SiNoptiC: swarm intelligence optimisation of convolutional neural network architectures for text classification

Imen Ferjani*

Laboratory of Robotics, Informatics, and
Complex Systems (RISC Lab - LR16ES07),
National Engineering School of Tunis,
University of Tunis El Manar,
BP. 37, Le Belvedere, 1002, Tunis, Tunisia
Email: imene.ferjani@enit.utm.tn
*Corresponding author

Minyar Sassi Hidri

Computer Department,
Deanship of Preparatory Year and Supporting Studies,
Imam Abdulrahman Bin Faisal University,
P.O. Box 1982, Dammam 31441, Saudi Arabia
Email: mmsassi@iau.edu.sa

Ali Frihida

Laboratory of Robotics, Informatics, and
Complex Systems (RISC Lab - LR16ES07),
National Engineering School of Tunis,
University of Tunis El Manar,
BP. 37, Le Belvedere, 1002, Tunis, Tunisia
Email: ali.frihida@enit.utm.tn

Abstract: Although many rules have been suggested by several researchers for designing deep neural architectures, trial-and-error is often exploited in practice to find the optimal model for a given problem. Thus, the automation of deep neural architecture search methods is highly recommended. In this work, we address this problem by proposing a hybrid coupling of Convolutional Neural Networks (CNNs) architectures with the swarm intelligence, especially the Fish School Search (FSS) algorithm. This coupling is capable of discovering a promising architecture of a CNN on handling text classification tasks. The proposed method allows users to provide training data as input, and receive a CNN model as an output. It is completely automatic and capable of fast convergence. Computational results show the effectiveness of the proposed method in achieving the best classification loss among manually designed CNNs. This is the first work using FSS for automatically designing the architectures of CNNs.

Keywords: deep learning; CNN; convolutional neural networks; swarm intelligence; FSS; text classification; NLP.

Reference to this paper should be made as follows: Ferjani, I. and Sassi Hidri, M. and Frihida, A. (2022) 'SiNoptiC: swarm intelligence optimisation of convolutional neural network architectures for text classification', *Int. J. Computer Applications in Technology*, Vol. 68, No. 1, pp.82–100.

Biographical notes: Imen Ferjani is a Doctoral student in Information and Communication Science and Technology at the National Engineering School of Tunis (ENIT), University of Tunis El Manar, Tunisia. She received Masters of Computer Science degree from the College of Science of Tunis in 2012. Her doctoral research is about applying deep learning to NLP.

Minyar Sassi Hidri received her PhD degree from the National Engineering School of Tunis, Tunisia in 2007. She is an Assistant Professor at the Imam Abdulrahman Bin Faisal University (Dammam, Saudi Arabia) since 2017. She is an Associate Professor at the Computer Department of the National Engineering School of Tunis (Tunis El Manar University, Tunisia) since November 2018. Her research interests mainly focus on combinatorial aspects in Big Data

analytics, Data mining, Machine Learning, Deep Learning, and Text Mining, with over 65 publications. She is currently member of the steering committee of many international conferences and reviewer of impacted journals.

Ali Frihida received his PhD degree in GeoSpatial Informatics from University of Montreal. He is a Senior Lecturer in National Engineering School of Tunis, University of Tunis El Manar (Tunisia). He is actually the Head of the Information and Communication Department. Her research interests include ontologies, big (spatial) data, AI applied to spatial data, IoT and NLP.

1 Introduction

Convolutional Neural Networks (CNNs) are biologically-inspired algorithms that have achieved excellent performance on many challenging tasks (Zhang et al., 2020; Zhu et al., 2020). The performance of a CNN is largely due to its capacity to find out complex structure and non-linear relationships within data. CNNs have achieved interesting results in many tasks in the computer vision field and have been effectively explored for text classification (Yann et al., 2015; Moitra and Mandal, 2020; Corbat et al., 2020; Ferjani et al., 2019). It is an emerging field of study that has been widely addressed in several real applications (Jiang et al., 2018).

The topology of a CNN is one of the most important aspects that affect its performance. Over the last decade, numerous CNN topologies have been proposed with the target of either improving the accuracy or reducing the computational complexity (Kim, 2014; Kalchbrenner et al., 2014; Johnson and Zhang, 2017; Zhang et al., 2015; Conneau et al., 2016).

Several approaches to designing Deep Learning (DL) architectures based on optimisation techniques and heuristic search (Ammar et al., 2020) have been proposed. Such techniques can either derive their inspiration from the natural evolution of the biological organisms (Fogel, 1995) or they can be built on the behavioural models of living organisms such as fish, ants, bees and birds. Such methods are called Swarm Intelligence (SI) algorithms (Kennedy, 2006).

Swarm intelligence refers to the collective behaviour of decentralised and self-organised systems inspired by the natural or biological behaviour (Kennedy, 2006). SI systems consist typically of a population of simple agents interacting locally with one another and with the environment. As a result of such interaction, an *intelligent* global behaviour unknown to the individual agents will emerge. Fish schooling, birds flocking, ant colonies, hawks hunting, animal herding and bacterial growth are examples of swarm intelligence in natural systems. The characteristics of a decentralised and flexible way of working make swarm intelligence a successful design paradigm for algorithms that deal with increasingly complex problems such as optimisation problems.

Many algorithms have been proposed to optimise CNNs. Some of them focused on hyper-parameters search (Jin et al., 2019; Lorenzo et al., 2017; Serizawa and Fujita, 2020). Other works optimised the weights of neural networks such as Khalifa et al. (2017) and Wang et al. (2019). However, it is hard to balance the trade-off between efficiency and effectiveness of automatically designing CNN. Many works suffered from the lack of architecture search and the huge computational requirements. Therefore, we develop a novel swarm-based optimisation algorithm to make a balance between architecture and hyperparameters optimisation with a reasonable complexity (Hidri, 2017).

The aim of this paper is to design and develop an effective and efficient SI-based method to automatically design the structures and parameters of deep CNNs without manual intervention. To achieve this goal, a new version of the Fish School Search (FSS) algorithm was designed. The proposed method, called SiNoptiC, is based on self-adjust exploration and exploitation modes. It will be validated and evaluated with a very well-known and widely used problem: Text Classification and compared with six state-of-the-art architectures on five widely-used data sets. Our major contributions are as follows:

- A novel algorithm that combines the strengths of the FSS algorithm and deep CNN to explore the search space of neural network architectures and their associated hyperparameters to be applied.
- The use of a variable-length encoding strategy virtually with no size limitation to allow Fishes to represent deep CNNs.
- A novel individual movement operator is defined that allows fishes to make a random change of their architectures in order to explore the research space which surrounds them.
- A novel collective movement operator is presented that can be used to allow a fish to move towards or spread away from the architecture representing the barycentre of the school. This collective operator allows us to regulate the school's exploration ability during the optimisation process

This paper is organised as follows: an overview of baseline CNN architectures and DL optimisation methods is presented in Section 2. In Section 3, we will present a detailed background about FSS and CNN. Then, we will explain our motivation in Section 4. A detailed description of the proposed algorithm is presented in Section 5. The experimental protocol and results of the proposed algorithm are shown in Section 6. Finally, the conclusion and future work are detailed in Section 7.

2 Related work

2.1 Baseline CNN architectures

Kim (2014) proposed a simple CNN operating on word-level and using a single convolution layer for text classification. Kim's model showed that a shallow architecture can outperform many existing models. However, its inability to model long-distance dependencies in the sentence stands as the main issue.

Following this work, Kalchbrenner et al. (2014) presented a convolutional architecture called the Dynamic Convolutional Neural Network (DCNN) that they adopt for the semantic modelling of sentences. Their model used Dynamic *k*-Max Pooling, a global pooling operation over linear sequences, and capable of handling varying length input sentences to capture short and long-range relations.

Johnson and Zhang (2017) studied deepening of word-level CNNs to capture global representations of text and proposed a model with 15 weight layers, called Deep Pyramid CNN (DPCNN). In their model, the down-sampling strategy was used with a fixed number of feature maps to reduce the computation time for convolution layers.

Zhang et al. (2015) introduced an alternative character-based model with six convolution layers, having kernels of different sizes (3 and 7) as well as simple max-pooling layers, followed by three fully connected classification layers. They found that considering the input text as a sequence of characters improves the model performance and does not require knowledge about the structure of a language.

Conneau et al. (2016) took a further step by introducing a deep char-level CNN using up to 29 layers with small convolutions and pooling operations.

The most crucial step of using convolutional architectures on text classification tasks is designing the best classification model. However, without a complete understanding of the characteristics of the problem domain, it remains not trivial to effectively decide the exact nature and order of layers, the number of filters in convolutional layers, the number of units in dense layers and other variables in the creation of deep neural networks. The assignment of these variables is the cornerstone of the success or failure of any CNN architecture which motivates the automation of their design.

Several studies have been developed to find a theoretical basis that can help in designing deep architectures and finding a compromise between depth and efficiency.

Montufar et al. (2014) studied the complexity of certain classes of function computable by deep feed-forward neural networks. They showed that deep networks have more representational power of these functions than shallow architectures.

In 2001, the universal approximation theorem was presented by Csáji (2001), which states that to approximate any function a single hidden layer is sufficient, but this costs an exponential number of neurons, making it often impossible in computing terms. To overcome this problem, Delalleau and Bengio (2011) suggested that deep networks offer a much more compact representation of a function at a reduced cost (Wang and Raj, 2017) when compared to shallow ones.

In Bengio et al. (2013), it was empirically verified that complex tasks require deep neural networks to guarantee computational efficiency. Many other studies highlight the importance of the depth factor in the regularisation of the network learning capacity (Szegedy et al., 2015). Despite numerous studies, the truth is that there are no analytical procedures to design the appropriate deep architecture for a given problem, and trial and error are often used instead.

2.2 Swarm intelligence optimisation of CNNs architectures: a brief review

Recently, the design of Convolutional Neural Network architectures has largely shifted from trial and error processes to automatic methods. But most of the proposed architectures are designed specifically to solve particular problems and their generalisation to other fields requires expertise even if they have the same architecture. In this section, we will review some relevant works based on Swarm Intelligence optimisation of convolutional neural networks.

