Journal of Engineering Science and Technology
Vol. 17, No. 2 (2022) 1532 - 1549
© School of Engineering, Taylor's University

OPTIMAL HYPER-PARAMETER TUNING USING CUSTOM GENETIC ALGORITHM ON DEEP LEARNING TO DETECT TWITTER BOTS

KARTHIKAYINI THAVASIMANI[1,*], N. K. SRINATH[2]

[1]Department of Computer Science and Engineering, RV College of Engineering, Mysore Rd, RV Vidyaniketan Post, Bengaluru, Karnataka 560059, India
[2]Department of Computer Science and Engineering, RV College of Engineering, Mysore Rd, RV Vidyaniketan Post, Bengaluru, Karnataka 560059, India
*Corresponding Author: karthikayini@outlook.com

Abstract

Deep learning, an evolution of the machine learning methods, is nowadays successfully used in various applications such as object detection, image recognition, language processing, bot detections, fraud detections, etc. Although its successful exploration in many areas, researchers find difficulties in choosing the network architectures suitable for the problems. It takes a substantial amount of time to create a babysitting model by manually tuning the hyperparameters for any deep learning architecture. Currently, research is going on in optimizing the hyperparameters of deep learning models using various search techniques. In our paper, we propose a Custom Genetic algorithm (CGA), an enhanced optimization technique to tune the hyperparameters of a deep learning model for Twitter bot detection. The proposed algorithm overcomes the limitations of the native genetic algorithm like early convergence and local optima traps. We compared our experimental results against default hyperparameter values and existing techniques. The results were promising and, our proposed CGA technique outperformed the current research techniques.

Keywords: Bot account detection, Custom genetic algorithm (CGA), Genetic algorithm, Hyperparameter optimization, Hyperparameter tuning, Optimization techniques.

1. Introduction

A bot is a program or script created to automate the tasks. There are two types of bots on the internet, legitimate and malicious bots. Although the bots are created for legitimate tasks [1] such as automatic execution of a script, web crawling, chatbots, etc., other bots' categories are malicious.[2] that is mainly intended for credential theft, dissemination of information, execution of network attacks, fake social network accounts, false allegations, counterfeit advertisements, malicious URLs, etc. Detecting such malicious bots is very crucial as it possesses serious security threats to online businesses. Statistics say that 94.2% of the entire websites in the world are affected by bots. Because they are more sophisticated and intelligent enough to mimic humans [3], it has created immense interest among the researchers to develop new bot detection techniques using ML and DL algorithms.

As mentioned by Ferrara et al. [3], bot detection is based on three approaches, static, behavioral, and hybrid. An online social network, in particular, Twitter, is highly targeted by the bots. They redirect the users to intermediate malicious URLs resulting in a security breach such as phishing attacks. Twitter recorded more than 9.9 million suspicious accounts per the Washington Post statistics in 2018, and the count was tripled in late 2017 [4]. There are about 48 million bot accounts on Twitter now, amounting to roughly 15% of all existing accounts [5].

Karthikayini and Srinath [6] applied different optimizers to detect the bots from CRESCI 2017 dataset using deep learning. Researchers are developing new approaches to fight the endless influenza of malicious bots targeting social networks. Most of the research is actively headed toward deep learning nowadays. It is successfully working on image classification, speech recognition, language processing, spam detections, etc.

The deep learning models are developed using several architectures such as Convolution neural network (CNN), Feedforward network, recurrent network, etc. These Deep learning models have different hyper-parameters such as Number of hidden units, Number of hidden layers, activation functions, momentum, learning rate, learning rate decay, mini-batch, Epsilon (€), β1 and β2 mainly used to optimize the performance. However, most researchers use default parameter values of the deep learning models that drastically impact the performance. The hyperparameters influence the results to a greater extent. So, using optimal hyper-parameter values are very crucial.

Our work is organized in the following sequence. Section 1 presents the related work on the existing bot detection techniques and hyper-parameter tuning techniques. Section 2 is about dataset collection and feature extraction steps. Section 3 explains the proposed Custom Genetic Algorithm (CGA) to auto-tune the hyper-parameters along with the proposed framework. Section 4 depicts the experimental results and discussion Section 5 Conclusion and future work.

2. Related Work

Cai et al. [7] proposed a behavior enhanced deep model to detect bots by combining content and behavior information on the Twitter Honeypot dataset. The BeDM model process the user contents through the convolutional layer and behavioral data such as timestamps and posting types using the LSTM network. Softmax is applied for the final output layer to classify bot accounts from human accounts. The author reported precision-88.41%, recall-86.26%, F1score- 87.32%, and proved that the proposed model outperforms existing techniques such as Boosting, BoostOR, and Stweeler. The author experimented with very few sets of parameters for CNN & LSTM like size, filter width, memory dimensions, Number of hidden units.

Antoun et al. [8] addressed three challenges of fake news detection. One of those is bot detection. The author proposed a Voting classifier for ensemble tree methods by fusing Random Forest, Adaboost, XGboost, and processing various user content features to classify bots and human accounts. He reported the accuracy range from 96% - 99%. They also said that the Random Forest shows 96% accuracy for a specific range of features.

Wei and Nguyen [9] classified the bot accounts and human accounts using BiLSTM(Bi-directional long, short term memory), a deep learning model on the CRESCI-2017 dataset. Authors concluded that they got similar performances compared with the existing work with 96 % accuracy, 96.3 % F-measure, 92 % MCC for Test 1 and 92.9% Accuracy, 92.6% F-measure, 0.857 % for Test 2.

Fernquist et al. [10] made a study on political bots from Twitter during the Swedish general election, September 2018. Authors applied a machine learning model on a Twitter dataset with a vast collection of the bot, and genuine accounts crawled from various sources. Different machine learning algorithms such as random forest, AdaBoost, support vector machines, logistic regression, and naive Bayes were tested. They concluded random forest was showing the highest accuracy compared to the other ML techniques. They compared their results with other existing models showing improvement in recall (97.6%) alone.

