filters at once. The empirical outcome proves that the FDLC13 achieved the best accuracy at this current time that equal to 96.89%.

Once again, we create two novel models, which are FDLC14 and FDLC15. In FDLC14, we augment the number of hidden layers to 6, and in FDLC15, we raise the number of used filters to 360. Accordingly, to the empirical results, we show that the FDLC14 increases the classification rate by 0.36% compared to FDLC13. But the augmentation in the number of used filters made in FDLC15 provokes a significant decrease in classification rate compared to FDLC13. In comparison, it reduced the accuracy from 96.89% in FDLC13 to 78.29 in FDLC15. These considerable decreases demonstrate that the optimal number of filters value is close to 180.

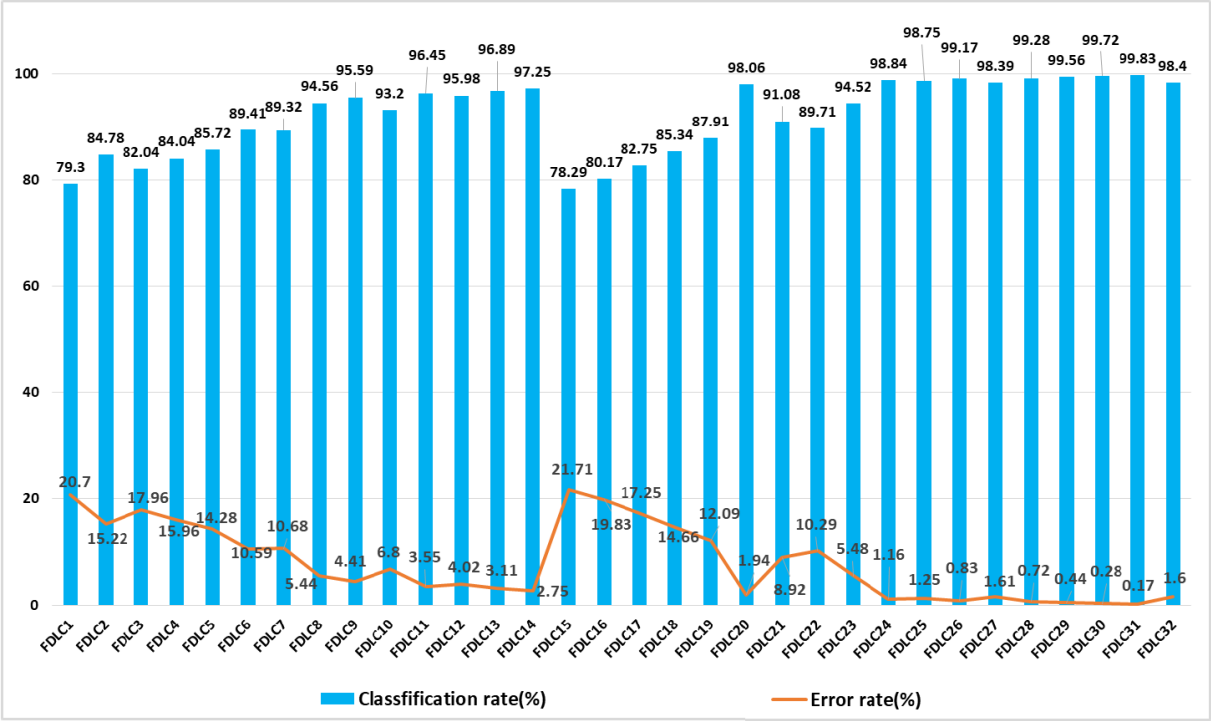


FIGURE 29. AC and ER of all evaluated FDLCs of our proposal.

Based on the previous experiment, we create four other novel models, which are FDLC16, FDLC17, FDLC18, and FDLC19. These models' objective is to find the possible interval in which the optimal value of the number of used filters exists. In the FDLC16, we used 270 as the number of adopted filters; for the FDLC17, we employed 225 as the number of employed filters; In the FDLC18, we utilized 200 as the number used filters. Finally, in the FDLC19, we used 190 as the number of applied filters. On the report of experimental results, the classification rate augments when the number of used filters is close to 180. So the optimal value will be in the interval [180;190].

To determine the optimal value, we assemble three models, which are FDLC20, FDLC21, Model 22. In FDLC20, we employed 185 as the number of filters. While, In FDLC21, the number of used filters is 183. Also, for FDLC22, the number of applied filters is 187. The preliminary outcome proved that the number of used filters' optimal value would be in the interval [183;185]. So, in the previous experiment, we computed the classification rate for 183 and those of 185. In this experiment, we will create a novel model called FDLC23, in which the number of used filters equal to 184–acquired experimental results of the FDLC20, which is utilized the 183 as a number of the used filters show that its classification rate is equal to 97.25%. It is currently the highest classification rate. So, we can deduce that the optimal value of the used filters is 183.

Based on FDLC20, we change the number of hidden layers to determine the most efficient number. To do that, we develop four models, which are FDLC24, FDLC25, FDLC26, and FDLC27. In FDLC24, we used the seven as the number of hidden layers. Also, in FDLC25, we change the number of convolutional layers to 2, and we alter the number of pooling layer to 2. Then in FDLC26, we modify at the same time the number of convolutional, pooling, and hidden layers. Also, in FDLC27, we change the number of hidden layers to eight. Then. Consequently, FDLC24 augments the classification rate by 0.78%. Thus, FDLC25 increases the classification rate by 0.67% compared to FDLC20. Hence, the accuracy in FDLC26 is raised to 99.17%. But in FDLC27 has less accuracy compared to FDLC26, which is equal to 98.39%. According to FDLC26 and FDLC27, we deduced that the optimal number of hidden layers is seven.

Based on the previous empirical results, we construct five models, in which we vary at the same time, the number of convolutional and pooling layers. These models are FDLC28, FDLC29, FDLC30, FDLC31, and FDLC32, as described in Table 11. On the report of the experiment results for the last five models, we remark that 6 is the optimal number of the convolutional and pooling layers. Therefore, according to the presented results in Table 12 and in Fig. 29, the most efficient introduced FDLC in this work is the FDLC31 with an accuracy equal to 99.83%. Fig. 30 illustrates the final architecture of our proposed fuzzy deep learning classifier after we carried out several experiments to find the optimal values in each phase.