Khalifa et al. (2017) combined two different optimisation algorithms in a ConvNet architecture with seven layers for handwritten digit classification; they used Particle Swarm Optimisation (PSO) algorithm for the last layer which is the output vector and Stochastic Gradient Descent (SGD) algorithm for the first six layers. They reported an accuracy improvement over a standard CNN that uses an SGD optimisation algorithm in all layers.

Lorenzo et al. (2017) proposed an approach that aims to automatically discover a more appropriate network structure with a better configuration of hyper-parameters for the final training of the neural network. They combine the PSO algorithm and the steepest gradient descent algorithm to optimise the hyperparameters (learning rate, dropout rate, momentum, weight decay and the number of neurons in each hidden layer) by designing a representation of the parameters that encode the configuration of the network as a real number vector for efficient processing of the individuals of PSO in the search process.

Wang et al. (2018) proposed a PSO modelling strategy to optimise the structure of a deep CNN for image classification problems. Their approach was inspired by how the Network IP address works. In order to facilitate the numerical convergence of their algorithm, they used the IP address format to represent the large number of parameters included in a CNN, with integers values within a certain range by distributing it into some smaller integers below 256.

Junior and Yen (2019) presented a PSO-based algorithm for optimising CNN architectures for image classification with the use of a direct encoding and a novel definition of difference and velocity operators. Similarly, Wang et al. (2019) proposed cPSO-CNN for optimising the hyper-parameter configuration of CNN architectures. cPSO-CNN utilises a confidence function to enhance the canonical PSO's exploration capability with the redefinition of the scalar acceleration coefficients of PSO as vectors to better adapt for the variant ranges of CNN hyper-parameters.

Serizawa and Fujita (2020) proposed a method of using linearly decreasing weights for PSO, which is one of the meta-heuristic algorithms, for hyper-parameter CNN optimisation. Another similar work was proposed in Jiang (2020) where a multi-objective particle swarm optimisation based on decomposition was used.

In summary, we have mentioned some relevant works on the application of SI methods for the automatic design of CNNs while minimising human interventions. Moreover, SI methods show as promising approaches in solving the problem of the automatic design of CNN architectures under the absence of analytic procedures for such problems.

3 Basic concepts of CNN and FSS algorithm

3.1 Standard CNN layers

A standard CNN architecture for text classification includes four basic components: embedding, convolution, pooling and fully connected (see Figure 1). In addition, another different regulatory layer which is dropout is also incorporated. The number of layers as well as the manner they are arranged is the most important part of the design of a CNN architecture and has a strong impact on obtaining improved performance. In the following sections, we briefly describe the role of each layer in a CNN architecture for text classification.

3.1.1 Embedding

The embedding layer provides a dense vector representation of a text from the vocabulary by mapping each character, word, or sentence to an embedding dimension vector of real numbers (Sivakumar and Rajalakshmi, 2021). It is an improvement over traditional encoding methods such as bag-of-words where each word was represented by a large sparse vector depending upon the size of vocabulary it is dealing with.

The pre-trained word embeddings such as word2vec and Glove (Pennington et al., 2014; Goldberg and Levy, 2014) have been widely used as inputs to deep neural models. On the other hand, some simple and efficient models which can directly learn task-specific word embeddings or fine-tune on pre-trained word embeddings have been proposed recently. This work focuses on learning task-specific word embeddings during the training of CNN.

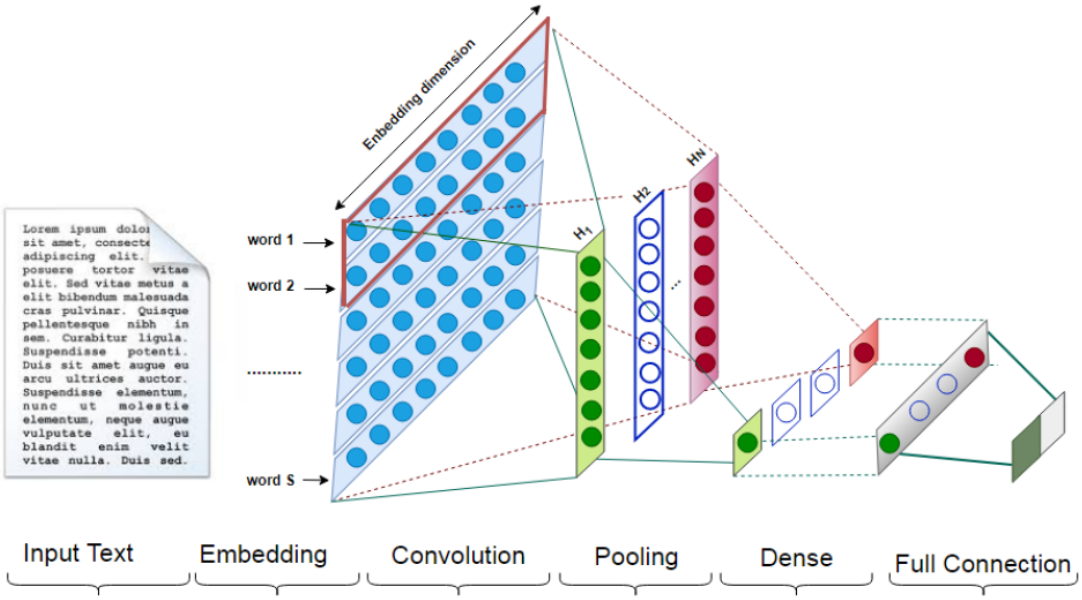
3.1.2 Convolution

The convolutional layer consists of a set of filters to perform convolutional operations on the sequences of word embeddings. The filter will be a wide rectangle with dimensions like 3×500 or 4×500 (if we have an embedding dimension of 500). The filter width is usually the same as the embedding dimension and its height or size may vary according to the number of embeddings (rows of the input embedding matrix) that will be seen at a time, similar to representing an n-gram in a word model.

During the convolutional operation, the filter vertically slides with the size of the filter kernel until the whole text sequence has been scanned. At each position, a multiplication operation in pairs is performed between the kernel weights and the embedding values followed by a summation to get a single output value. These aggregated filter outputs form a new vector known as a feature map.

Thus, the convolution operation can be viewed as a detector of features or patterns in sequential word groupings that indicate traits like the sentiment of a text, the grammatical function of different words and so on. In a convolutional layer, multiple filters are allowed to coexist, producing a set of feature maps.

Figure 1 A standard CNN architecture for text classification



3.1.3 Pooling

The pooling layers are a way to down-sample the incoming feature vectors from the convolution layer. In the case of the max-pooling layer, which is the most common pooling method, only the maximum value in a feature vector, which should be the most useful local feature, will be kept by the network. The concatenation of the max-values produced by processing each of the convolution feature vectors is used to generate the sentence representation.

3.1.4 Fully connected

A fully connected layer is in principle the same as the traditional Multi-Layer Perceptron (MLP) neural network. It performs a global operation by taking input from the previous layer and globally analysing the output of all the preceding layers. Specifically, it 'flattens' the output of the previous layers by turning them into a single vector that can be an input for the next layer. Then, it applies weights to predict the correct label and finally gives probabilities for each label.

3.1.5 Dropout

Dropout refers to ignoring units (i.e. neurons) during the training phase of a certain set of neurons that are chosen at random. More technically, at each training stage, individual nodes are either dropped out of the net with a certain probability, so that a reduced network is left, or incoming and outgoing edges to a dropped-out node are also removed. It is used to make a regularisation within the network by improving its generalisation and preventing over-fitting.

3.2 FSS optimisation

FSS (Bastos Filho et al., 2008) is a nature-inspired algorithm based on the concept of populations. The core idea is to make the fishes perform local searches and the school aggregates social information. The search is performed in a bounded search space. Each fish k is represented by a position within the search space, $x_k(t)$, which represents a candidate solution to the optimisation problem, and its corresponding weight is $W_k(t)$.

Feeding is a fundamental concept in the FSS, it updates the weight of each fish and reflects its level of success during the individual movement within the current iteration. There are three movement operators: individual, collective-instinctive, and collective-volitive. The individual movement is used to trigger the other operators. In this operator, each fish randomly chooses a new position in its neighbourhood following the equation (1):

$$x_k(t+1) = x_k(t) + rand(-1,1)step_{ind} \tag{1}$$

where:

- $x_k(t)$ and $x_k(t+1)$ are the positions of fish k before and after the individual movement respectively.
- $rand(-1,1)$ is an uniformly distributed random numbers array with the same dimension as $x_k(t)$ and values varying from -1 up to 1.
- $step_{ind}$ is the individual step. It is predefined prior to the search process. In general, it decays linearly along the iterations.

The new fish position $x_k(t+1)$ is only approved if there is an improvement of its fitness after the change. Otherwise, the position remains unchanged and $x_k(t+1) = x_k(t)$. After the individual movement, the feeding operator is executed, i.e. updating the weight W_k of fish k using equation (2):

$$W_k(t+1) = W_k(t) + \frac{\Delta f_k}{\max\left(\left|\Delta f_k\right|\right)} \tag{2}$$

where:

- Δf_k is the difference between the fitness of the neighbour position $f\left(x_k(t+1)\right)$ and the current position $f\left(x_k(t)\right)$.
- $\max\left(\left|\Delta f_k\right|\right)$ represents the maximum absolute value of fitness variation among all fishes in the school.
- W_k is only allowed to vary from 1 up to W_{scale}. All fishes are initialised with a weight equal to the value $W_{scale/2}$.

The collective-instinctive component of the movement is the average of individual movements for all x_k. A vector I representing the weighted average of displacements for each x_k is calculated according to equation (3):

$$I = \frac{\sum_{k=1}^{N} \Delta x_k \Delta f_k}{\sum_{k=1}^{N} \Delta f_k} \tag{3}$$

where N is the size of the school and Δx_k is the displacement of the fish k generated by the individual movement. The displacement represented by I is defined in a way that fishes with a higher improvement will attract other fishes to its position. After computing I, every fish moves following the equation (4):

$$x_k(t+1) = x_k(t) + I \tag{4}$$

The collective-volitive movement is used to determine the final position of all fish school. First, the barycentre is calculated according to equation (5):

$$B = \frac{\sum_{k=1}^{N} x_k(t) W_k(t)}{\sum_{k=1}^{N} W_k(t)} \tag{5}$$

Then, depending on the increase or decrease of the total school weight, the swarm is contracted or dilated towards or

outwards the barycentre of the school according to equations (6) and (7), respectively.

$$x_k(t+1) = x_k(t) - step_{vol} rand(0,1) \frac{x_k(t) - B(t)}{dist\left(x_k(t), B(t)\right)} \quad (6)$$

$$x_k(t+1) = x_k(t) + step_{vol} rand(0,1) \frac{x_k(t) - B(t)}{dist\left(x_k(t), B(t)\right)} \quad (7)$$

where:

- $step_{vol}$ represents the steps of the volitive movement.
- $dist\left(x_k(t), B(t)\right)$ is the Euclidean distance between the school barycentre B and fish k position.
- $rand(0,1)$ is a normal rand vector with the same dimension as B and values varying from 0 up to 1.