Khalil et al. [11] used two clustering methods db-scan and K-means, for bot detection. They concluded that db-scan shows better performance than K-means with 97.7% accuracy, 94%, f-measure, 91% precision, and 98% recall. Siddiqui et al. [12] used a javascript testing framework with REST API for bot detection. They experimented with 700 Twitter accounts and reported that 11% of the dataset are bot accounts.

Liu [13] developed a lag-sensitive hash technique named "De-bot," a method by clustering the user accounts into correlated sets in real-time and reported they could detect thousands of bots per day with a precision of 94%. They also mentioned that 544868 unique bots were captured in the year 2016. Kiran et al. [14] developed an artificial neural network model using the "neural net' library provided by R and experimented on the Twitter dataset crawled using Python's tweepy library. They came up with an accuracy ranging from 91.20% to 97.79%. John P.

Dickerson et al. [15] proposed a framework called SentiBot by analysing both user content features and behavior features for bot detection. The results showed an improvement in the performance, especially accuracy, which increased from 0.65 to 0.73. They used 7.7 M Indian political tweets of over 555,000 users.

Lingam et al. [16] introducing a social bot detection algorithm with a Trust model to identify a trustworthy path. The trust model comprises two parameters: The first parameter uses the Bayesian theory to obtain the trust value from direct participants. The second parameter uses the Dempster-Shafer theory to get the trust value from neighboring participants. They achieved an accuracy of 95% and a 4% increase compared to the existing algorithm.

Heredia et al. [17] investigated the US election cycle to examine how far the social bots influence the presidential candidates' sentiment. For this purpose, they created a CNN model to be trained on the Sentiment140 dataset first and then trained CNN used to label the election dataset captured in the year 2016. They finally concluded that the model was able to show an accuracy of 84%. Ranjit et al. [18] experimented on distributed hyperparameter tuning for long term convolutional recurrent neural network in cloud infrastructure using Bayesian optimization. They used the HyperDrive framework of Azure ML Service for the Video Activity Recognition problem. They claimed that Bayesian optimization in the cloud infrastructure resulted in better hyperparameter space coverage with improved validation accuracy.

Neary [19] proposed a reinforcement learning technique to optimize the convolutional neural network's hyper-parameters for object recognition. They used a multi-agent-based method where each agent is responsible for finding optimal parameters based on reinforcement learning. Using the MNSIT dataset, they reported that the final configuration of the algorithm's accuracy is 95% and, they were able to obtain the optimal neural network structure in a short period. To reduce the time and effort spent on the manual tuning of hyperparameters, Cho et al. [20] proposed a DEEPBO algorithm, an enhanced Bayesian optimization technique with diversification, early termination, parallelization, and cost function strategies for CNN. Compared with GP-Hedge and BOHB, they proved that their algorithm is robust and outperformed all the existing approaches with optimal parameters.

Goel et al. [21] proposed architecture for screening patients into three categories, COVID-19 positive, pneumonia and normal automatically. Through manual tuning of hyperparameters, they achieved an accuracy of 97.78% outperforming existing CNN models. Choi et al. [22] used MNIST and CIFAR-10 datasets by experimenting with learning curve predictions against different hyperparameter configurations. They analysed 20,000 learning curves' characteristics corresponding to the 20,000 different hyperparameter configs with two Early termination rules. They concluded that HPO (Hyperparameter Optimization) on CNN using the learning curve predicted that the prediction is tough and challenging as the curve drastically varies if there is a minor change in hyperparameters.

Aich et al. [23] proposed a CNN-based model to extract information from web content categories useful for text mining applications. They achieved accuracy in the range 85 and 92% for the text classification through manual tuning of hyperparameters. Ugli et al. [24] developed a deep learning system for detecting and classifying the distractions in real-time while driving a vehicle to avoid road accidents. A new method is developed with pre-trained weights and various optimizers using ResNet50 to improve the learning capabilities. On the "distracted drivers" test dataset, they achieved an accuracy of 98.4%.

Chakraborty et al. [25] achieved an accuracy of 95.29 from a 3D CNN model in detecting Parkinson's disease. Sequential model-based Bayesian Optimization is used with a fixed penalty value to tune the model's hyperparameters. Antonio [26] proposed a variation of Bayesian Optimization using Subject Machine Vector that had better computational efficiency over Bayesian Optimization with fixed penalty value. Kong et al. [27] proposed a tree-structured Parzen estimator integrated with short- term load forecasting (STLF) framework to obtain better forecasting performance for LSTM mode. Yi et al. [28] proposed the hypernet framework by fusing the Bayesian technique and meta-learning for better highway traffic prediction. The whole experiment was conducted using the Korean highway system dataset. Bui and Yi [29] used meta-learning to optimize the deep learning model's hyperparameters for better traffic prediction. They took data from the vehicle detection system for their experiment.

After reviewing the above papers on hyperparameter tuning, we understood that training a deep learning model requires lots of effort, especially by training it with a wide range of hyperparameter values manually, which is frustrating and time-consuming. Any deep learning specialists cannot create the best DL model on the first go, requiring lots of iterations and optimizations. So, it's crucial to have the best optimization algorithms to explore a wide range of parameter values. Currently, Bayesian Optimization dominates the hyperparameter optimization problem space. They focus on optimizing the configuration selection to solve the fundamental problem of optimizing a high dimensional function that is non-convex in nature with possible noisy evaluation and unknown smoothness [30].

Considering the current challenges and the black box methods that popular optimization methods use, we wanted to create a straight-forward approach that finds the optimal hyperparameters by relying on an early stopping strategy coupled with evaluating orders of magnitudes of optimal parameters. Our primary motive is to create a better optimization technique to find the best hyperparameters for a model.

3. Dataset

We used the CRESCI-2017 dataset [31, 32] in our proposed work that contains a combination of accounts such as genuine and spambots (social and traditional). Every category has a tweet and user file.

The dataset consists of 7,931 accounts and 3604238 tweets. Cresci et al. [31] used Manual analysis and digital DNA for the classification of accounts. For detecting the spambots, a combination of available user variables, LCS (Longest Common Substring) curve, and various Twitter entities such as hashtag, media, URL, user mentions, retweets, etc., are used.