From Table 12 and Fig. 31, we remark that our proposal has a higher classification rate but offers by higher

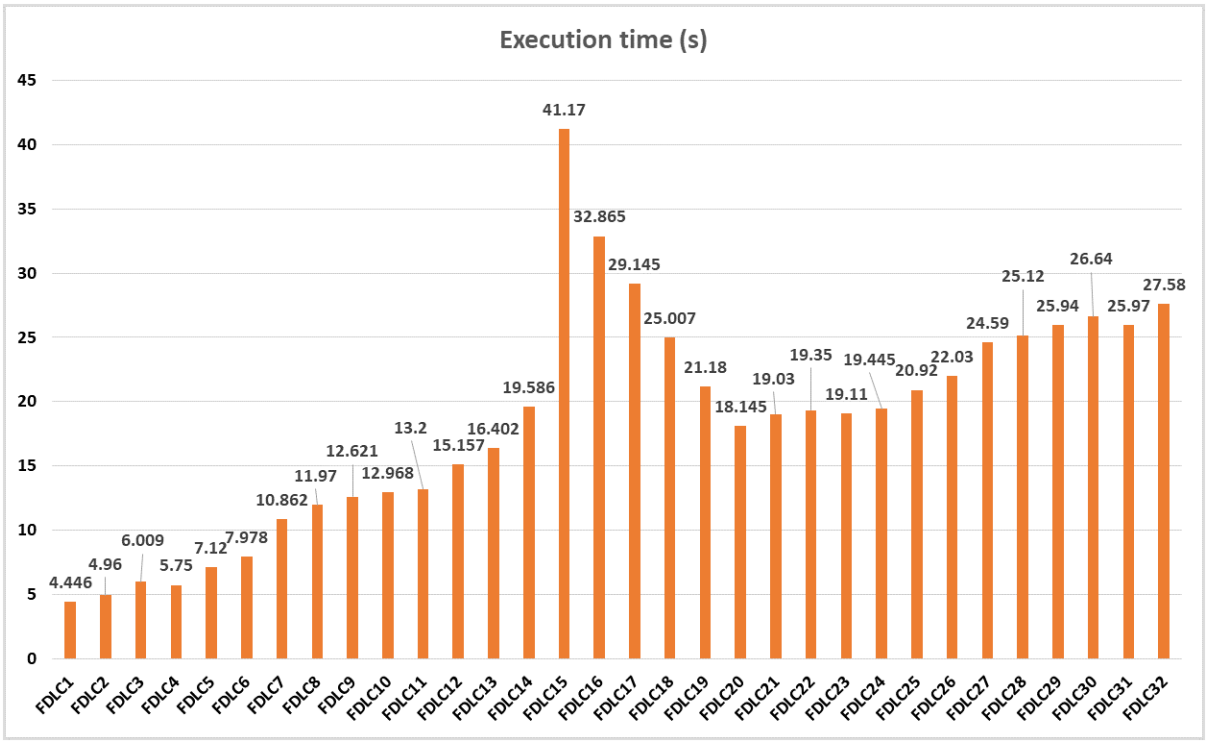


FIGURE 30. TC of all evaluated FDLCs of our proposal.

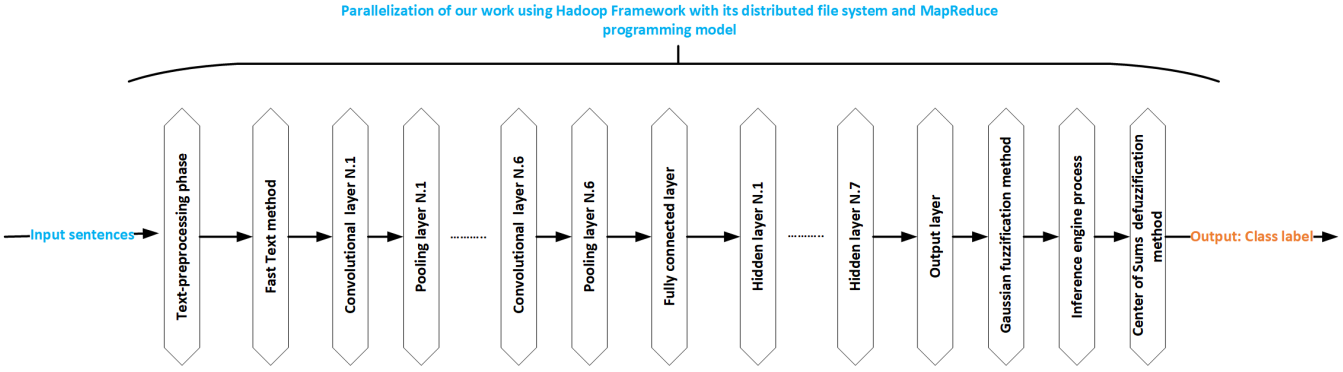


FIGURE 31. The architecture of our proposal after multiple experiments.

execution time. So, to prevent this problem, we decide to use more Hadoop computing nodes. Fig. 32 represents the experimental result after the execution of our proposal on several Hadoop computing nodes. Therefore, for this Fig. 32, we observe that the execution time decreased from 25.97s used five computing machine to 0.0089s used twelve computing machines.

D. EXPERIMENT 4

We experimented with demonstrating the obtained experimental results by applying our proposed fuzzy deep learning classifier. Its objective is to compare our proposal with multiple other methods from the literature. The selected works for these experiments are a ''Multi-task learning model based on Multi-scale CNN and LSTM for sentiment classification'' suggested by Jin *et al.* [29]. This work merges CNN and LSTM to improve sentiment analysis performance. 86.25%. a ''Stacked Residual Recurrent Neural Networks With Cross-Layer Attention for Text Classification'' proposed by Lan *et al.* [31]. This approach integrates the stacked residual RNN and cross-layer attention technique. Its objective is to capture and detect more linguistic features, thus employ them for the sentiment analysis task 89%. A ''Comparison Enhanced Bi-LSTM with MultiHead Attention (CE-B-MHA)'' developed by Lin *et al.* [32]. This paper, called CE-B-MHA, combines multi-Head attention to extracting global features and the strength of Bi-LSTM to discover the local sequence features. A ''hybrid method for

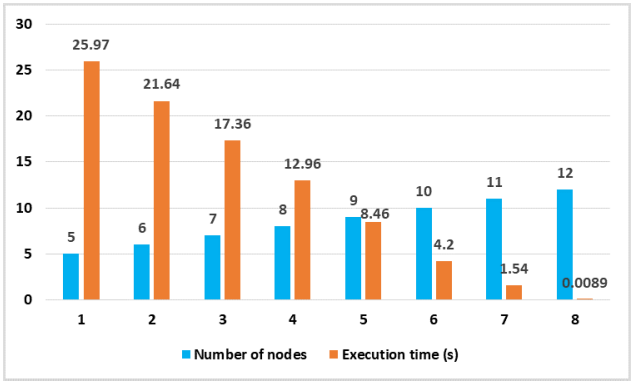


FIGURE 32. The execution time of our proposal illustrates in Fig. 31.