4 Motivation

In Section 2, we have reviewed some works about optimising CNNs using Swarm Intelligence. According to the study carried out in Baldominos et al. (2019), the research area studying the optimisation of deep architectures has been increasingly larger since 2017 and many works testing different methods have been proposed.

Training a neural network involves using a training data set to update the model weights in a way that minimises the error between training labels and the network predicted outputs, the cross-entropy loss is often used with CNNs. The training process is based on an optimisation algorithm in which the weights which maximise the performance of the model on the training data set are chosen. In general, the gradient descent and back-propagation (LeCun et al., 1998) are often used for minimisation. Owing the complexity of the gradient calculation which is usually computationally intensive and requires powerful hardware to perform the entire training in a reasonable time frame, the number of works studying the optimisation of deep CNNs is still very scarce.

The good news is that there is a remarkable advancement over the past few years in hardware devices (such as GPUs-graphic processor units- or TPU-Tensor Processing Unit (Dean and Hölzle, 2017)) which speed up the process of iterating over different models and topologies since the most consuming computation power tasks (forward propagation and backward propagation) are notably accelerated with the use of these devices. Additionally, some deep learning primitives such as NVIDIA's cuDNN (Chetlur et al., 2014) and frameworks such as TensorFlow (Abadi et al., 2016) or PyTorch (Paszke et al., 2016) are developed to make training and testing thousands of CNNs feasible. This was our main motivation to explore the field of deep architecture optimisation.

Also, deep networks have been demonstrated to be capable of achieving remarkable performance in NLP tasks such as the online Skype translator, Google Spam filters or Netflix. However, an overall analysis of the existing optimisation methods for DL architectures in Section 2 shows that image processing tasks have received more attention than the text processing field over the last few decades. Our aim is to extend architecture optimisation techniques to unexplored domains such as text classification.

While some research on Evolutionary Computing-based approaches such as Genetic Algorithms has reached a significant degree of advancement for traditional ANNs, the number of works using Swarm Intelligence meta-heuristic is still small, yet growing. Thereby, the development of swarm intelligence-based algorithms that can automatically create and evaluate CNN architectures is important.

An overall analysis of the Swarm Intelligence-based optimisation methods shows that the underlying idea behind all SI algorithms is similar, and various SI algorithms differ only in their details. Particle Swarm Optimisation (PSO) has been widely used while many other SI algorithms such as Fish School Search (FSS) optimisation may also be explored to evolve deep architectures, especially CNNs. Our main motivation to use the FSS algorithm is the use of the search operators: the core idea is to evolve CNN architectures individually toward better efficiency. Collectively, the best architectures have more influence on the search process as a whole, which makes the possible solutions move toward better architectures in the search space over the iterations.

In summary, considering the large use of CNNs in many fields, the automation of their architectures design is a very promising area that must be given a big interest. Another important field is SI optimisation which has gained significant attention in the previous few years due to its simplicity and fast convergence. In addition to current advances in computation technology which facilitate the development of new works within this line of research.

5 FSS and CNN coupling-based optimisation for text classification

In this section, we firstly present an overview of the proposed solution with various aspects that contribute to the solution, next we detail the main steps of our algorithm. In the next sections, we explain our encoding strategy of CNN architecture as well as the individual and the collective movement associated with a CNN which plays central roles in our solution and algorithm.

5.1 Model overview

The core idea of the proposed algorithm is the use of FSS optimisation to explore the search space of deep neural network architectures. A set of CNN architectures is considered as the fish school which will swim by applying individual and collective movements to achieve a collective goal that is discovering the best architecture for a given data set. The inputs of our proposed algorithm are parameters related to the initialisation of the school of CNN architectures, such as the minimum and maximum numbers of layers, parameters related to the FSS algorithm such as the school size and the number of iterations, and parameters referring to the text classification task, such as the training data.

The Algorithm 1 shows the framework of the proposed algorithm. The first step is initialising the school with the predefined school size where each fish encodes a particular architecture of the CNN using the proposed encoding strategy to encode the predefined building blocks. Then, all fishes perform an individual movement followed by an evaluation of the fitness of each fish after this movement. After that, the best fish is selected based on fitness. Then, all fishes perform a collective movement. Specifically, the barycentre of the school is calculated and fishes will either expand or contract from the barycentre based on the global fitness improvement of the school. The evolution continues until the number of iterations is reached.

Algorithm 1: SiNoptiC.

Data: School size (N), number of FSS iterations ($iter$), training data (X), number of classes ($nb_{classes}$), number of epochs (e)
Result: The best CNN architecture found

$S = \{F_1, F_2, ..., F_N\} \leftarrow InitSchool(N, nb_{classes})$;
for $i = 1$ *to* $iter$ **do**
  for $j = 1$ *to* N **do**
    $F_j.IndividualMovement()$
  end
  $BestFish \leftarrow UpdateBestFish(S)$
  $B \leftarrow ComputeBarycenter(S)$
  for $j = 1$ *to* N **do**
    $F_j.CollectiveMovement(B)$
  end
  $BestFish \leftarrow UpdateBestFish(S)$
end
$BestFish \leftarrow FullTrain(BestFish, X, e)$
return $BestFish.loss, BestFish.accuracy$

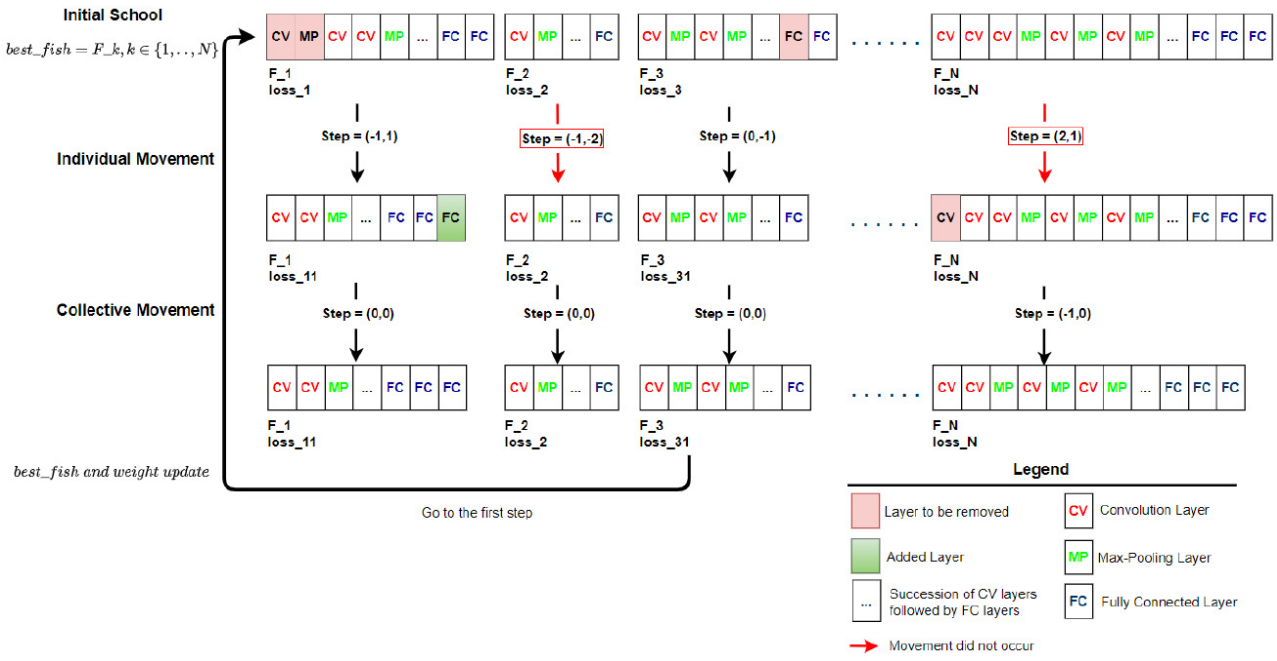
As shown in Figure 2 which presents a detailed illustration of a single iteration of our algorithm and the corresponding main steps. The proposed algorithm follows the standard pipeline of FSS (the phases of individual and collective movements). Note that, all used operators such as barycentre are redefined to fit a CNN architecture optimisation process.

Our algorithm includes five modules allowing it to find the optimal CNN architecture: a flexible CNN encoding, initialisation of a school, fitness evaluation of individual fishes, individual and collective movements that allow fishes to update their architectures. The corresponding algorithms of these modules are presented in detail in the following subsections.

5.2 CNN modelling

As introduced in Sub-section 3.1, a standard CNN is composed of the convolution layers, pooling layers and fully connected layers. Designing an algorithm that deals with such structures highly relies on the encoding strategy used. In the proposed encoding strategy, we use a variable-length array which offers better flexibility than the fixed-length encoding since it enables the proposed algorithm to automatically find the promising CNN architecture without any restriction in depth. For example, if the depth of the CNN providing the highest classification accuracy is 20, the proposed encoding strategy could enable candidates architectures to achieve this depth during the optimisation process, no matter what was their initial depth. The variable-length encoding strategy is the key to enabling the proposed algorithm to automatically optimise a CNN architecture without any domain knowledge from the users. In our encoding strategy, a direct-encoding representation of the parameters associated with each layer is performed. That is, every layer of information in the CNN is specified directly and explicitly in the encoding array.