We used a supervised deep learning model with tuned hyperparameters that require labelled data. So, we used CRESCI 2017 dataset in our work. We ignored CRESCI -2019 dataset due to a lack of tweets data. To evaluate our model, we created two datasets, Test Set 1 and Test Set 2. Test set 1 is about the retweets of an Italian political candidate. It contains 50% of human accounts and 50% of accounts from the group, social bot 1. Test set 2 is about product spammers on Amazon and contains 50% of human accounts and 50% of accounts from the group, social bot 3.

4. Data Pre-processing

The dataset includes sparse data with missing values and a combination of strings and numbers, including special characters. In the first step, we updated the null, missing, and empty values with 0 and NA for numbers and strings, respectively.

The user level attributes are id, screen_name, name, followers_count, statuses_count, friends_count, listed_count, favourites_count, lang, url, time_zone, default_profile, location, geo_enabled, default_profile_image, profile_banner_url, profile_image_url, profile_background_image_url_https, profile_use_background_image, profile_text_color, profile_sidebar_border_color, profile_image_url_https, profile_background_tile, profile_background_image_url, profile_sidebar_fill_color, profile_background_color, is_translator, profile_link_color, utc_offset, follow_request_sent, verified, protected, description, notifications, contributors_enabled, created_at, following and timestamp.

The tweet level attributes are id, user_id, source, text, truncated, in_reply_to_status_id, in_reply_to_screen_name, in_reply_to_user_id, geo, retweeted_status_id, contributors, place, reply_count, retweet_count, favorite_count, retweeted, favorited, num_hashtags, possibly_sensitive, num_mentions, num_urls, timestamp, created_at, crawled_at and updated.

4.1. Step 1

The user file consists of 40 features of categorical data. User-level features such as crawled at, and timestamp are removed since they don't impact the classification process. We added a new label named the target value with the value 0 or 1 depending on the group name mentioned in Table 1. Zero denotes human accounts, and 1 indicates bot accounts. We replaced the missing numerical values with 0 and non-numerical values with NA, where the non-numerical values are converted to numbers using the one-hot encoding technique. The preprocessed input files are merged into a single file called "users_data" with 39 features for further analysis.

Table 1. Descriptive statistics on each dataset.

Group Name	Description	Accounts	Tweets	Year
Genuine Accounts	Verified human accounts	3,474	8,377,522	2011
Social spambots #1	Retweeters of an Italian political candidate	991	16,10,176	2012
Social spambots #2	Spammers of paid apps for mobile devices	3,457	4,28,542	2014
Social spambots #3	Spammers of products on sale at Amazon.com	464	14,18,626	2011
Traditional spambots #1	training set of spammers used by C. Yang, R. Harkreader, and G.Gu	1,000	1,45,094	2009

4.2. Step 2

The tweet file consists of 25 features of categorical data and a total of 3604238 tweets. We used batch processing for analysing using tweets features such as tweets_url, retweet_count, tweets_place reply_count, favorite_count, num_hashtags, num_urls, and num_mentions. Other features are removed due to less impact on classification. Non-numerical values are converted to numbers using the one-hot encoding technique. The numerical features are imputed with the mean value. The preprocessed tweet files are merged into a single file called “tweets_data”.

4.3. Step 3

The final dataset is created by merging the user and tweet related data. For every user, associated tweet related information is extracted using percentage, impute and count functionalities. The final dataset contains a comprehensive list of features named, id, screen_name, name, followers_count, statuses_count, favourites_count, friends_count, listed_count, lang, url, time_zone, default_profile, location, default_profile_image, profile_image_url, geo_enabled, profile_background_image_url_https, profile_use_background_image, profile_banner_url, profile_text_color, profile_sidebar_fill_color, notifications, favorite_count, utc_offset, profile_image_url_https, profile_sidebar_border_color, updated, profile_background_tile, profile_background_image_url, profile_link_color, profile_background_color, follow_request_sent, protected, is_translator, verified, description, contributors_enabled, created_at, following, tweets_url, retweet_count, tweets_place reply_count, num_hashtags, num_urls and num_mentions and bot.

5.Feature Selection

In this step, we used Chi-squared technique to determine the important features for prediction against the target label. As a result of this step, features such as statuses_count, friends_count, followers_count, user_name, favorites_count, listed_count, lang, url, location, time_zone, default_profile, geo_enabled, profile_background_image_url_https, profile_image_url_https, verified, profile_text_color, profile_background_image_url, profile_background_tile, protected, profile_sidebar_border_color, profile_background_tile, profile_sidebar_fill_color, description, profile_background_color, profile_link_color, tweets_url_percentage, retweets_percentage, reply_percentage, hashtags_percentage, favourites_percentage, num_urls_percentage, tweets_place_percentage, num_mentions_percentage and bot are used for prediction in our proposed algorithm.

6. Proposed Work

We propose the Custom Genetic Algorithm (CGA), a hyperparameter optimization technique for tuning the deep learning model. The genetic algorithm (GA) is inspired by natural selection and belongs to the larger class of evolutionary algorithms (EA). Its widely used to generate high-quality solutions for search problems and optimization. Some of the limitations are the tendency to converge towards local optima rather than global optimum and problems with fitness measure, success or failure as there is no way to converge on the solution.

We created a Custom Genetic Algorithm (CGA) with three modules, population selection, retain, and penalty-bonus to address this. Due to the fitness function's success/failure measure, the GA algorithm will not converge on the global optimum. The algorithm reaches the solution very quickly and will not explore the search space. We keep a part of randomly discarded individuals to analyse the search space with a wide range of values. To avoid early convergence, we apply penalties and bonuses to the population. Any population reaching the convergence early will be assigned a penalty, otherwise a bonus. While creating the future population, parents with bonuses are preferred for crossover and mutation. As a result, a wide range of the population is created for every generation. The above concepts are implemented in deep learning models to determine the optimal hyperparameters for a problem.

6.1. Module 1 (M1) – Retaining a part of discarded neural networks

The fittest Neural Networks (NNs) are used for producing offspring, and the remaining networks are discarded in future selection. This results in reaching convergence at an early stage and a low rate of mutation. To increase the mutation rate, diversity, and to stop the premature convergence, a random part of discarded neural networks is considered for further reproduction. To generate the random list of discarded, NNs with M2 values as discussed in Module 2 is preferred. In case of insufficient NNs with M2 values, the population includes randomly picked networks without M2 as well.