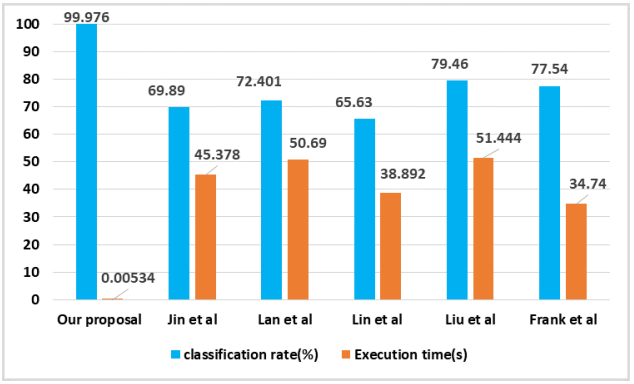


FIGURE 34. Experimental result of the comparative study carried out between our proposal and other works for the literature using COVID-19_Sentiments dataset.

bilingual text sentiment classification based on deep learning'' implemented by Liu *et al.* [33]; this suggested method integrates NB, SVM machine learning algorithm with RNN, and LSTM deep learning model. And finally, ''Intelligent asset allocation via market sentiment views'' designed by Xing *et al.* [36]; also, this proposed method combines the evolving clustering with LSTM deep learning model. Our proposal and all these chosen approaches from the literature are applied to both used datasets in this work, which are Sentiment140 and COVID-19_Sentiments datasets. Fig. 33 depicted the obtained results in terms of classification rate and consumption time by applying our proposal and other selected methods on the Sentiment140 dataset.

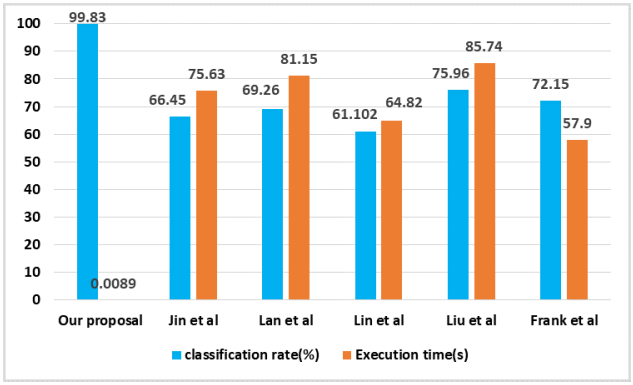


FIGURE 33. Experimental result of the comparative study carried out between our proposal and other works for the literature using Sentiment140 dataset.

Fig. 34 illustrates the obtained results after the application of our suggested FDLC and other selected techniques from the literature on the COVID-19_Sentiments dataset

From Fig. 33 and Fig. 34, we observe that our proposal based on the deep learning model (CNN+FFNN), Hadoop framework, and Mamdani fuzzy system outperforms the other used approaches (Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36]) with accuracy equal to 99.83%,99.99%, and execution time equal to 0.0089s, and 0.00534 on Sentiment140 dataset and COVID-19_Sentiments dataset respectively. Our proposal's significant effectiveness and performance are due to the application of fuzzy logic, integration of CNN and FFNN deep learning models, and the utilization of twelve computing nodes in the Hadoop cluster.

For more evaluation of our proposed fuzzy deep learning classifier, we did another experiment that compares our proposal with the other selected approaches from the literature (Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36]). But in this case, the evaluations used criterion will be TPR, FNR, TNR, FPR, PR, KS, and FS, as presented in the performance metrics subsection. This comparative is performed using both datasets, which are Sentiment140 and COVID-19_Sentiments. Its experimental results are illustrated in Table 13.

Based on the experimental results depicted in Table 13, we observe that our proposal (CNN+FNN+FuzzyLogic+Hadoop) outperforms the other selected approaches from the literature in both datasets (Sentiment140 and COVID-19_Sentiments) and at the level of TPR(99.98%, 99.81%), FNR(0.02%, 0.19%), TNR(98.61%,98.91%), FPR(1.39%, 1.09%) PR(98.55%,97.75%), KS(97.96%,98.96%) and FS(98.35%,96.54%)

E. EXPERIMENT 5

In this last experiment, we have evaluated the effectiveness of our proposed FDLC in terms of complexity, convergence, and stability. This experiment serves to compare our proposed FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] and to discover the more efficient method among all evaluated approaches in terms of complexity, convergence, and stability.

F. COMPLEXITY

The complexity of a model is a measurement of the time consumption and space used by a model. in this subsection we evaluated the time complexity and space complexity of our proposed FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36]. Additionally, Table 14 presents the experimental results obtained after

TABLE 13. The Experimental outcome of TPR, FNR, TNR, FPR, PR, KS, and FS. of the our FDLC,Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] approaches.

		TPR	FNR	TNR	FPR	PR	KS	FS
COVID-19_Sentiments dataset	Our FDLC31	99.98	0.02	98.61	1.39	98.55	97.96	98.35
	Jin *et al.* [29]	70.89	29.11	69.08	30.92	70.85	67.49	71.14
	Lan *et al.* [31]	72.64	27.36	71.51	28.49	69.54	70.66	69.20
	Lin *et al.* [32]	65.76	34.24	61.49	38.51	58.27	57.37	62.81
	Liu *et al.* [33]	78.64	21.36	76.98	23.02	74.23	72.15	69.45
	Xing *et al.* [36]	76.45	23.55	73.26	26.74	42.72	50.52	62.42
Sentiment140 dataset	Our FDLC31	99.81	0.19	98.91	1.09	97.75	98.96	96.54
	Jin *et al.* [29]	67.74	32.26	68.41	31.59	69.18	71.87	66.94
	Lan *et al.* [31]	69.32	30.68	69.91	30.80	70.29	72.16	68.42
	Lin *et al.* [32]	61.304	30.696	62.04	37.96	69.12	65.25	67.59
	Liu *et al.* [33]	75.87	24.13	74.05	25.95	70.92	72.62	69.33
	Xing *et al.* [36]	73.19	27.81	75.31	24.96	72.29	75.14	72.48

TABLE 14. Space complexity of the our FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] approaches.