Figure 2 SiNoptiC algorithm



The parameters of a convolution layer are the number of output filters and the size of the convolution window. In addition, the pooling layers used in the proposed encoding strategy are max-pooling layers without padding. So the only parameter encoded for these layers is the size of the max pooling window. For the fully-connected layers, the number of neurons is encoded.

An example of the proposed encoding strategy representing a CNN is illustrated in Table 1, where CV, MP and FC stand for convolution, max-pooling and fully-connected layers, respectively. This CNN is composed of three layers: convolution – max-pooling – fully connected. The list encoding this CNN is composed of the parameters representing each layer.

Table 1 CNN layers encoding example

CV: Convolution	*MP: MaxPooling*	*FC: Fully Connected*
Type: Convolution	Type: Pooling	Type: Fully connected
Output filters: 178	Pooling size: 3	Neurons: 200
Kernel size: 5		

5.3 School initialisation

In our algorithm, fishes dynamics was based on initialisation, movement of the initial set using individual and collective movements. Thus, the initialisation of the school ($InitSchool()$) is an important step in the proposed algorithm. The school was initialised with a set of N fishes representing a set of CNN architectures.

Each architecture has a random depth, between three and $(Depth_{\max})$ (the upper bound of the initial number of layers) which limits the depth only in the initialisation step. For the convenience of the discussions, each architecture is composed of three main parts starting with the embedding layer which is a common layer for all fishes, the second part is composed of the convolution and pooling layers and the last part is the fully connected layers.

In general, the CNN architectures for text classification start with an embedding layer. In our work, we train our word (and character) embeddings during the training of the CNN without using any pre-trained word embeddings. The second part takes the sequence of embedding vectors as input and can only be added after the first part. In this part, we use blocks of layers where each block is either composed of one convolution layer followed by one max-pooling layer or only one convolution layer. The choice of inserting a max-pooling layer after a convolution layer is made randomly. In fact, we do not require an alternation of convolution and pooling layers, from which each architecture can contain two or more successive convolution layers. It is important to notice that in case two or more convolution layers are stacked together, the compatibility of inputs and outputs between successive layers is ensured by defining ranges of possible values for each.

The last part, which contains a succession of fully connected layers, takes as input data by flattening all elements of the second part of feature maps.

Usually, the convolution and the pooling layers can be stacked together after the embedding layer, while the fully connected layers are stacked with each other at the tail of the architecture. Typically, fully connected layers take the deep representation from the convolution and pooling layers and transform it into the final output classes or class scores. It is not common to insert fully-connected layers between convolution and pooling layers because it makes the entire training process of CNN inefficient as well as time-consuming. This is due to the increase in the number of parameters of the overall CNN.

The Algorithm 2 lists the major steps of the school initialisation, where the initial depth is randomly generated. 70% of this depth is allocated to convolution blocks $(F_i.CV)$ and 30% is allocated to fully connected layers ($F_i.FC$). Line 5 shows the generation of the first part of a given fish. The function $addEmbedding()$ will add an embedding layer to a fish architecture given the embedding dimension $(embed)$.

Lines 6–8 show the generation of the second part, where the algorithm stacks the convolution and pooling layers using the function $addConvPool()$. In this step, an arbitrary selection is performed to decide whether a convolution layer is followed by a pooling layer or not. If so, the pooling size will be randomly generated.

The convolution layer has two random parameters: the number of output convolution filters from 7 up to $maps_{\max}$ and the length of the convolution window from 3 up to $k_{\max}$, where $k_{\max}$ indicates the maximum size of the convolution kernel. Lines 9–11 show the generation of the last part using the $add\_FC()$ function that will join an FC layer to the fish architecture. This layer has a number of hidden neurons generated randomly between 1 up to a maximum of $n_{\max}$. All layers use the rectified linear unit $(ReLU)$ as an activation function. Line 12 adds the last Fc layer with $nb_classes$ as output. In lines 13-14, each fish is trained on the training data and their accuracy and loss are saved.

Algorithm 2: Initialization of the school of fishes in the proposed algorithm ($InitializeSchool()$) .

Data: School size (N), Upper bound of the initial depth (D_{max}), Embedding dimension ($embed$), Maximum number of feature maps ($maps_{max}$), Maximum convolution kernel size (K_{max}), Maximum pooling size $poolSize$, Highest number of neurons in FC layers (n_{max}), Number of classes ($nb_{classes}$)
Result: A set of N fishes: $S = \{F_1, F_2, ..., F_N\}$

```
for i = 1 to N do
    F_i.depth = rand(3, D_max);
    F_i.CV = Rand(70%);
    F_i.FC = Rand(30%) ;
    F_i.Layers = addEmbedding(embed);
    for i = 1 to F_i.CV do
        F_i.Layers = addConvPool(K_max, maps_max, poolSize)
    end
    for i = 1 to F_i.FC − 1 do
        F_i.Layers = addFc(n_max)
    end
    F_i.Layers = add_FC(nb_classes);
    Train F_i.model on the training data;
    F_i.loss,F_i.accuracy = Evaluate F_i.model on
     the evaluation data;
end
return S = {F_1, F_2, ..., F_N};
```

5.4 Fitness function

The objective of the fitness function evaluation is to give a quantitative measure determining which architectures will be selected as the best solutions. Because the proposed algorithm is related to text classification tasks, we used the classification loss as a metric to assign the fitness of fishes architectures, in our case we used the cross-entropy loss function. The best architecture in our algorithm is the one having the smallest loss, no matter what are the values of the other parameters and without considering any other criteria.

Regarding this step, we have trained each CNN architecture over a training set. Before the training, each CNN is compiled based on the encoded information in the list of layers. Noting that, a dropout layer is added between each two FC layers to avoid the over-fitting problem (Ioffe and Szegedy, 2015).

Secondly, we have reduced the size of the training data during the optimisation algorithm to decrease the Synoptic execution time. Indeed, we used a sample of 50% of the whole data set which has been trained over ten epochs. The best model found will be trained on the entire data set. The Training process is performed using Adam (Kingma and Ba, 2014) and weights are initialised with Xavier (Glorot and Bengio, 2010).

5.5 Individual movement

The individual component of the movement in FSS is responsible for each fish local search looking for promising architectures in the search space, as described by equation (1). In our algorithm, each fish architecture F_i is represented by a position $(F_i.CV, F_i.FC)$ within the search space, where $F_i.CV$ represents the number of convolution blocks and $F_i.FC$ represents the number of fully connected layers.

During the individual movement, each fish will follow a random change in its architecture. This change consists of increasing or decreasing $F_i.CV$ and $F_i.FC$ by a small step according to equation (8).

$$\begin{aligned} F_i.CV(t+1) &= F_i.CV(t) + step_{ind} \\ F_i.FC(t+1) &= F_i.FC(t) + step_{ind} \end{aligned} \tag{8}$$

where:

- $F_i.CV(t+1)$ and $F_i.FC(t+1)$ are the new values of $F_i.CV(t)$ and $F_i.FC(t)$ after the individual movement.
- $step_{ind}$ defines the value of the number of layers added or removed during this movement and it is a random number varying from -2 up to 2 .

Changing the architecture of a fish after an individual movement is an important step in the Synoptic algorithm. The change happens in the module MoveRandomly() in Algorithm 3.

Algorithm 3: Individual movement of fishes in the proposed algorithm ($MoveRandomly()$).

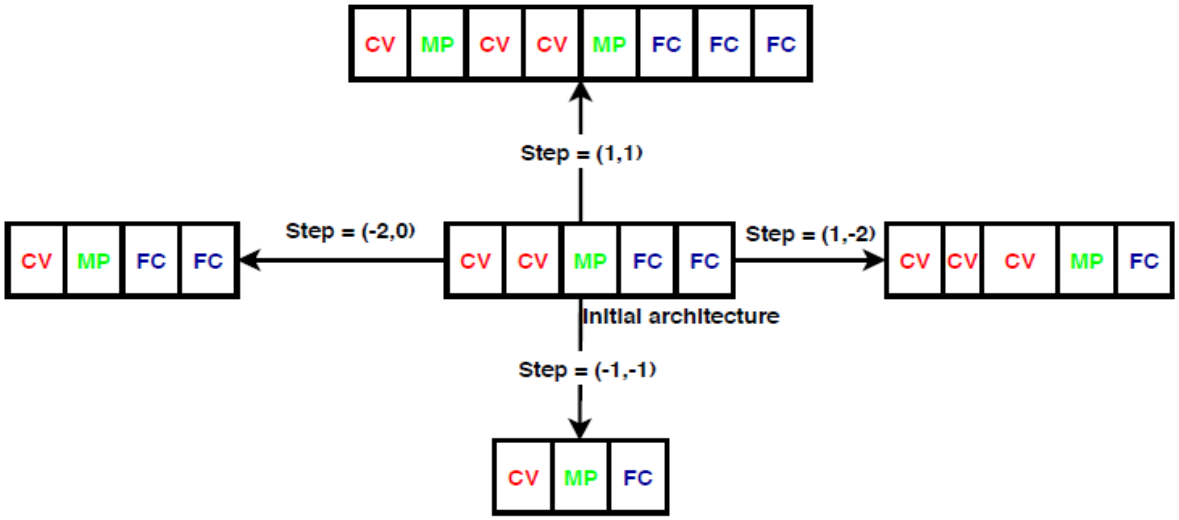
Data: School of fishes S
Result: A school of fishes S_{t+1} after individual movement

```
S = {F_1, F_2, ..., F_N};
for i = 1 to N do
    CV_step = rand(−2, 2);
    FC_step = rand(−2, 2);
    F_i(t + 1).CV = F_i.CV + CV_step ;
    F_i(t + 1).FC = F_i.FC + FC_step ;
    Train F_i(t + 1).model on the training data;
    F_i(t + 1).loss = Evaluate F_i(t + 1).model on
     the evaluation data;
    if F_i(t + 1).loss < F_i.loss then
        F_i = F_i(t + 1);
    end
end
return S = {F_1, F_2, ..., F_N};
```

The movement only occurs if there is an improvement in the fitness of fish i after the change. Otherwise, the fish keeps the same architecture.

The change of *ConvPool* part is done independently of the *FC* part by generating a random step for each part. If the step is a negative value, layers will be removed from the fish architecture.

