RunPool: A Dynamic Pooling Layer for Convolution Neural Network

Huang Jin Jie\(^1,\)*, Putra Wanda\(^1,2,\)*

\(^1\)School of Computer Science & Technology, Harbin University of Science and Technology, Harbin, China
\(^2\)University of Respati Yogyakarta, St. Sol, Depok, Yogyakarta, Sleman, 55281, Indonesia

1. INTRODUCTION

In recent years, deep learning (DL) has achieved remarkable performance in various computer applications, especially in computer vision problems [1–3]. Convolutional neural networks (CNN) is one of the popular DL algorithms for solving image recognition [4]. The architecture is based on a supervised learning model that consists of several convolutional layers and pooling layers. In neural network computation, it is required to extract more robust features against position or movement for object recognition. CNN architecture harnesses pooling layers to decrease the size of the feature map and extract features [4].

Those pooling layers are a crucial part of CNN to achieve an efficient training process. To obtain better accuracy with a small loss, subsampling plays an important role. Commonly, the CNN pooling techniques are max pooling and average pooling. In the operation process, max pooling takes the maximum score in the pooling window, and average pooling selects the average score of the pooling window. However, the current study picks pooling types without much consideration.

A paper reveals that the two conventional pooling models are not optimal [5]. In some cases, the average pooling diminishes critical information in strong activation so that it can decrease the performance on CNN. On the other hand, the max pooling shortcomings remain because it ignores all crucial information except the most significant value. It can degrade an opportunity to calculate other informative features in large and dynamic input data. A comparison with max or average pooling schemes cannot predict which parts of the input will be selected.

A paper proposes the pooling technique by replacing the conventional deterministic pooling with a stochastic model to increase pooling layer performance. It randomly selects the activation within each pooling region according to a multinomial distribution [5]. The current study introduces Deep Generalized Max Pooling that balances the contribution of all activations of a region by reweighting all descriptors [6]. Another paper proposes, alpha-pooling, which utilizes a trainable parameter \(\alpha\) to decide the type of pooling. It is a type of pooling scheme, including max pooling and average pooling as a particular case. Demonstrated by experiment, alpha-pooling can improve the accuracy of image recognition problems. They compared several pooling layers and found that max pooling was not the optimal pooling method [7].

Besides, there are many other pooling schemes, including geometric and harmonic average. It is still challenging to find the best pooling method. Thus, it is required to keep in line with the philosophy of DL. In dynamic neural network architecture, it is not practically possible to find a proper pooling technique for the layers. It is the primary reason why various pooling cannot be applied in the dynamic and multidimensional dataset. To deal with the limitations, it needs to construct an optimal pooling method as a better option than max pooling and average pooling. Therefore, we introduce a dynamic pooling layer called RunPool to train the convolutional neural network (CNN) architecture. RunPool pooling is proposed to regularize the neural network that replaces the deterministic pooling functions. In the final section, we test the proposed pooling layer to address classification problems with online social network (OSN) dataset.
predict which parts of the input will be selected as the most appropriate pooling value.

In this paper, we introduce a generic pooling layer as an option to construct CNN architecture. Based on our survey, it is the one pooling layer that utilizes network parameters in each layer to determine pooling value in the hidden layer. Notably, we present the main contributions of our work, particularly in CNN as follows:

1. We introduce a new pooling layer in the CNN architecture. Different from max and average pooling, this pooling implements a magnetic field concept in calculating the pooling layer. It runs by dynamically changing the pooling layers’ values for each convolution in the hidden layer.

2. We construct the pooling layer by obtaining several hyperparameter in each layer. Then, this function calculates the parameters with a range distribution. By establishing the concept, we get the state-of-the-art in CNN architecture. By demonstrating the architecture, we harvest higher accuracy and a smaller loss than conventional CNN architecture max pooling and average pooling layer.

3. Notably, this function is different from traditional pooling that obtains the highest valued activation out of the pooling window or takes an average value of a pooling window. The proposed pooling layers try to find appropriate pooling value to achieve efficient computation by calculating several informative parameters in each layer. It is a novel pooling approach in CNN architecture to reduce overfitting while training a model.