Datasets	Algorithms	No. operations	No. parameters
COVID-19_Sentiments	Our FDLC	42M	21.76M
	Jin *et al.* [29]	79.5M	51.10M
	Lan *et al.* [31]	98M	46.5M
	Lin *et al.* [32]	57.37M	27.80M
	Liu *et al.* [33]	102M	70.11M
	Xing *et al.* [36]	45M	29.64M
Sentiment140	Our FDLC	102M	47.82M
	Jin *et al.* [29]	203.75M	107.21M
	Lan *et al.* [31]	294M	79.5M
	Lin *et al.* [32]	130.13M	63.4M
	Liu *et al.* [33]	306M	150.33M
	Xing *et al.* [36]	105.78M	56.43M

we measure the space complexity of our FDLC model and other chosen models in terms of the number of executed operations, and the number of the network parameters.

From Table 14, we note that our proposed FDLC has performed multiple operations with a size equal to (42M,102M) for COVID-19_Sentiments and Sentiment140 dataset, respectively. The size of our FDLC parameters is equal to (21.76M,47.82M) for COVID-19_Sentiments and Sentiment140 dataset, respectively. As the experimental result shown, our proposed FDLC requires much lower space computational complexity compared to Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] approaches.

Table 15 shows the empirical results obtained after measuring the time computational complexity of our FDLC model and other chosen models in terms of training time consumption and testing time consumption.

From Table 15, we observe that our proposed FDLC has consumed a training time equal to (0.00534s,0.0089s) for COVID-19_Sentiments and Sentiment140 dataset, respectively. The testing time of our FDLC model is equal to (0.00178s,0.00287s) for COVID-19_Sentiments and Sentiment140 dataset, respectively. As the empirical result described, our proposed FDLC requires much lower time computational complexity compared to Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] approaches.

G. CONVERGENCE

Our proposed FDLC will be proved if it is convergent or not according to the specific number of learning iterations, which is useful to control the time-consuming. The following formula specifies the condition of the convergent

TABLE 15. Time complexity of the our FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] approaches.

Datasets	Algorithms	Training time	Testing time
COVID-19_Sentiments	Our FDLC	0.00534s	0.00178s
	Jin *et al.* [29]	45.378s	16.50s
	Lan *et al.* [31]	50.69s	18.10s
	Lin *et al.* [32]	38.892s	13.84s
	Liu *et al.* [33]	51.444s	17.148s
	Xing *et al.* [36]	34.74s	12.86s
Sentiment140	Our FDLC	0.0089s	0.00287s
	Jin *et al.* [29]	75.63s	26.17s
	Lan *et al.* [31]	81.15s	27.05s
	Lin *et al.* [32]	64.82s	23.15s
	Liu *et al.* [33]	85.74s	31.75s
	Xing *et al.* [36]	57.9s	18.67s

trend:

$$Error_{former} - Error_{current} \geq T \tag{40}$$

where $Error_{former}$ is our FDLC average error of the previous training iteration, $Error\_current$ is our FDLC average error of the present training iteration, and T is the threshold that determinate the convergence rate value and after multiple experiments, we set this threshold value to 0.0001.

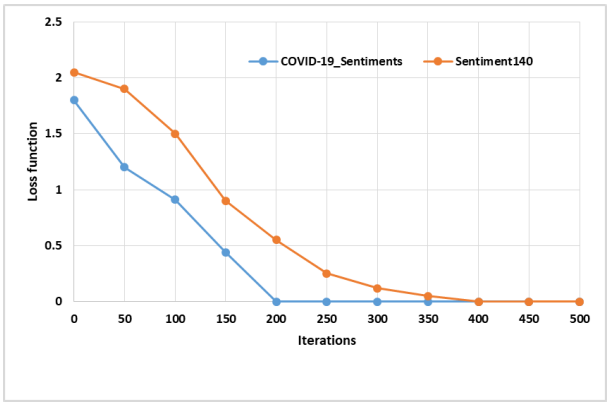


FIGURE 35. Convergence rate of our proposed FDLC in both used datasets.

Our FDLC average error is computed as follows:

$$Error = \frac{1}{2} * \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} (c - c_{label})^2}{N} \tag{41}$$

where N is the total number of sentences in the given dataset, M is the total number of FDLC output units, c is the required output class label, and c_{label} the obtained output class label. If the equation (40) is met, our proposed FDLC can be deemed convergent, and the algorithm is trained until the FDLC's average error satisfied the condition. Otherwise, our proposed FDLC is not convergent. Fig. 35 shows the convergence rate of our proposed FDLC in both used datasets. From Fig. 35, we note that our proposed FDLC converged towards the threshold value 0.0001 after our FDLC reached the iterations 200, and 400 for COVID-19_Sentiments and Sentiment140 dataset, respectively.

TABLE 16. Convergence rate of our FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] approaches.

Datasets	Algorithms	iterations
COVID-19_Sentiments	Our FDLC	200
	Jin *et al.* [29]	1020
	Lan *et al.* [31]	1425
	Lin *et al.* [32]	800
	Liu *et al.* [33]	1289
	Xing *et al.* [36]	500
Sentiment140	Our FDLC	400
	Jin *et al.* [29]	1523
	Lan *et al.* [31]	2008
	Lin *et al.* [32]	1264
	Liu *et al.* [33]	1749
	Xing *et al.* [36]	712

TABLE 17. Stability of the our FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] over five cross-validations.