6.2. Module 2 (M2) – Penalty and bonus

We used penalties and bonuses to maximize the convergence towards the global optimum of the problem. Population (Neural networks with hyperparameters selected due to crossover and mutation) reaching early convergence are penalized. Otherwise, a bonus is assigned. During crossover and mutation for generating new neural networks, parent networks with bonuses are preferred. The M2 values (bonus/penalty) is routed using mRNA in the hyperparameters. Each Network, over a period of iterations, will contain M2. The future selection criteria focus on and prioritize the M2 output.

6.3. Module 3 (M3) – Population selection

The fittest neural networks, part of the population for initial and progressive steps, are determined using the proposed algorithm. The initial process involves starting with the default population. Selecting a random parameter for the default population can significantly impact the overall time to find the hyperparameters. It may be either faster or slower, or moderate. One of the significant challenges faced by genetic algorithms is the execution time, which depends on the hyperparameter space, available resources, and populations in the traversed generations.

The proposed algorithm contains the below steps,

i. Pick a random set of hyperparameter configurations from the list
ii. Evaluate the set's performance
iii. Select the top 50% sets and ignore the rest
iv. Go back to step 2 until the optimal population is found (default to 10)

The algorithm focuses on reducing the training time of the sets, which doesn't lead us to any concrete solution; instead, it picks up the top-performing settings. The proposed algorithm is different from the existing algorithm in the below ways,

i. Resource overflow is controlled since there is a max of configs to converge
ii. Execution time is faster as top performing 50% sets are considered to progress further, essentially dropping the sets to traverse to a greater extend

Module 3 (M3) - Population Selection

```
Input: Limit L, Set s where $j_{i,c}$ denotes the cth loss from the ith set, Max size M
    1.  Initialization:
        $S_0 = [n]$, N=10
    2.  Finding optimal set:
        For i $\leftarrow$ 0,1, …. c
                Set $s_i = [sN^{-i}]$, $m_i = [MN^{i-c}]$
                Pull each set in $S_i$ for $m_i$ times
        Keep the best set in terms of $m_i$th observed loss as $S_{i+1}$
Output: Optimal set, $O_s$
```

Custom Genetic Algorithm

Inputs: epochs $e_{i…n}$, batch size $b_{i…n}$, layers $l_{i…n}$, neurons $n_{i…n}$, optimizers O_i, activation_function a_i, last_layer_activation_function ll_i, losses ls, $H_{i…n}$, where H_i is the hyperparameters space.

```
1. Initialization
   $H_o \leftarrow e_o, b_o, l_o, n_o, O_o, a_o, ll_o, ls_o$, penalty=null; bonus=null; early stop with
   patience, $es_p$=5, N=10; (Refer to Table 2 for range)

2. Create the initial population
   For $i \leftarrow$ 0 to initial_pop_size,
       Create N
   End For
   Invoke M3 to return the optimal $N_o$ to start
   Return $N_o$

3. For $i \leftarrow$0 to number of generations
     Train the networks, $N_i$
     Evolve P
         $a_{cc}$ = Score and Sort $N_i$ in descending order
         crossover, $N_c$ = Select the parents,
            p with bonus and high $a_{cc}$
         while len(children c) < desired length:
               if $N_c == N_i$
               $N_c$ = random (p)
               #Child Network
               $N_c$ = Breed and mutate
     #M2
     If early stop, es:
               #Add penalty, $p_{es}$
               $N_c = N_c , p_e$
               else:
               #Add bonus, $b_{es}$
               $N_c = N_c , b_{es}$
     N $\leftarrow$ N { $p_e , b_{es}$ } (M2 values via mRNA)

     #M1
     For $i \leftarrow$ 1 to retain_length
               #Keep some for diversity
               If random_select > random.random():
                      population.append(individual)

     Invoke M3 to generate optimal P for future evolutions
     Iterate until H (Stop if H is no longer increasing)

4. #Optimal Hyperparameter space
     Return H with highest $a_{cc}$
```

For validation, we used the CRESCI dataset to identify human and bot accounts on twitter.

7. Algorithm Description

Figure 1 represents the high-level workflow of the algorithm in how the optimal hyperparameters are selected. The initial hyperparameter space is set with a range of values for batch size, neurons, layers, Optimizer, activation function, last layer activation function, and loss to create the default population. For the custom module, penalty and bonus are initialized with a default value, null. The initial population is created using the population selection module that returns the optimal hyperparameter space, to begin with. Every generation starts with the default population and progresses with offspring through crossover and mutation.

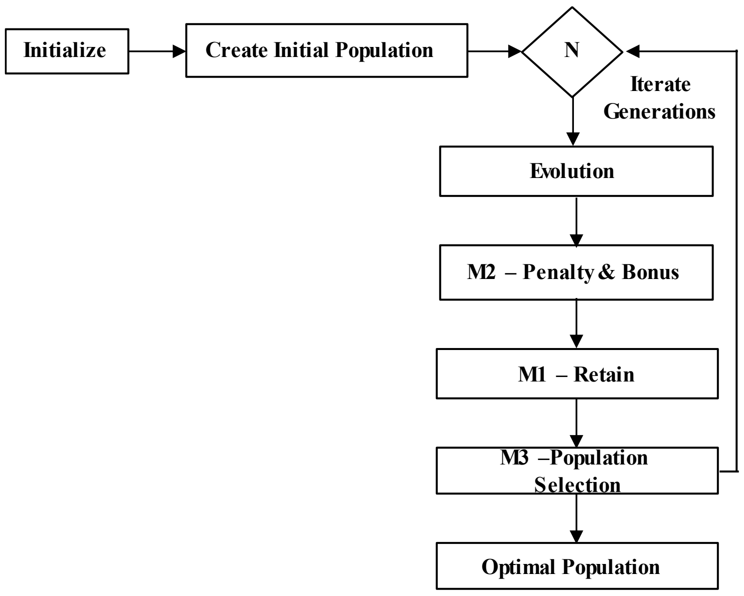

Fig. 1. Algorithm workflow.