Figure 3 Individual movement proposed showing the initial fish architecture and new architectures after different possible randomly chosen steps



Otherwise, layers will be added without defining the maximum number of layers that the fish architecture can reach. However, a finite minimum of layers is allowed. If, after removing layers, the fish ends up with layers less than that allowed, the fish architecture will be set to the minimum architecture composed of three layers. This process is illustrated in Figure 3.

5.6 Collective movement

The collective movement in the proposed algorithm is defined as the combination of both volitive and instinctive movements. We merged the two types of movements in a single movement in order to adapt them to the nature of CNN architectures. After all fishes architectures have changed individually, the list of fish's loss is computed and sorted in an ascending order to select the fishes with the lowest loss values. Selected fishes that had successful individual movements will influence later the resulting direction of the collective movement more than other ones. The number of selected fishes is proportional to the size of the school. In our algorithm, we set this proportion to 50%. The reason for choosing a percentage of the fish to contribute to the collective movement is random initialisation. Indeed, we may have a number of generated CNNs with bad loss values and we want to exclude them from the barycentre calculation to ensure an improvement of the overall loss of the school over iterations. After the selection step is performed, all fishes in the school perform a collective movement. CNNs having bad loss values will be more affected by this movement.

This movement is considered as an overall success/failure evaluation based on the incremental sum of the loss of the fish school as a whole. We refer to the loss of the school as the school weight calculated according to equation (9).

$$W = \sum_{i=1}^{N} F_i.loss \tag{9}$$

where W represents the weight, N is the size of the school and $F_i.loss$ is the loss of the fish i architecture after evaluating its fitness.

The collective movement is based on the school weight: if there is an improvement in the weight, i.e. the value of the total error decreases (which means that the search was successful), the radius of the school should contract, i.e. the fish will approach the barycentre; otherwise, it should dilate (the fish will move away from the barycentre).

This operator is supposed to greatly help improve the exploration capabilities of our algorithm. The collective movement is applied as a small change to every fish architecture concerning the school barycentre.

The fish-school barycentre is obtained by considering the best fish architectures with corresponding CV and FC layers according to equation (10).

$$B = (B_{CV}, B_{FC}) = \left(\frac{\sum_{i=1}^{N1} F_i.CV}{N1}, \frac{\sum_{i=1}^{N1} F_i.FC}{N1} \right) \tag{10}$$

where:

- B_{CV} and B_{FC} are the coordinates of the barycentre B in terms of number of convolution blocks and fully connected layers, respectively.
- $N1$ is the number of fishes to be considered when calculating the barycentre of the school (N1=N/2).
- $F_i.CV$ and $F_i.FC$ are the number of CV blocks and the number of FC layers for fish i respectively.

In our barycentre calculation, we considered that all fishes have the same coefficient (equal to 1). In each iteration, a

comparison between the previously recorded overall weight of the school and the new overall weight observed at the end of the current search cycle is made. According to whether an improvement has been found, contractions and expansions of the school will occur.

For this movement, we also defined a parameter called collective step ($step_{col}$) which is fixed to 0.5 along the iterations. The fish moves to the new position as in equation (11) if the overall weight of the school decreases in the FSS cycle; if the overall weight increases, we use equation (12).

$$\begin{aligned} F_i(t+1).CV &= F_i(t).CV - step_{col} \mid F_i(t).CV - B_{CV} \mid \\ F_i(t+1).FC &= F_i(t).FC - step_{col} \mid F_i(t).FC - B_{FC} \mid \end{aligned} \quad (11)$$

$$\begin{aligned} F_i(t+1).CV &= F_i(t).CV + step_{col} \mid F_i(t).CV - B_{CV} \mid \\ F_i(t+1).FC &= F_i(t).FC + step_{col} \mid F_i(t).FC - B_{FC} \mid \end{aligned} \quad (12)$$

The above equations show how a fish school architecture changes in collective movement towards the search objective which is finding the best architecture.

6 Computational results

6.1 Data of experimentation

The comparison of many different optimisations approaches to show the benefit of each one is highly related to the availability of large benchmark data sets (e.g. 1000 classes for ImageNet for computer vision). To evaluate the ability of our approach to competitively produce CNN architectures, we conducted our experiments on five text classification benchmark data sets, which are freely available and widely used for investigating the performance of CNN architectures. They are the AG'News (AG.N), DBPedia (DBP), Yelp Review Polarity (Yelp.P), Yelp Review Full (Yelp.F) and Yahoo answers (Yah.A) (Zhang et al., 2015).

Several classification tasks such as news categorisation, topic classification, or sentiment analysis are covered by the used data sets The number of classes is comprised between 2 and 14. The number of training examples varies from 120 k up to 3.6 M, with equal numbers of examples in each class for both training and test sets. A summary of their classes, training and test sizes is shown in Table 2. The reader is referred to Zhang et al. (2015) for more details on the construction of the data sets.

6.2 Comparison models

To evaluate the ability of our SiNoptiC algorithm to competitively produce CNN architectures, we compared our results with state-of-the-art text classification CNN models which have been introduced in Section 2.

Table 2 Data of experimentation

Data set	*#Classes*	*#Train*	*#Test*
AG's News (AG.N)	4	120 k	7.6 k
DBPedia (DBP)	14	560 k	70 k
Yelp Review Polarity (Yelp.P)	2	560 k	38 k
Yelp Review Full (Yelp.F)	5	650 k	50 k
Yahoo! Answers (Yah.A)	10	1400 k	60 k

It is important to notice that these models are not found using an optimisation algorithm as in our case. These models are manually designed for specific problems. A direct comparison is not possible in our case due to the absence of optimisation algorithms for text classification, especially the comparison of the computation complexity. Instead, we evaluate the performance of CNNs automatically found by our SiNoptiC against those designed manually on the same data sets. Although there are recent models based on CNN, we choose the first CNNs to compare our results with them (Conneau et al., 2016; Zhang et al., 2015; Kim, 2014; Johnson and Zhang, 2016, 2017). Both char-level and word-level CNNs were used for comparison in order to evaluate our algorithm for different token types of text input. Their classification results are directly cited from the original publications except for Kim's model (2014) for which we use our implementation to get results on our chosen benchmark data sets.

6.3 Experimental protocol

The parameters of the models used for comparison have been manually tuned with the use of domain expertise of their authors. Thus, none of their parameter settings needs to be specified since we directly cited their classification results from the original publications.

For Kim's model, we implemented the *CNN-rand* model and we set the parameters as they were mentioned by the author in Kim (2014). All our parameters related to CNN are based on the settings employed by the state-of-the-art CNNs. We can group the parameters used in our algorithm into three categories: CNN architecture initialisation, FSS optimisation and CNN training (see Table 3).

The parameters of the first category control the initial fishes' architectures generated randomly during the initialisation step. These parameters are:

- The lower bound of the initial depth.
- The upper bound of the initial depth.
- The highest number of neurons in FC layer.
- The lowest number of neurons in FC layer.
- The character-based convolution window size.
- The word-based convolution window size.
- The number of output filters from a convolution layer.

Table 3 List of parameters used for the fish school search algorithm

Parameter	*Value*
CNN architecture initialisation	
Lower bound of initial layers	3
Upper bound of initial layers	30
Lowest number of neurons in a FC layer	2
Highest number of neurons in a FC layer	300
Char-Conv window size: random from	[3, 5 ,7]
Word-Conv window size: random from	[3, 4]
#output filters: random choice from	$\left(2^6, 2^7, 2^8, 2^9\right)$
FSS optimisation	
School size	20
Number of iterations	15
Individual step movement	1
Collective step movement	0.5
CNN training	
Sampling percentage	50%
#epochs for fish evaluation	1
#epochs	10
Dropout rate	0.5

Initialisation is the first step in our algorithm. During this step, a number of CNN architectures with different depths and configurations will be generated. The number of layers of each fish's architecture will be randomly chosen. The highest and the lowest number of layers parameters will bound the initial fish's architecture. After the initialisation step, the fish's architecture will be changed by following our defined FSS movements with respect only to the minimum number of layers and without an upper bound depth.

In addition, these parameters also allow us to regulate the size of the search space for optimisation. Indeed, large values of the maximum depth will guide optimisation in deep architectures search space. But, the constraint here will be the size of available memory as well as the computing power used for running the algorithm.

When an FC layer is added to the fish architecture, its number of neurons will be randomly chosen between the highest and the lowest numbers of neurons in FC layers. Also, the number of feature maps that any given convolution layer can output is limited by the number of output filters. In order to ensure the compatibility of output and input dimensions between two convolution layers, the minimum value of the output filters size is always greater than the maximum value of the convolution window size. In this way, we avoid having a kernel size greater than a feature maps size when two successive convolutions happen. Our chosen values are inspired by the state-of-the-art existing models in both character and word CNNs for text categorisation. The convolutional window's size will always be a random number chosen from a predefined range of values.

The second category contains four parameters that are related to the FSS algorithm: the school size, the number of iterations and the individual and collective movement step. The school size defines the number of fishes used in our algorithm knowing that each fish represents a CNN architecture which can be a possible solution for our optimisation problem. Increasing the number of fishes has an impact on expanding the search space of possible architectures during the search process.

The individual step parameter is predefined before the search process. In our case, it is set to a neutral value (equal to 1). Indeed, each movement will increase or decrease the number of convolution and fully connected layers according to the randomly generated couple of integer values. Multiplying by a neutral value is appropriate in this case.

The collective movement step will control how fast the fishes in the school will be attracted to or spread away from the school barycentre. In our case, a smaller step will make the fish's movement very slow and ineffective in the case where the fish have almost an architecture similar to the barycentre. Moreover, a neutral collective step value (equal to 1) will reduce the diversity of the school and, consequently, all fishes will have the same architecture after each collective movement which will decrease the diversity of the school and limit the search space.

The parameters of the third category are used in the control of the training process of each fish. It contains two parameters:

- The percentage of samples used from the full data set in question to evaluate fishes during the algorithm.
- The number of epochs
- The dropout rate.

The computational resources required for training only one epoch on the used large-scale data sets would require significant computational resources and take a long time. For this reason, we only train and evaluate fishes on 50% of the data set during the optimisation process with respect to the number of samples in each class. 70% of the data are used for training, 20% for testing and 10% for validation. The same distribution is used when the full data set is used.