The rest of the paper is structured as follows: Section II gives insight into the proposed technique, Section III presents the experimental setup that consists of case study in a social network, dataset, and data preprocessing. Section IV describes the result and discussion. Section V offers the conclusion and highlights some open research questions in CNN architecture.

2. PROPOSED TECHNIQUE

As the hot topic of research, DL is becoming an increasingly popular approach for solving diverse applications. The CNN algorithms imitate the structure and function of the human brain [8]. In recent years, CNN has achieved state-of-the-art results on a variety of challenging problems in computer vision and speech recognition. Research communities have implemented this concept to deal with many computer assignments. Several papers explore the use of deep belief networks (DBN) for authorship verification by a Gaussian–Bernoulli in DBN [9], face verification with a Deep Neural Network (DNN) [10], and document recognition [11].

2.1. Convolutional Neural Network

Most of the current research in DL propose CNN architecture to deal with various problems, including image, voice, video, and object detection [8]. In this paper, we construct a CNN architecture by adding a new pooling layer in the neural network. We mention the graph as dynamic CNN, a CNN graph with a RunPool pooling function that has three layers in the network, including an input layer, a hidden layer, and an output layer. In the input layer, the network layer projects the feature extraction as matrices input. To compute the feature extraction result, we adopt convolution layer processing with weight sharing. Thus, it is unnecessary to optimize many parameters because the method can construct convergence faster.

This architecture adopts the convolution to address multiple input feature maps and to produce stacked feature maps of increasing layers. In this experiment, we define a feature map of \( f_i \) with \( F_i \). We denote the total of \( n \) feature maps in layer \( i - 1 = F_{i,1}^{1}, \ldots, F_{i,1}^{n} \). Layer \( i \) has a specific filter size \( l \) with \( F_{i,j}^{l} \) feature map. The map \( F_{i,j}^{l} \) has \( j \) parameter which represents the index of a feature map in layer \( i \). We calculate each feature map \( F_{i,j}^{l} \) by convolving it with a matrix \( M_{i,j}^{l} \), thus it yields the results:

\[
F_{i,j}^{l} = \sum_{k=1}^{n} M_{i,j}^{l,k} \ast F_{i-1}^{k,1}
\]  

The model defines the computation of the convolution process between filters and input matrices. The model convolves a set of filters form \( F \in \mathbb{R}^{bc(de)} \) with the input matrix \( S \in \mathbb{R}^{bc(d)} \). The goal of the convolutional layer is to compute feature extraction.

After convolution operation, we construct a pooling layer to obtain the most informative values in the hidden units. Pooling aims to convert the joint feature representation into a more valuable point that preserves important information and diminishing irrelevant details. In operation, the pooling reduces the resolution of the feature maps over the local neighborhood of the previous layer. Commonly, a CNN architecture employs conventional pooling schemes such as max pooling and average pooling to obtain a maximum or average value in the hidden layer. The max pooling chooses the biggest element in each pooling region as:

\[
y_{kij} = \max_{(p,q)\in R_j} x_{kpq}
\]

where \( y_{kij} \) is the output of the pooling operator related to the \( k_{th} \) feature map, \( x_{kpq} \) is the element at \( (p,q) \) within the pooling region \( R_j \) which represents a local neighborhood around the position \( (i,j) \). The average pooling obtains the arithmetic mean of the elements in each pooling region as:

\[
y_{kij} = \frac{1}{|R_j|} \sum_{(p,q)\in R_j} x_{kpq}
\]

On the average pooling function, \( |R_j| \) represents the size of the pooling region \( R_j \). Although those two types of pooling functions produce a good result on some datasets, it is still unknown, which will work better for dealing with a new task with a different dataset. Thus, it is a little bit complex to choose the pooling operator.

In some image recognition problems, both the max pooling and average pooling have their shortcomings. Figure 1 illustrates the drawbacks of max pooling and average pooling in detecting a diagonal line.

In the pooling region, max pooling only considers the maximum element and ignores others. In some cases, it will lead to an unacceptable result because if the most elements consist of high magnitudes, the distinguishing feature vanishes after max pooling. On the
other hand, the average pooling gets worse if there are many zero elements on the input features; the characteristic of the feature map will be reduced largely [12]. Therefore, as a new way to downsampling the features, we consider replacing the deterministic pooling function with a stochastic scheme. Instead of using the standard pooling method, we introduce a new pooling function by adopting a nondeterministic pooling concept.