Datasets	Algorithms	AVA (%)	MSD (%)
COVID-19_Sentiments	Our FDLC	99.97	0.18
	Jin *et al.* [29]	69.54	2.45
	Lan *et al.* [31]	73.12	2.27
	Lin *et al.* [32]	64.80	3.52
	Liu *et al.* [33]	79.04	1.49
	Xing *et al.* [36]	77.62	1.79
Sentiment140	Our FDLC	99.81	0.23
	Jin *et al.* [29]	66.78	2.84
	Lan *et al.* [31]	70.43	2.68
	Lin *et al.* [32]	62.25	3.76
	Liu *et al.* [33]	76.52	2.09
	Xing *et al.* [36]	74.38	2.35

Table 16 represents the convergence rate of our FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] approaches. From Table 16, we deduce that our proposed FDLC converges very fast compared to other approaches.

H. STABILITY

We determine if our FDLC is stable or not by computing the mean standard deviation (MSD) corresponding to the classification-based models' accuracy on the different

five cross-validations of the given dataset. Our FDLC is trained with the same hyper-parameters and configurations but with different five cross-validations dataset. Table 17, shows the obtained average accuracy (AVA) and mean deviation standard of our FDLC, Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] in the five cross-validations of the both used datasets in this work.

From Table 17, we note that our FDLC is practically always capable of achieving higher average accuracy with a very low mean standard deviation. This suggests that our FDLC is more stable than other algorithms.

VIII. CONCLUSION

The diversity in social media platforms leads to massive usage by the personals, and they deem these platforms as an efficient tool of communication. Therefore the feedback of users on these platforms has generated the big sentiment analysis data to learn. At present, NLP, Hadoop framework, and deep learning models provide a set of tools that aim to capture and detect the expressed users' sentiments in the collected massive datasets from social media platforms. In this work, a novel parallel fuzzy deep learning classifier is developed; This classifier incorporates NLP text-preprocessing methods, NLP word embedding approaches, CNN+FFNN deep learning model, and Mamdani fuzzy system. This proposal's primary goal is to determine a significant relationship between word embedding approaches and both used deep learning models (CNN+FNN). Also of its objective is to deal with ingrained ambiguities in data by applying the Mamdani fuzzy system. The proposed classifier is parallelized using the Hadoop framework for avoiding the long-running problem and improve the classification rate.

In our parallel fuzzy deep learning classifier, we proposal a new structure that works with pre-processing technique, word embedding algorithms such as FastText, Word2vec, and GolVe under FFNN, CNN and MFS algorithms. Furthermore, the first step of our work is the application of text pre-processing for reducing the noisy data, after that we have applied the word embedding method to transform the text based-data to numerical based data, then we employ the CNN deep learning model to detect and extract features from the obtained embedding matrix in the previous step. In addition we have used the FFNN to compute the NSS and PSS, finally we applied the Mamdani fuzzy approaches to deal with ingrained ambiguities for NSS and PSS values.

Six experiments were performed to demonstrate the effectiveness of our developed classifier. In the first experiment, we have executed our approach with and without text pre-processing techniques, and we deduce that the application of text pre-processing methods reduce the error rate for the classification of Sentiment140 from 35.59% to 5.98% and it reduces from 29.04% to 3.61% in the COVID-19 Sentiments dataset

In the second experiment, we have evaluated different used word embedding methods (i.e., FastText, GolVe, and Word2Vec). The experimental result shows that Fast-text has less error rate compared to Word2vec and GloVe techniques, which is equal to 8.28%, and 5.51% in the case of sentiment140, and COVID-19 Sentiments datasets, respectively.

In the third experiment, we carried out a comparative study between twelve aggregation fuzzification approach /defuzzification methods to find the most efficient fuzzification method and the better defuzzification approach. These aggregation methods are MF/Centroid of Area, Triangular MF/Bisector of Area, Triangular MF/Weighted Average, Triangular MF/Center of Sums, Trapezoidal MF/Centroid of Area, Trapezoidal MF/Bisector of Area, Trapezoidal MF/Weighted Average, Trapezoidal MF/Center of Sums, Gaussian MF/Centroid of Area, Gaussian MF/ Bisector of Area, Gaussian MF/ Weighted Average, and Gaussian MF/ Center of Sums. This experiment shows that the Gaussian MF/Center of Sums method raises the classification rate to 89.75% and decreases the error rate to 10.25%. Also, this combination is better in terms of execution time that equals 22.01s.

In the fourth experiments, 32 deep learning models were built, harnessing different parameters for each layer. The most efficient FDLC model is the FDLC31, which consists of a text preprocessing phase, one embedding layer, six convolutional layers, six pooling layers, 183 filters, 5×5, 7×7, 9×9, 10×10 size of filters, one fully connected layer, seven hidden layers, one output layer, Gaussian fuzzification method, inference engine process, and Sum of center defuzzification process. This FDLC31 has achieved an accuracy equals to 99.97% if applied to the COVID-19_Sentiments dataset, and it equals to 99.83%. if the FDLC31 is used on the Sentiment140 dataset

In the fifth, we have compared our proposed FDLC31 with Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] We deduce that our proposal outperforms all these used approaches in terms of TPR, TNR, FPR, FNR, ER, PR, AC, KS, FS, and TC.

Finally, we have evaluated the performance our proposed FDLC31 with Jin *et al.* [29], Lan *et al.* [31], Lin *et al.* [32], Liu *et al.* [33], and Xing *et al.* [36] in terms of complexity, convergence, and stability. We reveal that our approach outperforms all other approaches in terms of speed convergence, much lower computational complexity and it is more stable.

Our future work is the combination of our approach with the wireless sensor networks. The main goal of these future work is to classify the collected data by sensor nodes, taking into consideration multiple parameters associated with feature detection and extraction and data aggregation.

REFERENCES

[1] M. Anagha, R. R. Kumar, K. Sreetha, and P. C. Reghu Raj, "Fuzzy logic based hybrid approach for sentiment analysisl of malayalam movie reviews," in *Proc. IEEE Int. Conf. Signal Process., Informat., Commun. Energy Syst. (SPICES)*, Feb. 2015, pp. 1–4, doi: 10.1109/SPICES.2015.7091512.

[2] J. B. Sathe and M. P. Mali, "A hybrid sentiment classification method using neural network and fuzzy logic," in *Proc. 11th Int. Conf. Intell. Syst. Control (ISCO)*, Jan. 2017, pp. 93–96, doi: 10.1109/ISCO.2017.7855960.