Accuracy of the Neural Network (NN) model obtained from Module 3, population selection is used to determine the parents for creating the child Neural Networks (NNs). Parent NNs with higher accuracy are preferred. Since the networks are created using random hyperparameters controlled by the population selection module and grow based on accuracy, there is a probability of getting the same NNs for crossover. In the case of the same NNs, a random function is used to select different parent NNs in corner scenarios. Selected parents NNs create new child NNs through the crossover and mutation. If the chosen parents result in early convergence, a penalty is applied. Otherwise, a bonus is applied.

The bonus and penalty are passed as M2 values using messenger RNA (mRNA). The mRNA is added to NNs that pass-through penalty or bonus criteria. To increase the mutation rate and the diversity of the parent's selection, a random part of discarded NNs after crossover and mutation are reused for creating future offspring. This process involves prioritizing the networks with M2 values and any processed neural network setup by the population selection module. After iterating over the selected Number of generations and populations, the algorithm returns the optimal hyperparameter values with the highest accuracy.

8. Experimental Results and Discussion

In this section, we present the experimental results and comparative analysis of our model's performance with existing work [9, 31-34].

8.1. Hyperparameters range

Table 2 represents the input hyperparameters range used for optimization. The optimal hyper parameters of CGA are epoch = 300, batch size = 20, number of layers = 4, Optimizer = Adam, Activation function = relu, last layer activation function = sigmoid, loss = binary cross entropy and dropout rate = 0.2. We used accuracy as the metrics for evaluation.

Table 2. Hyperparameters space and optimized.

Hyper parameters	Input Range (default)	CGA Model values
Number of epochs	[50,100,150,200,250,300,350,400]	300
Batch size	[10,20,30,40,50,60,70,80]	20
Number of layers	[2,4,6,8,10,12,14,16]	4
Number of Neurons	[2,5,10,15,20,25,30,35]	40
Optimizers	["adam", "sgd", "rmsprop", "Adadelta", "Adamax", "Nadam"]	Adam
Activations	["Relu", "sigmoid", "tanh"]	Relu
Last layer activation function	Sigmoid	Sigmoid
Losses	Binary Cross Entropy	Binary Cross Entropy
Metrics	Accuracy (Test Set 1)	0.992
	Accuracy (Test Set 2)	0.958
Regularizer	Dropout	0.2

Figures 2 to 5 show the training and validation, accuracy, and loss of the CGA tuned model for test set 1 and test set 2. We used a validation split of 0.3 to avoid overfitting and underfitting problems. It is noted that the loss decreases as the accuracy increases for both training and validation sets.

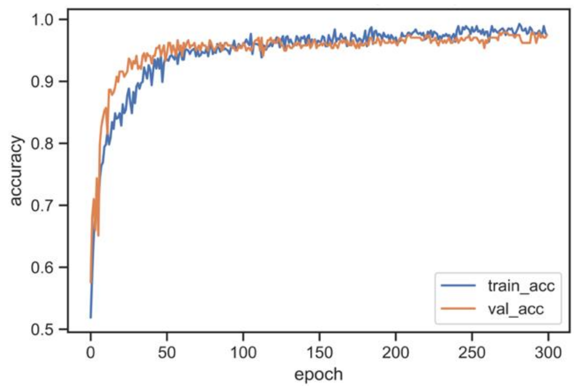

Fig. 2. Test Set 1 – Training Accuracy of CGA.

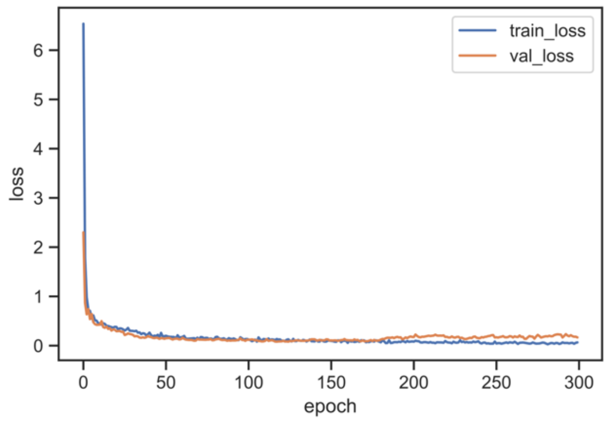

Fig. 3. Test Set 1 – Training Loss of CGA.

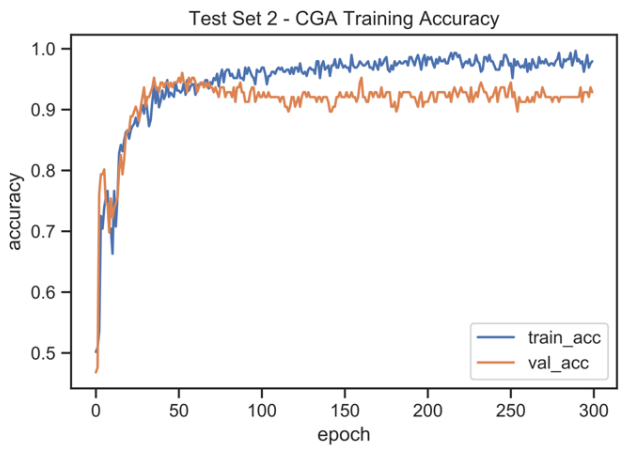

Fig. 4. Test Set 2 – Training accuracy of CGA.

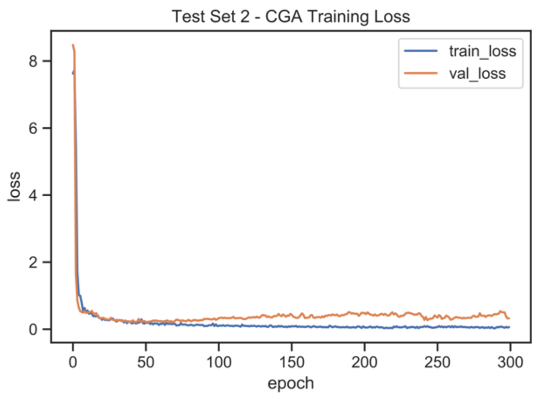

Fig. 5. Test Set 2 – Training loss of CGA.