### 2.2. Main Idea

To construct a pooling function with a stochastic pooling paradigm, we take a magnetic field problem as the basic idea. The main problem of this function is how to establish a CNN architecture by optimizing a new pooling layer. We are inspired by a model that adopts the dynamic k-max pooling to compute the engineered feature in each layer. By choosing the hyperparameters in each layer, the function calculates the pooling value to obtain a pooling sequence in 1D dataset [13].

To implement the magnetic field concept, we calculate the pooling layer with the Gaussian Law theory (Gaussian random variable). At the initial step, we compute the pooling function by generating random numbers from a Gaussian distribution. We employ the random technique because the randomness comes from atmospheric noise. In several computer applications, it is better than the common pseudo-random algorithms. A normal distribution has a bell-shaped density curve described by its mean \( \mu \) and standard deviation \( \sigma \). To calculate the Gaussian distribution of the range number, we compute the function:

\[
f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

In formula 2, \( \mu \) represents the mean, median, mode, or expectation of the distribution. Notation \( \sigma \) reflects the standard deviation, and \( \sigma^2 \) is the variance. To construct the pooling function, we multiply several numbers of Gauss random with \( k_{map} \) in each layer. It calculates the \( k_{map} \) as a dynamic value, and it depends on the current parameters. Finally, this function will compare the \( k_{map} \cdot f(x|\mu, \sigma^2) \) with the constant \( k_{top} \) to obtain the final \( k \) value for the current pooling layer.

### 2.3. RunPool Pooling

CNN is a famous DL architecture that uses a lot of identical copies of the same neuron. The process runs extensive computation with a little number of parameters and has many interleaved convolutional and pooling layers over the network. The CNN layer receives a single input (the feature maps) and computes feature maps as its output by convolving filters across the feature maps. CNN can ignore the aspect of feature engineering with automatic representation.

One critical layer in CNN training is the pooling layer. The pooling can act as a generalizer of the lower-level data in the NN training. It is a sliding window technique that utilizes several kinds of statistical functions to get the contents on each window. There are two common types of pooling techniques, max pooling and average pooling. Max pooling employs the max function to obtain informative materials, while the average pooling harnesses the statistical mean function to take the contents of the window. Principally, a pooling function simplifies the output features by performing nonlinear computation and reducing the number of parameters.

Instead of using the conventional pooling layers, we construct a method that dynamically changes the pooling values based on the current network parameter. It takes the length of the input vector of the network to obtain a feature map according to the current parameters in each layer. The basic idea of the function is how to calculate network elements of the current map pooling and to find appropriate pooling value by comparing it with a constant pooling value. It is to recompute the pooling value using the layer parameters and learn a CNN hierarchy of feature orders. Figure 2 depicts

\[\text{Figure 1} \quad \text{Pooling process of input feature that illustrates the drawbacks of max pooling and average pooling on CNN.}\]
The hidden layer runs the convolution process by using filters and input matrices. The dynamic CNN computes the forward pass with the following function:

\[ y_i^n = \sum_j k_{ij}^n \cdot x_i^n \]  

The hidden layer runs the convolution process by using filters and input matrices. In the model, a set of filters form \( F \in \mathbb{R}^{d \times m \times l} \) will be convolved to the input matrix \( S \in \mathbb{R}^{n \times d} \). The convolutional layer aims to compute feature extraction, such as word sequences throughout the training process. The RunPool computes the matrices conversion based on the feature extraction process. Table 1 describes the mathematical notation for the RunPool function.