[3] M., Biltawi, W. Etaiwi, S. Tedmori, A. Shaout, "Fuzzy based sentiment classification in the Arabic language," in *Intelligent Systems and Applications* (Advances in Intelligent Systems and Computing), vol. 868, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-01054-6_42.

[4] U. Kumari, A. K. Sharma, and D. Soni, "Sentiment analysis of smart phone product review using SVM classification technique," in *Proc. Int. Conf. Energy, Commun., Data Anal. Soft Comput. (ICECDS)*, Aug. 2017, pp. 1469–1474, doi: 10.1109/ICECDS.2017.8389689.

[5] Y. Thein and T. N. Khin, "Comparing SVM and KNN algorithms for Myanmar news sentiment analysis system," in *Proc. 6th Int. Conf. Comput. Data Eng. (ICCDE)*, New York, NY, USA: Association for Computing Machinery, 2020, , pp. 65–69, doi: 10.1145/3379247.3379293.

[6] S. N. Singh and T. Sarraf, "Sentiment analysis of a product based on user reviews using random forests algorithm," in *Proc. 10th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2020, pp. 112–116, doi: 10.1109/Confluence47617.2020.9058128.

[7] W. P. Ramadhan, S. T. M. T. Astri Novianty, and S. T. M. T. Casi Setianingsih, "Sentiment analysis using multinomial logistic regression," in *Proc. Int. Conf. Control, Electron., Renew. Energy Commun. (ICCREC)*, Sep. 2017, pp. 46–49, doi: 10.1109/ICCEREC.2017.8226700.

[8] S. Chakraborty, A. Biswas, B. Bose, and S. Tiwari, "Sentiment analysis of review datasets using naive Bayes and K-NN classifier," in *Proc. IJIEEB*, Kolkata, India, vol. 8, 2016, pp. 54–62, doi: 10.5815/ijieeb.2016.04.07.

[9] F. Es-Sabery and A. Hair, "An improved ID3 classification algorithm based on correlation function and weighted attribute*," in *Proc. Int. Conf. Intell. Syst. Adv. Comput. Sci. (ISACS)*, Dec. 2019, pp. 1–8.

[10] A. Severyn and A. Moschitti, "Twitter sentiment analysis with deep convolutional neural networks," in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Aug. 2015, pp. 959–962, doi: 10.1145/2766462.2767830.

[11] A. Vassilev, *BowTie—A Deep Learning Feedforward Neural Network for Sentiment Analysis*. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2019, doi: 10.6028/NIST.CSWP.04222019.

[12] D. Li and J. Qian, "Text sentiment analysis based on long short-term memory," in *Proc. 1st IEEE Int. Conf. Comput. Commun. Internet (ICCCI)*, Oct. 2016, pp. 471–475, doi: 10.1109/CCI.2016.7778967.

[13] M. Kuta, M. Morawiec, and J. Kitowski, "Sentiment analysis with tree-structured gated recurrent units," in *Text, Speech, and Dialogue*, K. Ekstein, and V. Matousek, Eds. Cham, Switzerland: Springer, 2017, pp. 74–82, doi: 10.1007/978-3-319-64206-2_9.

[14] M. Al-Smadi, O. Qawasmeh, M. Al-Ayyoub, Y. Jararweh, and B. Gupta, "Deep recurrent neural network vs. Support vector machine for aspect-based sentiment analysis of arabic hotels' reviews," *J. Comput. Sci.*, vol. 27, pp. 386–393, Jul. 2018, doi: 10.1016/j.jocs.2017.11.006.

[15] S. M. Rezaeinia, R. Rahmani, A. Ghodsi, and H. Veisi, "Sentiment analysis based on improved pre-trained word embeddings," *Expert Syst. Appl.*, vol. 117, pp. 139–147, Mar. 2019, doi: 10.1016/j.eswa.2018.08.044.

[16] Y. Li and B. Shen, "Research on sentiment analysis of microblogging based on LSA and TF-IDF," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Dec. 2017, pp. 2584–2588, doi: 10.1109/CompComm.2017.8323002.

[17] N. Cummins, S. Amiriparian, S. Ottl, M. Gerczuk, M. Schmitt, and B. Schuller, "Multimodal bag-of-words for cross domains sentiment analysis," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 4954–4958, doi: 10.1109/ICASSP.2018.8462660.

[18] B. Liu, Y. Zhou, and W. Sun, "Character-level hybrid convolutional and recurrent neural network for fast-text categorization," in *Proc. ELM*, J. Cao, C. M. Vong, Y. Miche, and A. Lendasse, Eds. Cham, Switzerland: Springer, 2020, pp. 108–117, doi: 10.1007/978-3-030-23307-5_12.

[19] S. Modha, T. Mandl, P. Majumder, and D. Patel, "Tracking hate in social media: Evaluation, challenges and approaches," in *SN Comput. Sci.*, vol. 1, p. 105, Mar. 2020, doi: 10.1007/s42979-020-0082-0.

[20] B. L. Devi, V. V. Bai, S. Ramasubbareddy, and K. Govinda, "Sentiment analysis on movie reviews," in *Emerging Research in Data Engineering Systems and Computer Communications*, P. V. Krishna and M. S. Obaidat, Eds. Singapore: Springer, 2020, pp. 321–328, doi: 10.1007/978-981-15-0135-7_31.

[21] R. Bose, R. K. Dey, S. Roy, and D. Sarddar, "Sentiment Analysis on Online Product Reviews," In: M. Tuba, S. Akashe, and A. Joshi, (eds.) *Inf. Commun. Technol. for Sustain. Develop.*, pp. 559–569. Springer, Singapore (2020), doi: 10.1007/978-981-13-7166-0_56.

[22] A.-S. Uban and L. P. Dinu, "On transfer learning for detecting abusive language online," in *Advances in Computational Intelligence*, I. Rojas, G. Joya, and A. Catala, Eds. Cham, Switzerland: Springer, 2019, pp. 688–700, doi: 10.1007/978-3-030-20521-8_57.

[23] H. Rosa, N. Pereira, R. Ribeiro, P. C. Ferreira, J. P. Carvalho, S. Oliveira, L. Coheur, P. Paulino, A. M. V. Simão, and I. Trancoso, "Automatic cyberbullying detection: A systematic review," *Comput. Hum. Behav.*, vol. 93, pp. 333–345, Apr. 2019, doi: 10.1016/j.chb.2018.12.021.