The number of epochs for fish evaluation will define the number of times that the fish's architecture will work through the training data set before evaluating its training accuracy. When the best is found at the end of the optimisation, it will be evaluated in the test set after a full training for a number of epochs.

The dropout rate parameter is used to randomly omit connections in fully connected layers during the training process of fish's architecture to avoid over-fitting. For all data sets, we trained and evaluated the best architecture found with and without a dropout rate.

Finally, in order to obtain significant results in our experiments, a total of 50 independent tests was performed. We used in our tests a virtual machine having the following characteristics: a single Nvidia Tesla K80 GPU and 13 GB of RAM memory. Although these are considered powerful

features for CNNs training, the amount of RAM remains insufficient to train complex CNNs.

6.4 Results and analysis

6.4.1 About architecture optimisation

For the AG.N, DBP, Yelp.P, Yelp.F and Yah.A data sets, the best test errors recorded by our SiNoptiC for char-level and word-level CNN, shown in Table 5, are 2.73, 0.60, 1.28, 8.47 and 13.23, respectively. The accuracy-test on these data sets are 92%, 98%, 95%, 65% and 56%, respectively. In Figure 5, a boxplot representation of the test accuracy distribution for character and word level CNN on these data sets.

In character-level CNN, our method recorded the best loss results on all data sets with respect to chosen char-based models used for comparison. For the DBP data set, our char best model found has a loss of 1.20 which is nearly equal to the best value found by VDCNN (Conneau et al., 2016).

In word-level CNN, our method recorded the best results on AG.N and Yelp.F data sets for chosen word-based models used for comparison. Although our results are not the best on the DBP, Yelp.P and Yah.A, they are among the two best results. Whereas we are not using any kind of data augmentation, our results are considered promising. For example, Kim's (2014) word-level model was able to achieve test errors of 5.5, 11.5 and 12.99 on DBP, Yelp.P and Yah.A, respectively using a simple 3 layers CNN. It is important to note that a simple CNN containing only 3 layers such as Kim's CNN outperforms the best test errors on three data sets from a total of five. Word-CNN models generated by our algorithm are 3 layered models with different kernel sizes and feature maps (see Table 4). In our word CNN, we apply a max-overtime pooling operation (Collobert et al., 2011) over the feature map. For this reason, the size of the max pooling operation depends on the number of words in the sentence fed to the network.

For all data sets, the terms (+)*dropout* and (−)*dropout* in Table 6 are provided to indicate whether the result generated by the proposed method is with or without the use of a dropout layer between fully connected layers. The best-obtained test errors were found when using dropout. The mean test errors for AG.N, DBP, Yelp.P, Yelp.F and Yah.A data sets in character level CNN are 4.46, 2.26, 2.14, 13.65 and 17.22 respectively, when using dropout, and 7.45, 2.46, 2.73, 14.62 and 19.88, respectively without using dropout.

The mean test errors for the same data sets in word-level CNN are 3.76, 0.88, 2.98, 9.22 and 15.14, respectively, when using dropout, and 4.12, 1.17, 3.12, 10.03 and 17.26, respectively without using dropout.

Table 4 Best found word level models by our algorithm on the benchmark data sets (the size of max-pooling kernel depends on the number of words in the sentence fed to the network)

Data set	*Layer*	*Parameters*
AG.N	CV	kernels:[4, 5, 6]; output filters: 512
	MP	kernel size: variable
	FC	output neurons: 190
DBP	CV	kernels: [2, 3, 4, 5]; output filters: 64
	MP	kernel size: variable
	FC	output neurons: 152
Yelp.P	CV	kernels: [3, 4, 5]; output filters: 512
	MP	kernel size: variable
	FC	output neurons: 126
Yelp.F	CV	kernels: [5, 6, 7]; output filters: 512
	MP	kernel size: variable
	FC	output neurons: 192
Yah.A	CV	kernels: [3, 3, 3]; output filters: 512
	MP	kernel size: variable
	FC	output neurons: 169

Table 5 Best results obtained by SiNoptiC and their computational complexity for five data sets

	Data set	*Error*	*Accuracy (%)*	*# Parameters*
Char CNN	AG.N	3.86	87	523k
	DBP	1.20	97	3M
	Yelp.P	1.92	93	591k
	Yelp.F	9.79	20	983k
	Yah.A	14.08	55	1.6M
Word CNN	AG.N	2.73	92	5M
	DBP	0.60	98	4M
	Yelp.P	1.28	95	2M
	Yelp.F	8.47	65	4.4M
	Yah.A	13.23	56	3M

Table 6 Test errors of SiNoptiC on five data sets with and without dropout

	SiNoptiC	*AG.N*	*DBP*	*Yelp.P*	*Yelp.F*	*Yah.A.*
Char CNN	(+) dropout (Best)	3.86	1.20	1.92	9.79	14.08
	(+) dropout (Mean.)	4.46	2.26	2.14	13.65	17.22
	(–) dropout (Best)	5.16	1.88	2.05	14.18	15.88
	(–) dropout (Mean)	7.45	2.46	2.73	14.62	19.88
Word CNN	(+) dropout (Best)	2.73	0.60	1.28	8.47	13.23
	(+) dropout (Mean)	3.76	0.88	2.98	9.22	15.14
	(–) dropout (Best)	3.08	0.98	1.60	9.47	15.99
	(–) dropout (Mean)	4.12	1.17	3.12	10.03	17.26

Table 7 SiNoptiC test errors comparison with different models

	Model	*AG.N*	*DBP*	*Yelp.P*	*Yelp.F*	*Yah.A.*
Char CNN	Char-level CNN (Zhang et al., 2015)	12.82	1.73	5.89	39.62	29.55
	VDCNN (Conneau et al., 2016)	8.73	1.29	4.28	35.28	26.57
	SiNoptiC	**3.86**	**1.20**	**1.92**	**9.79**	**14.08**
Word CNN	Kim's model (Kim, 2014)	2.78	0.55	1.15	8.60	12.99
	Word-level CNN (Zhang et al., 2015)	8.55	1.42	4.60	40.16	31.50
	Word-CNN (Johnson and Zhang, 2016)	6.95	1.12	3.44	34.21	26.06
	DPCNN (Johnson and Zhang, 2017)	6.87	0.88	2.64	30.58	23.90
	SiNoptiC	**2.73**	0.60	1.28	**8.47**	13.23

With regard to illustrating the convergence behaviour of Synoptic along with the iterations, we run SiNoptiC 5 times on AG.N data set where 20 fishes were used over 15 iterations. We recorded the best fish accuracy at the end of each run, as we can see in Figure 4, the best accuracy was achieved from the 8th iteration and there is no remarkable improvement until the last iteration.

It is important to note that such convergence has nothing to do with the quality of the solutions at the end of the run. It only talks about a set of architectures that will evolve to end up with the same architecture rather than giving different architectures with different depths and parameters.

In Figure 4, it is clear that we could have more chances of reaching a good accuracy at the end of each run. A possible improvement will take place if we could increase the number of iterations and the number of fishes to explore more CNN architectures and find better models for the problem in question. Unfortunately, Owing our limited computational power, it was not possible to increase the number of fishes and iterations in our runs. Reducing the running time of SiNoptiC is a major contribution to the field.

In the proposed algorithm, there are two separate versions: the first considers the entry as a sequence of characters, and the second considers the entry as a sequence of words.

In Figure 5, we illustrated both character and word level CNNs found by our algorithm, the figure shows that the best accuracy was achieved using the word CNN for all benchmark data sets. One of our future research directions is to merge the two versions and create an algorithm where input granularity (character or word) and depth are optimised at the same time.

Figure 4 The accuracy of the best architecture accuracy found over 15 iterations of our SiNoptiC algorithm on AG.N data set

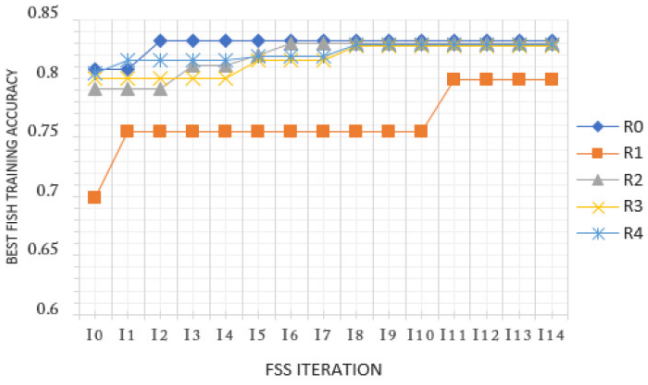
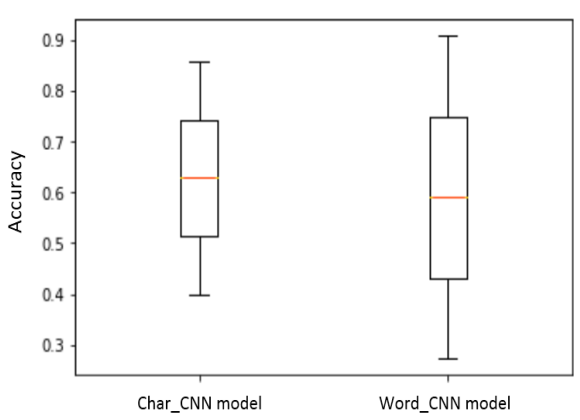
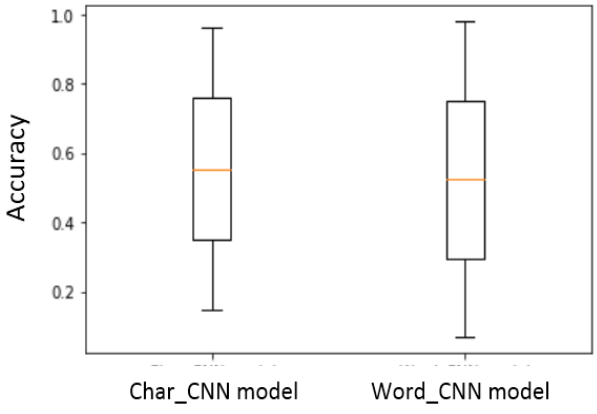


Figure 5 Boxplots of the test accuracy for: (a) AG'News, (b) DBPedia, (c) Yelp Polarity, (d) Yelp Full and (e) Yahoo Answers data sets obtained with the proposed SiNoptiC