8.2. Comparison with existing research works

We compared the CGA tuned deep learning model with the existing research works [9, 31-34]. Cresci et al. [31] categorized features into account and tweel-level. Tweet type and tweet contents are used as DNA for analysis. In our work, we applied the chi-squared technique that returned 33 important features for prediction and accuracy as the final metrics. Miller et al. [32] used vectors created from 126 features that are extracted from both tweets and accounts. Ahmed and Abulaish [33] used 14 behavioral -based generic statistical features like retweets, mentions, hashtags, and URLs. Yang et al. [34] grouped twenty-five features into six categories: automation-based, timing-based, content-based, profile-based features, neighbor-based, and graph-based features. Davis et al. [35] grouped more than a thousand features in 6 main classes: user, sentiment, friend, content, temporal, and network.

We evaluated our CGA tuned model using the test set 1 and test 2 to find accuracy, f measure, precision, and recall. Table 3 compares the results of test set 1 and test set 2 with existing spambot detection techniques. For the test set 1, the CGA model outperformed others with an accuracy of 0.991 and a recall of 0.976. For test 2, the CGA model outperformed existing methods with an accuracy of 0.958, recall of 0.960, and F measure of 0.960. We found that the remaining metrics showing competitive performance with the existing techniques.

Table 3. Performance comparison of spambot detection techniques applied on CRESCI 2017 dataset with existing research.

		Test Set #1			
Technique	**Type**	**Accuracy**	**F measure**	**Precision**	**Recall**
Human annotators	Manual	0.698	0.123	0.267	0.080
Wei et al. [9]	Supervised	0.961	0.963	0.940	0.976
Cresci et al. [31]	Unsupervised	0.976	**0.977**	**0.982**	0.972
Miller et al. [32]	Unsupervised	0.526	0.435	0.555	0.358
Siddiqui et al. [12]	Supervised	0.734	0.288	0.471	0.208
Ahmed and Abulaish [33]	Unsupervised	0.943	0.944	0.945	0.944
Yang et al. [34]	Supervised	0.506	0.261	0.563	0.170
CGA	Supervised	**0.991**	0.974	0.976	**0.976**
		Test Set #2			
Human annotators	Manual	0.829	0.570	0.647	0.509
Wei et al. [9]	Supervised	0.929	0.926	0.933	0.919
Cresci et al. [31]	Unsupervised	0.929	0.923	**1.000**	0.858
Miller et al. [32]	Unsupervised	0.481	0.370	0.467	0.306
Siddiqui et al. [12]	Supervised	0.922	0.761	0.635	0.950
Ahmed and Abulaish [33]	Unsupervised	0.923	0.923	0.913	0.935
Yang et al. [34]	Supervised	0.629	0.524	0.727	0.409
CGA	Supervised	**0.978**	**0.988**	0.994	**0.982**

8.3. Comparison with existing optimization techniques

We compared CGA with other optimization techniques such as Random Search, Bayesian Optimization, and Optimization using Genetic Algorithms. Grid Search is ignored since it traverses through all the parameters, which could take days or months. For the Genetic Algorithm, we used the TPOT classifiers Neural Network Module. We used 50 populations and 50 generations, both the Genetic Algorithm and the proposed Custom Genetic Algorithm, without any caveats around the selection for a fair comparison.

All the sparse values are cleaned, and the dataset contains complete values for all features after the data preprocessing step. So, we considered accuracy and F-measure as our metrics. Weighted f1-score is ignored to give equal importance to precision and recall.

Of all the techniques, TPOT took very high execution, followed by Bayesian and Random search. CGA took the least execution time. However, we couldn't calculate the accurate timings since the execution was paused multiple times due to the Cloud's busy machines. Theoretically, if there are n configurations, each with an error factor of ei for i = 1 … n. The allocation and selection of configurations (hyperparameter spaces) by random, Bayesian, grid, and TPOT will take more time since they traverse through the spaces on specific set criteria independent of optimal configs traversal. However, CGA picks only the best parameters and traverses through the spaces/configurations that perform better with a high weightage to the top 50%.

Figures 6 to 9 show the ROC Curve for Random Search, Bayesian Search, Genetic Algorithm, and the proposed Custom Genetic Algorithm (CGA). The CGA ROC graphs clearly show the model's performance in identifying true positives and false negatives and outperforms the other optimization techniques with area under the curve value of 0.98 and 0.94 for Test Set 1 and Test 2 respectively.

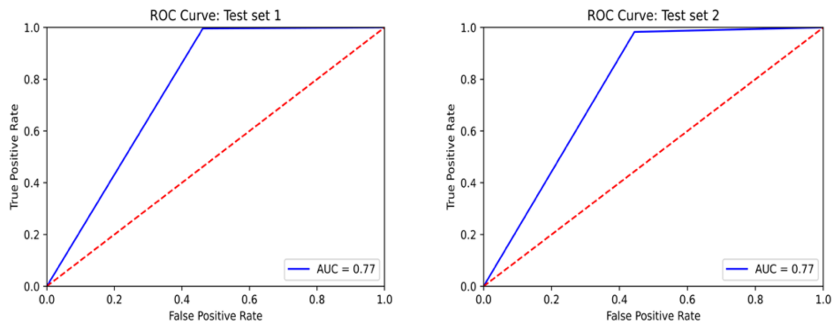

Fig. 6. Random search - ROC Curve for Test Set 1 and Test Set 2.

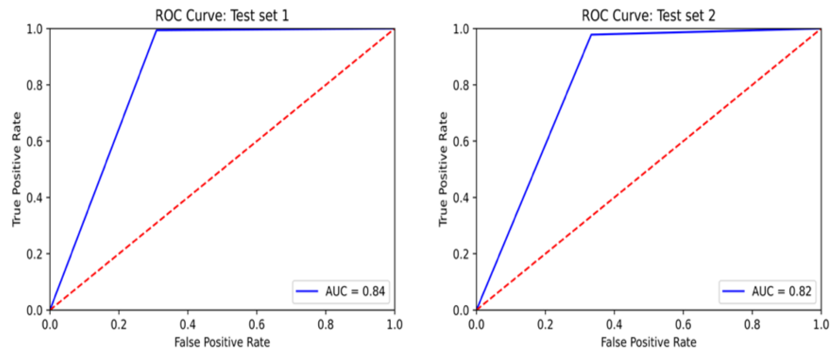

Fig. 7. Bayesian search - ROC Curve for Test Set 1 and Test Set 2.