[24] D. Sharma, M. Sabharwal, V. Goyal, and M. Vij, "Sentiment analysis techniques for social media data: A review," in *Proc. 1st Int. Conf. Sustain. Technol. Comput. Intell.*, A. K. Luhach, J. A. Kosa, R. C. Poonia, X.-Z. Gao, and D. Singh, Eds. Singapore: Springer, 2020, pp. 75–90, doi: 10.1007/978-981-15-0029-9_7.

[25] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, p. 292, Mar. 2019.

[26] C. Beer, "Fuzzy thinking: The new science of fuzzy logic. Bart Kosko," *Quart. Rev. Biol.*, vol. 70, no. 2, p. 210, 1995.

[27] L. A. Zadeh, "Fuzzy logic = computing with words," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 2, pp. 103–111, May 1996.

[28] F. Es-sabery and A. Hair, "A MapReduce C4.5 decision tree algorithm based on fuzzy rule-based system," *Fuzzy Inf. Eng.*, pp. 1–28, Jun. 2020, doi: 10.1080/16168658.2020.1756099.

[29] N. Jin, J. Wu, X. Ma, K. Yan, and Y. Mo, "Multi-task learning model based on multi-scale CNN and LSTM for sentiment classification," *IEEE Access*, vol. 8, pp. 77060–77072, 2020, doi: 10.1109/ACCESS.2020.2989428.

[30] F. Almeida and G. Xexéo, "Word embeddings: A survey," 2019, *arXiv:1901.09069*. [Online]. Available: http://arxiv.org/abs/1901.09069

[31] Y. Lan, Y. Hao, K. Xia, B. Qian, and C. Li, "Stacked residual recurrent neural networks with cross-layer attention for text classification," *IEEE Access*, vol. 8, pp. 70401–70410, 2020, doi: 10.1109/ACCESS.2020.2987101.

[32] Y. Lin, J. Li, L. Yang, K. Xu, and H. Lin, "Sentiment analysis with comparison enhanced deep neural network," *IEEE Access*, vol. 8, pp. 78378–78384, 2020, doi: 10.1109/ACCESS.2020.2989424.

[33] G. Liu, X. Xu, B. Deng, S. Chen, and L. Li, "A hybrid method for bilingual text sentiment classification based on deep learning," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, May 2016, pp. 93–98, doi: 10.1109/SNPD.2016.7515884.

[34] A. Jain and V. Jain, "Sentiment classification of Twitter data belonging to renewable energy using machine learning," *J. Inf. Optim. Sci.*, vol. 40, no. 2, pp. 521–533, Feb. 2019, doi: 10.1080/02522667.2019.1582873.

[35] A. Yenter and A. Verma, "Deep CNN-LSTM with combined kernels from multiple branches for IMDb review sentiment analysis," in *Proc. IEEE 8th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2017, pp. 540–546, doi: 10.1109/UEMCON.2017.8249013.

[36] F. Z. Xing, E. Cambria, and R. E. Welsch, "Intelligent asset allocation via market sentiment views," *IEEE Comput. Intell. Mag.*, vol. 13, no. 4, pp. 25–34, Nov. 2018.

[37] J.-À. González, F. Pla, and L.-F. Hurtado, "ELiRF-UPV at SemEval-2017 task 4: Sentiment analysis using deep learning," in *Proc. 11th Int. Workshop Semantic Eval. (SemEval-)*, 2017, pp. 723–727, doi: 10.18653/v1/s17-2121.

[38] G. Wang, J. Qiao, J. Bi, W. Li, and M. Zhou, "TL-GDBN: Growing deep belief network with transfer learning," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 874–885, Apr. 2019, doi: 10.1109/TASE.2018.2865663.

[39] Z. Hameed and B. Garcia-Zapirain, "Sentiment classification using a single-layered BiLSTM model," *IEEE Access*, vol. 8, pp. 73992–74001, 2020, doi: 10.1109/ACCESS.2020.2988550.

[40] L. Han and C. Mihaela, "Fuzzy rule based systems for interpretable sentiment analysis," in *Proc. 9th Int. Conf. Adv. Comput. Intell.*, Feb. 2017, pp. 129–136.

[41] K. Wu, M. Zhou, X. S. Lu, and L. Huang, "A fuzzy logic-based text classification method for social media data," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 1942–1947, doi: 10.1109/SMC.2017.8122902.

[42] S. Vashishtha and S. Susan, "Fuzzy rule based unsupervised sentiment analysis from social media posts," *Expert Syst. Appl.*, vol. 138, Dec. 2019, Art. no. 112834, doi: 10.1016/j.eswa.2019.112834.

[43] M. Abdul-Jaleel, Y. H. Ali, and N. J. Ibrahim, "Fuzzy logic and genetic algorithm based text classification Twitter," in *Proc. 2nd Sci. Conf. Comput. Sci. (SCCS)*, Mar. 2019, pp. 93–98, doi: 10.1109/SCCS.2019.8852607.
[44] G. Wang, Q.-S. Jia, J. Qiao, J. Bi, and C. Liu, "A sparse deep belief network with efficient fuzzy learning framework," *Neural Netw.*, vol. 121, pp. 430–440, Jan. 2020, doi: 10.1016/j.neunet.2019.09.035.
[45] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1746–1751.
[46] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics Intell. Lab. Syst.*, vol. 39, no. 1, pp. 43–62, 1997, doi: 10.1016/S0169-7439(97)00061-0.
[47] E. Pourjavad and R. V. Mayorga, "A comparative study and measuring performance of manufacturing systems with mamdani fuzzy inference system," *J. Intell. Manuf.*, vol. 30, no. 3, pp. 1085–1097, Mar. 2019, doi: 10.1007/s10845-017-1307-5.
[48] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Netw.*, vol. 1, no. 2, pp. 119–130, 1988, doi: 10.1016/0893-6080(88)90014-7.
[49] Y. Chen. (Aug. 2015). *Convolutional Neural Network for Sentence Classification*. UWSpace. Accessed: Dec. 29, 2020. [Online]. Available: http://hdl.handle.net/10012/9592
[50] *The Most Popular Research, Guides, News and More in Artificial Intelligence*. Accessed: Dec. 29, 2020. [Online]. Available: https://deepai.org/
[51] H. Yousuf and S. Salloum, "Survey analysis: Enhancing the security of vectorization by using word2vec and CryptDB," *Adv. Sci., Technol. Eng. Syst. J.*, vol. 5, no. 4, pp. 374–380, 2020.
[52] *COVID-19_Sentiments India[20/03/20–31/05/20]*. Accessed: Dec. 30, 2020. [Online]. Available: https://kaggle.com/abhaydhiman/covid19-sentiments
[53] V. Gangadharan, D. Gupta, L. Amritha, and T. A. Athira, "Paraphrase detection using deep neural network based word embedding techniques," in *Proc. 4th Int. Conf. Trends Electron. Informat. (ICOEI)*, Jun. 2020, pp. 517–521.
[54] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543, doi: 10.3115/v1/D14-1162.
[55] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013, *arXiv:1310.4546*. [Online]. Available: http://arxiv.org/abs/1310.4546
[56] F. Es-Sabery and A. Hair, "Big data solutions proposed for cluster computing systems challenges: A survey," in *Proc. 3rd Int. Conf. Netw., Inf. Syst. Secur.*, Mar. 2020, pp. 1–7, doi: 10.1145/3386723.3387826.