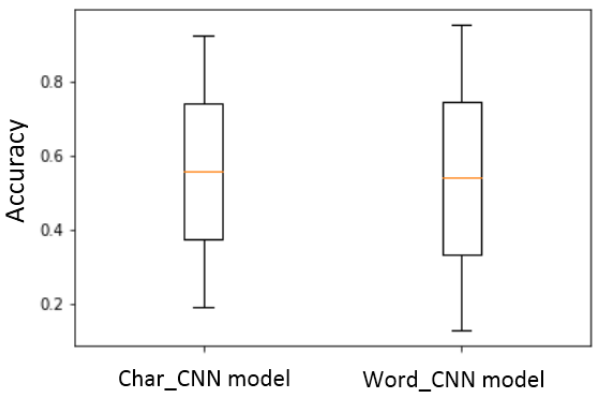


(a)

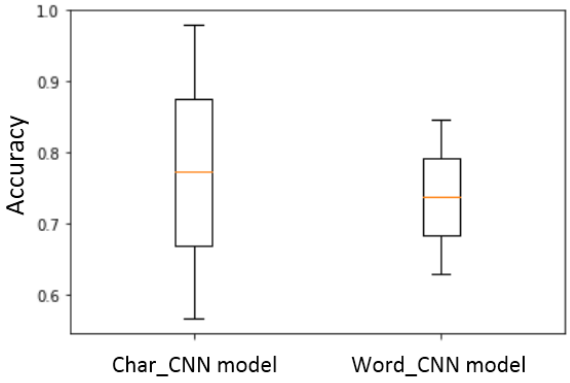


(b)

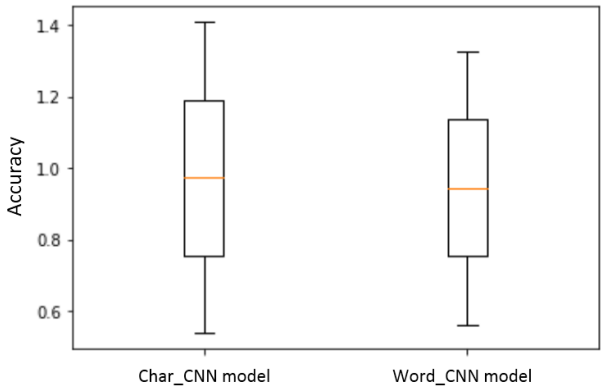
Figure 5 Boxplots of the test accuracy for: (a) AG'News, (b) DBPedia, (c) Yelp Polarity, (d) Yelp Full and (e) Yahoo Answers data sets obtained with the proposed SiNoptiC (continued)



(c)



(d)



(e)

We implemented two versions of our SiNoptiC: the first one initialises the school with architectures using a word embedding as the first layer while the second version uses a character embedding instead. The word-level SiNoptiC provides as best architecture found a model with three layers (Convolution – Max Pooling – Fully Connected) for all data sets.

The character level SiNoptiC performs models with different depths that we presented in Table 8. For each data set, we presented the layers found with their related parameters. All the best CNNs found contain only a single FC layer at the tail of each model. This highlights the performance of the SiNoptiC algorithm because many recent studies have proved that CNNs having a single FC layer at the end are more efficient (Springenberg et al., 2014).

Our results show that the SiNoptiC algorithm can find optimal architectures without any expertise on the domain in question. Even though SiNoptiC was tested for character level and word level separately, our results show that in both cases SiNoptiC can be considered as an efficient way to find optimised CNN architectures. Thus, the proposed algorithm can be a good way to help non-experts automatically design deep architectures and it can be extended to several areas other than text classification.

6.4.2 About individual and collective movements

Individual and collective movements occur for each fish architecture in the school at every iteration of SiNoptiC algorithm according to equations (8), (6) and (7). The step is randomly chosen and each fish depth will increase or decrease without any maximum boundaries.

Figure 6 shows the accuracy of each fish before and after the individual movement during the first iteration of SiNoptiC for the AG.N data set. We have chosen to illustrate the effect of the individual movement during the first iteration because the fish were initialised with different architectures randomly generated and the algorithm has not yet converged to the same architecture.

Table 8 Best found character level models by our algorithm on the benchmark data sets

Data set	*Layer*	*Parameters*
AG.N	CV	window size: 5; output filters: 256
	MP	kernel size: 5
	CV	window size: 3; output filters: 128
	MP	kernel size: 3
	CV	window size: 3; output filters: 256
	MP	kernel size: 5
	FC	output neurons: 300
DBP	CV	window size: 3; output filters: 512
	CV	window size: 5; output filters: 256
	MP	kernel size: 3
	CV	window size: 3; output filters: 64
	MP	kernel size: 3
	CV	window size: 3; output filters: 128
	MP	kernel size: 3
	CV	window size: 3; output filters: 64
	MP	kernel size: 3
	FC	output neurons: 1024
Yelp.P	CV	window size: 3; output filters: 256
	MP	kernel size: 3
	CV	window size: 3; output filters: 512
	CV	window size: 3; output filters: 256
	MP	kernel size: 5
	CV	window size: 3; output filters: 128
	MP	kernel size: 5
	CV	window size: 3; output filters: 64
	MP	kernel size: 3
	FC	output neurons: 378
Yelp.F	CV	window size: 5; output filters: 512
	MP	kernel size: 5
	CV	window size: 3; output filters: 256
	MP	kernel size: 5
	CV	window size: 3; output filters: 64
	MP	kernel size: 3
	CV	window size: 3; output filters: 128
	CV	window size: 3; output filters: 64
	MP	kernel size: 3
	FC	output neurons: 257
Yah.A	CV	window size: 5; output filters: 128
	MP	kernel size: 3
	CV	window size: 3; output filters: 256
	MP	kernel size: 3
	CV	window size: 3; output filters: 256
	MP	kernel size: 3
	CV	window size: 3; output filters: 128
	MP	kernel size: 3
	CV	window size: 5; output filters: 128
	MP	kernel size: 3
	FC	output neurons: 309

Figure 6 Individual movements illustrated during the first iteration of SiNoptiC on AG.N data set

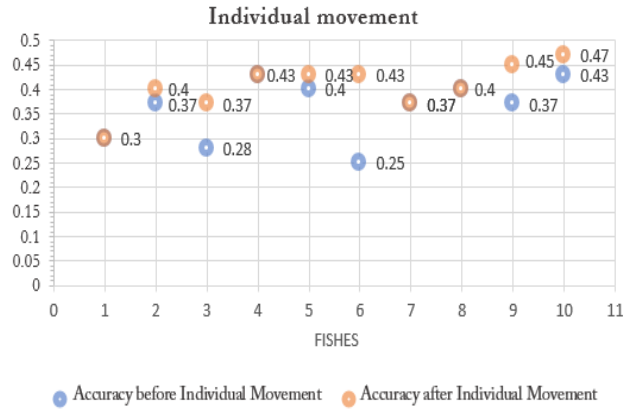
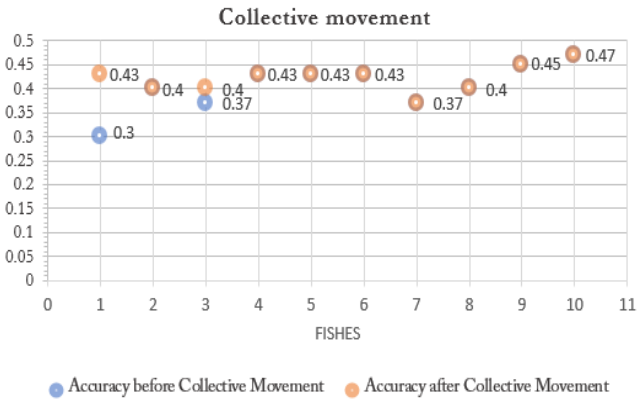


Figure 7 Collective movements illustrated during the first iteration of SiNoptiC on AG.N data set



Hence, during this iteration, the effect of the individual movement is more clear. Over ten fishes, six fishes had successful individual movements. For example, fish 2 moved from the accuracy 37 to 40% after a change in its architecture. Unlike fish 1 which remained at the same position with the same accuracy of 30% and without any change in its architecture. We should mention that according to our algorithm, the individual movement of the fish does not occur if there is no improvement in the accuracy. Figure 6 shows the accuracy of each fish before and after the collective movement that accrued directly after the previous individual movement to track the fish's accuracy. We can see that the effect of the collective movement has less impact on the improvement of the fish's accuracy when compared to the individual movement. This is explained by the fact that during the collective movement fishes having unsuccessful movements will have more chances to change their architectures.

Indeed, the fishes having successful individual movements will influence the resulting direction of movement more than the unsuccessful ones. Only two fish (Fish 1 and Fish 3) over ten recorded an improvement after their collective movement. This shows that our defined collective operator is working according to the designed objective which is enhancing the overall performance of the school.

In the SiNoptiC algorithm, we can see the importance of both individual and collective movement on the best architecture found. Figure 6 shows that the accuracy of the fish is getting better after each individual movement which could be an increase or decrease in the number of layers or a change in the layers' parameters. We found that sometimes a random change can achieve good accuracy. Figure 7 shows that the proposed collective movement operator is performing an update to the fish architecture in a way that either improves its accuracy or makes no change. In all cases, the design of this operator in our algorithm is indeed working similarly to an FSS algorithm by making only good fishes influence the resulting direction of this movement. All the designed operators exert a strong influence on the search result.

6.4.3 About computational complexity

Although it is not easy to compute the algorithm's computational complexity theoretically because of the stochastic nature of FSS and CNNs, the obtained results show that SiNoptiC can find very competitive architectures with fewer parameters. For example, if we compare our char-CNN best model with VDCNN (Conneau et al., 2016) on Yelp.P data set, we can see that our best model has 10 layers, an error of 1.92 and only 591 thousand parameters, is much simpler than VDCNN.

SiNoptiC has shown its ability to find simpler models with fewer parameters than those of other comparable models. For example, in DBP, Yelp.P and Yah. Data set, best models found by SiNoptiC algorithm only have 3, 0.591 and 1.6 million parameters respectively (see Table 5) which are much less than those of VDCNN (Conneau et al., 2016).