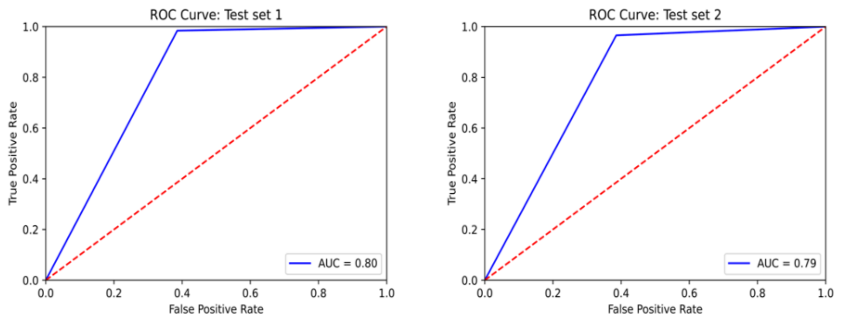

Fig. 8. Genetic algorithm (TPOT) - ROC curve for Test Set 1 and Test Set 2.

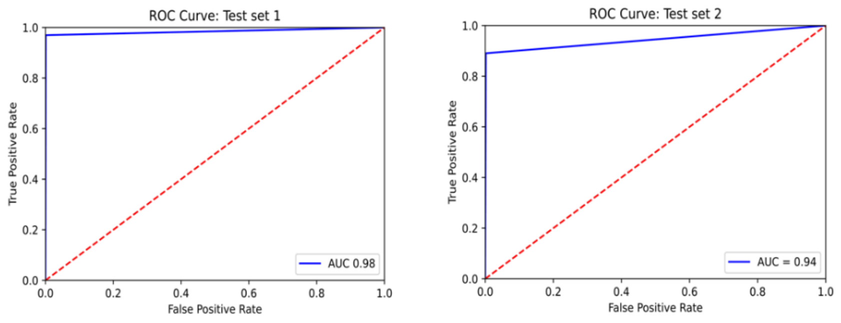

Fig. 9. Custom genetic algorithm - ROC curve for Test Set 1 and Test Set 2.

Table 4 compares the metrics such as Accuracy, F measure, Precision, and Recall for all the optimization techniques. CGA tuned model outperformed all the existing techniques and methods on both sets with an accuracy of 0.991 for Test Set 1 and 0.958 for Test Set 2, and F-measure of 0.974 for Test Set 1 and 0.960 for Test Set 2, respectively.

Table 4. Performance comparison of spambot detection techniques applied on CRESCI 2017 dataset with existing optimization techniques.

		Test Set #1		
Technique	**Accuracy**	**F measure**	**Precision**	**Recall**
Random Search	0.864	0.905	0.83	0.995
Bayesian Search	0.954	0.97	0.946	0.995
TPOT	0.763	0.822	0.704	0.988
CGA	0.991	0.974	0.976	0.976
		Test Set #2		
Random Search	0.818	0.886	0.801	0.991
Bayesian Search	0.944	0.945	0.907	0.987
TPOT	0.731	0.822	0.704	0.987
CGA	0.978	0.988	0.994	0.982

9. Conclusion and Future Work

In this paper, we proposed a Custom Genetic Algorithm for tuning hyperparameters. Finding optimal parameters for any deep learning model is a tedious and time-consuming task. We identified that our proposed CGA algorithm reduces the search time to a greater extent in finding the optimal hyperparameters. Experimental results show tuned models outperforming existing techniques. We believe that the proposed model can be applied to any classification dataset with minimal changes. As future work, we are planning to apply the CGA to wider datasets for further analysis and improvement.

Nomenclatures

a_{cc}	Accuracy
N	Network
N_g	Number of Generations
N_o	Network Object
P	Population

Abbreviations

CGA	Custom Genetic Algorithm
CNN	Convolutional Neural Network
DL	Deep Learning
LSTM	Long Short Term Memory
ML	Machine Learning
mRNA	Messenger Ribonucleic Acid
NN	Neural Network

References

1. Abbas, A.; Khan, M.U.S.; Ali, M.; Khan, S.U.; and Yang, L.T. (2015). A cloud based framework for identification of influential health experts from Twitter. *IEEE* 12*th Intl Conf on Ubiquitous Intelligence and Computing and* 2015 *IEEE* 12*th Intl Conf on Autonomic and Trusted Computing and* 2015 *IEEE* 15*th Intl Conf on Scalable Computing and Communications and Its Associated Workshops* (*UIC-ATC-ScalCom*), 831-838.
2. Khan, M.U.S.; Ali, M.; Abbas, A.; Khan, S.U.; and Zomaya, A.Y. (2016). Segregating spammers and unsolicited bloggers from genuine experts on Twitter. *IEEE Transactions on Dependable and Secure Computing*, 15(4), 551-560.
3. Ferrara, E.; Varol, O.; Davis, C.; Menczer, F.; and Flammini, A. (2016). The rise of social bots. *Communications of the ACM*, 59(7), 96-104.
4. Wu, B.; Liu, L.; Yang, Y.; Zheng, K.; and Wang, X. (2020). Using improved conditional generative adversarial networks to detect social bots on Twitter. *IEEE Access*, 8, 36664-36680.
5. Loyola-González, O.; Monroy, R.; Rodríguez, J.; López-Cuevas, A.; and Mata-Sánchez, J.I. (2019). Contrast pattern-based classification for bot detection on twitter. *IEEE Access*, 7, 45800-45817.
6. Karthikayini Thavasimani; and Srinath, N.K. (2020). Deep learning techniques: A case study on comparative analysis of various optimizers to detect bots from CRESCI-2017 dataset. *International Journal of Advanced Science and Technology*, 29(04), 10040 -10053.
7. Cai, C.; Li, L.; and Zengi, D. (2017). Behavior enhanced deep bot detection in

social media. *IEEE International Conference on Intelligence and Security Informatics* (*ISI*), 128-130.