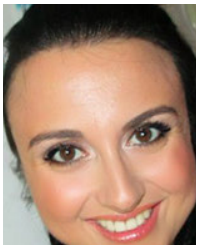
FATIMA ES-SABERY received the technical University degree from the Department of Computer Science, Higher School of Technology, Casablanca, Morocco, in 2013, the professional license with option IT development from the Department of Computer Sciences, Faculty of Science, Casablanca, Morocco, in 2014, and the master's degree in business intelligence from the Department of Computer Sciences, Sultan Moulay Sliman's University, Beni Mellal, Morocco, in 2016. She has published several research papers in many international conferences and journals, i.e., *International Journal of Fuzzy Information and Engineering*, *International Journal of Informatics and Communication Technology*, The 3rd International Conference on Networking, Information Systems & Security, and 2019 International Conference on Intelligent Systems and Advanced Computing Sciences. Her general research interests include data mining area, big data field, wireless sensor networks, fuzzy systems, machine learning, deep learning, and the Internet of Things.

ABDELLATIF HAIR currently works as a Full Professor with the Computer Department of FST of Beni Mellal (Morocco), and a member of LAMSC Laboratory. His research interests include object-oriented analysis/design, security of mobile agents, wireless sensor network (WSN), data warehousing, and machine learning (ML).

JUNAID QADIR received the M.Sc. degree in electronics from the Department of Electronics, University of Peshawar, in 2016, and the M.Phil. degree in electronics from Quaid-i-Azam University, Islamabad, Pakistan, in 2019. He is currently pursuing the Ph.D. degree with the Department of Electrical, Electronic and Telecommunications Engineering, and Naval Architecture (DITEN), University of Genova, Italy. Also, he is a Research Collaborator with the Department of Signal Theory, Communications and Telematics Engineering, University of Valladolid, Spain. He has published many research articles in highly reputed international journals and conferences, such as the IEEE Access, *MDPI-Energies*, *Journal of Intelligent & Fuzzy System*, International Multitopic Conference (INMIC), and in CISIS 2018, Kunibiki Messe, Matsue, Japan. His current research interests include underwater wireless sensor networks, wireless sensor networks, the Internet of Things, mobile edge computing (MEC), machine learning, and 5G mobile communication. He is a verified reviewer with a number of prestigious international publishers, such as the IEEE Access, the IEEE Sensors Journal, *International Journal of Distributed Sensor Networks* (IJDSN), *Journal of King Saud University* (Elsevier), *Computer Methods and Programs in Biomedicine* (Elsevier), *Heliyon*, IEEE International Wireless Communications and Mobile Computing Conference (IEEE IWCMC 2019), Network Modeling and Analysis in Health Informatics & Bioinformatics, and Acta Acustica united with Acustica *Journal of the European Acoustics Association* (EAA).

BEATRIZ SAINZ-DE-ABAJO received the Ph.D. degree *(summa cum laude)* from the University of Cordoba, in 2009. She is currently an Associate Professor in Telecommunications Engineering with the University of Valladolid, Spain. Her fields of action are the development and evaluation of e-Health systems, m-Health, medicine 2.0., cloud computing, among others. She focuses on topics related to electronic services for the information society. She belongs to the GTe Research Group, integrated within the UVa Recognized Research Group "Information Society." Among the lines of research, the group works to develop innovative solutions in the field of health that help patients improve their quality of life and facilitate the work of health professionals.

BEGOÑA GARCÍA-ZAPIRAIN (Member, IEEE) graduated in telecommunications engineering and specialized in telematics at the University of the Basque Country (UPV/EHU), Leioa, Spain. She received the Ph.D. degree *(summa cum laude)* in the pathological speech digital processing field in 2003, and the Executive MBA degree from the University of the Basque Country, in 2011, with the Best Student Award. In 2012, she graduated from the Advanced Program in Health Management at the Deusto Business School, Deusto University, Bilboa, Spain. After spending five years at ZIV Company, she joined the Engineering Faculty at the University of Deusto, in 1997, as a Lecturer in Signal Theory and Electronics, where she led the Telecommunications Department from 2002 to 2008. She received the Accesit to the Ada Byron Award to the Technologist Woman 2015. In recognition of the quality of its research activities, the research group she leads won the Research Award 2007 UDGrupo Santander, the ONCE Euskadi Solidarity Award 2007, the award for the best article in the Games 2009 International Congress, the prize for the best poster at ISIVC 2008, and was the finalist for the Social Innovation in Ageing - The European Award 2014.



ISABEL DE LA TORRE DIEZ is currently a Professor with the Department of Signal Theory and Communications and Telematic Engineering, University of Valladolid, Spain. She is also the Leader of the GTe Research Group. Her research interests include design, development, and evaluation of Telemedicine applications, services and systems, e-health, m-health, EHRs (Electronic Health Records), EHRs standards, biosensors, cloud and fog computing, data mining, QoS (Quality of Service), and QoE (Quality of Experience) applied to the health field.