It should be noted that there are studies that have shown that, at the word level, simple three-layer models are much more efficient than deep models. At the character level, the models provided by our algorithm are simple when compared with VDCNN (Conneau et al., 2016). So, SiNoptiC can find models with less number of layers than the peer competitor models; this can be explained by the choice of initialising the fish with 'shallow' networks (having a small number of layers). This emphasises the fact that the depth in many handcrafted models is not optimal, and it is possible to find models that are shallower and have higher precision.

Also, SiNoptiC has proven its efficiency in giving competitive results even without the use of complex architectures and data augmentation techniques. From the results presented in Section 6, the models found by the proposed word-level SiNoptiC are three-layered models and similar in their architecture to Kim's model (2014).

In each iteration, SiNoptiC evaluates the fitness function of each fish after the individual and the collective movement in order to update the best fish. Each evaluation involves training the fish's model. Fortunately, the number of epochs required for the training convergence is relatively small. This has reduced the computation time of our algorithm.

7 Conclusions

In this paper, a new method of automatic CNNs architecture design for text classification problems was proposed. The proposed algorithm helps to optimise CNNs in a proficient way by using a coupling CNN with FSS-based optimisation. We adopted an encoding strategy allowing for variable-length CNN architectures. We re-defined the operators of the standard FSS algorithm such as the individual and collective movement for efficiently searching for adequate architecture.

The experimental result has shown that the proposed method can automatically find competitive CNN architectures compared with the state-of-the-art models. The experimental results show that if we had more computing power, the proposed method could provide even better architectures. We will publish our code so that the research community can easily build on top of our work.

Future work involves investigating alternative adaptation of the meta-heuristic FSS to solve CNN architecture optimisation with many objectives such as the granularity of the CNN input, computational complexity and classification accuracy. Therefore, a decision-maker could configure his model according to his needs, which will further improve the classification accuracy of the resulting models.

So far the proposed algorithm is limited to CNNs but can be easily adapted to work for any feedforward architecture including recurrent or modular architectures. Another challenge for future research is the extension to general networks.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G. and Isard, M. et al. (2016) 'Tensorflow: a system for large-scale machine learning', *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, pp.265–283.

Ammar, M., Hidri, A. and Sassi Hidri, M. (2020) 'Time-sensitive clustering evolving textual data streams', *International Journal of Computer Applications in Technology*, Vol. 63, Nos. 1/2, pp.25–40, 2020.

Baldominos, A., Saez, Y. and Isasi, P. (2019) 'On the automated, evolutionary design of neural networks: past, present, and future', *Neural Computing and Applications*, pp.1–27.

Bastos Filho, C.J.A., de Lima Neto, F.B., Lins, A.J.C.C., Nascimento, A.I.S. and Lima, M.P. (2008) 'A novel search algorithm based on fish school behavior', *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, IEEE, pp.2646–2651.

Bengio, Y., Courville, A. and Vincent, P. (2013) 'Representation learning: a review and new perspectives', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 8, pp.1798–1828.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B. and Shelhamer, E. (2014) 'Cudnn: efficient primitives for deep learning', *arXiv preprint arXiv:1410.0759*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011) 'Natural language processing (almost) from scratch', *Journal of Machine Learning Research*, Vol. 12, pp.2493–2537.

Conneau, A., Schwenk, H., Barrault, L. and Lecun, Y. (2016) 'Very deep convolutional networks for natural language processing', *arXiv preprint arXiv:1606.01781*, Vol. 2, pp.1–10.

Corbat, L., Nauval, M., Henriet, J. and Lapayre, J.C. (2020) 'A fusion method based on deep learning and case-based reasoning which improves the resulting medical image segmentations', *Expert Systems with Applications*, Vol. 147, No. 8. Doi: 10.1016/j.eswa.2020.113200.

Csáji, B.C. (2001) *Approximation with Artificial Neural Networks*, Faculty of Sciences, Etvs Lornd University, Hungary, Vol. 24, p.48.

Dean, J. and Hölzle, U. (2017) *Build and Train Machine Learning Models on Our New Google Cloud Tpus*, Google Cloud.

Delalleau, O. and Bengio, Y. (2011) 'Shallow vs. deep sum-product networks', *Advances in Neural Information Processing Systems*, pp.666–674.

Ferjani, E., Hidri, A., Sassi Hidri, M. and Frihida, A. (2019) 'Mapreduce-based convolutional neural network for text categorization', *Proceedings of the 11th International Conference on Computational Collective Intelligence (ICCCI)*, pp.155–166.

Fogel, D.B. (1995) 'Phenotypes, genotypes, and operators in evolutionary computation', *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp.193–198.

Glorot, X. and Bengio, Y. (2010) 'Understanding the difficulty of training deep feedforward neural networks', *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pp.249–256.

Goldberg, Y. and Levy, O. (2014) 'word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method', *arXiv preprint arXiv:1402.3722*.

Hidri, A. (2017) 'Optimization for training CNN deep models based on swarm intelligence', *International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, pp.284–289.

Ioffe, S. and Szegedy, C. (2015) 'Batch normalization: accelerating deep network training by reducing internal covariate shift', *arXiv preprint arXiv:1502.03167*.

Jiang, J., Han, F., Ling, Q., Wang, J., Li, T. and Han, H. (2020) 'Efficient network architecture search via multiobjective particle swarm optimization based on decomposition', *Neural Networks*, Vol. 123, pp.305–316.

Jiang, M., Liang, Y., Feng, X., Fan, X., Pei, Z., Xue, Y. and Guan, R. (2018) 'Text classification based on deep belief network and softmax regression', *Neural Computing and Applications*, Vol. 29, No. 1, pp.61–70.

Jin, H., Song, Q. and Hu, X. (2019) 'Auto-keras: an efficient neural architecture search system', *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.1946–1956.

Johnson, R. and Zhang, T. (2016) 'Convolutional neural networks for text categorization: shallow word-level vs. deep character-level', *arXiv preprint arXiv:1609.00718*.

Johnson, R. and Zhang, T. (2017) 'Deep pyramid convolutional neural networks for text categorization', *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp.562–570.

Junior, F.E.F. and Yen, G.G. (2019) 'Particle swarm optimization of deep neural networks architectures for image classification', *Swarm and Evolutionary Computation*, Vol. 49, pp.62–74.

Kalchbrenner, N., Grefenstette, E. and Blunsom, P. (2014) 'A convolutional neural network for modelling sentences', *arXiv preprint arXiv:1404.2188*.

Kennedy, J. (2006) 'Swarm intelligence', *Handbook of Nature-Inspired and Innovative Computing*, pp.187–219.

Khalifa, M.H., Ammar, M., Ouarda, W. and Alimi, A.M. (2017) 'Particle swarm optimization for deep learning of convolution neural network', *Sudan Conference on Computer Science and Information Technology (SCCSIT)*, pp.1–5.

Kim, Y. (2014) 'Convolutional neural networks for sentence classification', *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, October 25–29, Doha, Qatar, pp.1746–1751.

Kingma, D.P. and Ba, J. (2014) 'Adam: a method for stochastic optimization', *arXiv preprint arXiv:1412.6980*.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, Vol. 86, pp.2278–2324.

Lorenzo, P.R., Nalepa, J., Kawulok, M., Ramos, L.S. and Pastor, J.R. (2017) 'Particle swarm optimization for hyper-parameter selection in deep neural networks', *Proceedings of the Genetic and Evolutionary Computation Conference*, pp.481–488, 2017.

Moitra, D. and Mandal, R.K. (2020) 'Classification of non-small cell lung cancer using one-dimensional convolutional neural network', *Expert Systems with Applications*, Vol. 159. Doi: 10.1016/j.eswa.2020.113564.

Montufar, G.F., Pascanu, R., Cho, K. and Bengio, Y. (2014) 'On the number of linear regions of deep neural networks', *Advances in Neural Information Processing Systems*, pp.2924–2932.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. (2017) 'Automatic differentiation in pytorch', *Proceedings of the 31st Conference on Neural Information Processing Systems*, Long Beach, CA, USA, pp.1–4.

Pennington, J., Socher, R. and Manning, C.D. (2014) 'Glove: global vectors for word representation', *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp.1532–1543.

Serizawa, T. and Fujita, H. (2020) 'Optimization of convolutional neural network using the linearly decreasing weight particle swarm optimization', *arXiv preprint arXiv:2001.05670*.

Sivakumar, S. and Rajalakshmi, R. (2021) 'Self-attention based sentiment analysis with effective embedding techniques', *International Journal of Computer Applications in Technology*, Vol. 65, No. 1, pp.65–77.

Springenberg, J.T., Dosovitskiy, A., Brox, T. and Riedmiller, M. (2014) 'Striving for simplicity: the all convolutional net', *arXiv preprint arXiv:1412.6806*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) 'Going deeper with convolutions', *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.1–9.

Wang, B., Sun, Y., Xue, B. and Zhang, M. (2018) 'Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification', *IEEE Congress on Evolutionary Computation (CEC)*, pp.1–8.

Wang, H. and Raj, B. (2017) 'On the origin of deep learning', *arXiv preprint arXiv:1702.07800*, pp.1–71.

Wang, Y., Zhang, H. and Zhang, G. (2019) 'cpso-cnn: an efficient pso-based algorithm for fine-tuning hyper-parameters of convolutional neural networks', *Swarm and Evolutionary Computation*, Vol. 49, pp.114–123.

Yann, L., Yoshua, B. and Geoffrey, H. (2015) 'Deep learning', *Nature*, Vol. 521, pp.436–444.

Zhang, T., Zhang, Y., Cao, Y., Li, L. and Hao, L. (2020) 'Diagnosing parkinson's disease with speech signal based on convolutional neural network', *International Journal of Computer Applications in Technology*, Vol. 63, No. 4, pp.348–353.

Zhang, X., Zhao, J. and LeCun, Y. (2015) 'Character-level convolutional networks for text classification', *Advances in Neural Information Processing Systems*, pp.649–657.

Zhu, L., Wang, J. and Li, K. (2020) 'Computer image analysis for various shading factors segmentation in forest canopy using convolutional neural networks', *International Journal of Computer Applications in Technology*, Vol. 64, No. 4, pp.415–428.