8. Antoun, W.; Baly, F.; Achour, R.; Hussein, A.; and Hajj, H. (2020). State of the art models for fake news detection tasks. *IEEE International Conference on Informatics, IoT, and Enabling Technologies* (*ICIoT*), 519-524.
9. Wei, F.; and Nguyen, U.T. (2019). Twitter bot detection using bidirectional long short-term memory neural networks and word embeddings. *First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications* (*TPS-ISA*), 101-109.
10. Fernquist, J.; Kaati, L.; and Schroeder, R.; (2018). Political bots and the swedish general election. *IEEE International Conference on Intelligence and Security Informatics* (*ISI*), 124-129.
11. Khalil, H.; Khan, M.U.; and Ali, M. (2020). Feature Selection for Unsupervised Bot Detection. 3*rd International Conference on Computing, Mathematics and Engineering Technologies* (*iCoMET*), 1-7.
12. Siddiqui, H.; Healy, E.; and Olmsted, A. (2017). Bot or not. 12*th international conference for internet technology and secured transactions* (*ICITST*), 462-463.
13. Liu, A. (2000). DeBOT - an approach for constructing high performance, scalable distributed object systems (panel session). *Proceedings of the* 22*nd International Conference on Software Engineering*, 782.
14. Kiran, K.; Manjunatha, C.; Harini, T.S.; Shenoy, P.D.; and Venugopal, K.R. (2019). Identification of anomalous users in Twitter based on user behaviour using artificial neural networks. *IEEE* 5*th International Conference for Convergence in Technology* (*I*2*CT*), 1-5.
15. Dickerson, J.P.; Kagan, V.; and Subrahmanian, V.S. (2014). Using sentiment to detect bots on twitter: Are humans more opinionated than bots? *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (*ASONAM* 2014), 620-627.
16. Lingam, G.; Rout, R.R.; and Somayajulu, D.V.L.N. (2018). Detection of social botnet using a trust model based on spam content in Twitter network. *IEEE* 13*th International Conference on Industrial and Information Systems* (*ICIIS*), 280-285.
17. Heredia, B.; Prusa, J.D.; and Khoshgoftaar, T.M. (2018). The impact of malicious accounts on political tweet sentiment. *IEEE* 4*th International Conference on Collaboration and Internet Computing* (*CIC*), 197-202.
18. Ranjit, M.P.; Ganapathy, G.; Sridhar, K.; and Arumugham, V. (2019). Efficient deep learning hyperparameter tuning using cloud infrastructure: Intelligent distributed hyperparameter tuning with Bayesian optimization in the cloud. *IEEE* 12*th International Conference on Cloud Computing* (*CLOUD*), 520-522.
19. Neary, P. (2018). Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning. *IEEE International Conference on Cognitive Computing* (*ICCC*), 73-77.
20. Cho, H.; Kim, Y.; Lee, E.; Choi, D.; Lee, Y.; and Rhee, W. (2020). Basic enhancement strategies when using Bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE Access*, 8, 52588-52608.
21. Goel, T.; Murugan, R.; Mirjalili, S.; and Chakrabartty, D.K. (2021). OptCoNet:

an optimized convolutional neural network for an automatic diagnosis of COVID-19. *Applied Intelligence*, 51(3), 1351-1366.

22. Choi, D.; Cho, H.; and Rhee, W. (2018). On the difficulty of DNN hyperparameter optimization using learning curve prediction. *In TENCON* 2018-2018 *IEEE Region* 10 *Conference*, 0651-0656.
23. Aich, S.; Chakraborty, S.; and Kim, H.-C. (2019). Convolutional neural network-based model for web-based text classification. *International Journal of Electrical and Computer Engineering*, 9(6), 5785-5191.
24. Ugli, I.K.K.; Aich, S.; Ryu, H.; Joo, M.I.; and Kim, H.-C. (2021). Detection of distracted driving using deep learning. *International Conference on Future Information & Communication Engineering*, 12(1), 29-32.
25. Chakraborty, S.; Aich, S.; and Kim, H.C. (2020). Detection of Parkinson's disease from 3T T1 weighted MRI scans using 3D convolutional neural network. *Diagnostics*, 10(6): 402.
26. Antonio, C. (2021). Sequential model based optimization of partially defined functions under unknown constraints. *Journal of Global Optimization*, 79(2), 281-303.
27. Kong, W.; Dong, Z.Y.; Luo, F.; Meng, K.; Zhang, W.; Wang, F.; and Zhao, X. (2017. Effect of automatic hyperparameter tuning for residential load forecasting via deep learning. *Australasian Universities Power Engineering Conference* (*AUPEC*), 1-6.
28. Yi, H.; and Bui, K.-H.N. (2020). An automated hyperparameter search-based deep learning model for highway traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 22(9), 5486-5495.
29. Bui, K.H.N.; and Yi, H. (2020). Optimal hyperparameter tuning using meta-learning for big traffic datasets. *IEEE International Conference on Big Data and Smart Computing* (*BigComp*), 48-54.
30. Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(1), 6765-6816.
31. Cresci, S.; Di Pietro, R.; Petrocchi, M.; Spognardi, A.; and Tesconi, M. (2016). DNA-inspired online behavioral modeling and its application to spambot detection. *IEEE Intelligent Systems*, 31(5), 58-64.
32. Miller, Z.; Dickinson, B.; Deitrick, W.; Hu, W.; and Wang, A.H. (2014). Twitter spammer detection using data stream clustering. *Information Sciences*, 260, 64-73.
33. Ahmed, F.; and Abulaish, M. (2013). A generic statistical approach for spam detection in online social networks. *Computer Communications*, 36(10-11), 1120-1129.
34. Yang, C.; Harkreader, R.; and Gu, G. (2013). Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Transactions on Information Forensics and Security*, 8(8), 1280-1293.
35. Davis, C.A.; Varol, O.; Ferrara, E.; Flammini, A.; and Menczer, F. (2016). Botornot: A system to evaluate social bots. *Proceedings of the* 25*th International Conference Companion on World Wide Web*, 273-274.