We construct the function to allow a smooth extraction of higher order and to adapt with a longer range element at the hidden layer. The model employs the \( k \) value at intermediate layers (hidden layer). The dynamic CNN calculates the forward pass with the following function:

\[ y_i^n = \sum_j k_{ij}^n \cdot x_i^n \]  

In the above formula, the first network computes the features maps as the input layer to the CNN. In Equation (4), \( x_i^n \) is the \( i \)-th input feature map of the sample \( n \) and \( k_{ij}^n \) is the \( ij \)-th input kernel of the sample \( n \). Then, \( j - \theta \) becomes the output feature map of sample \( n \) networks. Backward and forward operation run simultaneously in one iteration. The CNN calculates the backward pass with the formulas:

\[ \frac{\partial l}{\partial x_i^n} = \sum_j \left( \frac{\partial l}{\partial y_j^n} \right) \ast (k_{ij}^n) \]  

\[ \frac{\partial l}{\partial k_{ij}^n} = \left( \frac{\partial l}{\partial y_j^n} \right) \ast x_i^n \]  

In the backward pass process with Equation (6), the layer will compute the gradient of the \( l \) (loss function) with respect to \( x_i^n \). The values of the gradient are calculated by the partial derivative function \( \frac{\partial l}{\partial x_i^n} \) and are passed to the first layer of the network. Then Equation (7) is used to calculate the gradient of the loss function with respect to \( k_{ij}^n \).

**RunPool Function:** Assume \( k \) be a function of the length of the input and the depth of the network as the computing model. \( l \) is a current convolution in the hidden layer, \( L \) is the total number of convolution layers. At the initial process, we determine \( k_{top} \) as a constant value for the pooling layer calculation. The function needs input matrices and some padding to calculate the extracted features. It computes \( k \) by calculating network parameters in each window.

In CNN process, each convolutional filter produces a matrix of hidden variables. The pooling layer is used to reduce the size of the matrix. Max pooling is a procedure that takes an \( \text{Nin} \times \text{Nin} \) input matrix and produces a smaller output matrix (feature maps) \( \text{Nout} \times \text{Nout} \). This is achieved by dividing the \( \text{Nin} \times \text{Nin} \) square into \( N_{\text{out}} \) pooling regions \( P_{ij} \):

\[ P(i, j) \subset \{1, 2, ..., N_m\}^2 \text{ for each } (i, j) \in \{1, ..., N_m\}^2 \]

Pooling layers are a downsampling layer that combines the output of layer to a single neuron. The model calculates the input data in the hidden layer with \( n_i \) represents \( i \)-th layer, \( f \) represents filters, and \( s \) reflects stride in the process. Then, the output dimension of the pooling layer will be calculated:

\[ \max_{N_{\text{in}} \times N_{\text{in}}} = n_1 \ast n_2 \ast n_c = \left[ \frac{n_1 - f}{s} + 1 \right] \times \left[ \frac{n_2 - f}{s} + 1 \right] \times n_c \]
Inspired by DCNN model in [13] with one-dimensional convolution, we construct a function to obtain appropriate k-max pooling value by using more layer parameters with dynamic values. Then, we take Gauss distribution to calculate the final $k_{map}$. In forward propagation, the $N_k \times N_p$ part will produce a single "winning unit" value. We construct a function to obtain value at each layer instead of choosing the exact maximum value from $l$ layer. Hence, we get the dynamic winning out by calculating the simple pooling function to obtain $k$ value of layer $l$. We calculate $k_{map}$ with a function to get the $k_l$ value (value of layer $l$) as:

$$k_{map} = \left( \frac{C(L - b)}{L} \right) \quad k_l = \max (k_{top}, k_{map}) \quad (9)$$

From the function above, the RunPool selects $k_{top}$ as the current pooling value if $k_{top} \geq k_{map}$ or selects $k_{map}$ otherwise. The aim of the RunPool Equation (9) is to find $k_l$ value by comparing $k_{top}$ constant and $k_l$ produced by parameter calculation. It is a dynamic pooling function that calculates window parameters to take pooling value in each hidden layer. The model uses wide convolution in each window, to ensures that all weights get the entire parameter space. To obtain final value of $k_{map}$, we multiply it with a Gaussian random value $r$ to produce a positive integer value $\mathbb{Z}^+$ as $k_{map} = r \cdot r \in \mathbb{Z}^+$, $r \geq 0$. Figure 3 depicts how to get a pooling value in the RunPool function.

The main idea of the function is how to take some parameters of the current layer to obtain appropriate pooling value in the current hidden layer. In the RunPool function, $l$ is some current convolution in the hidden layer, $L$ is the total number of convolution layer, $k_{top}$ is the fixed max pooling parameter for the topmost convolutional layer. To obtain optimal max pooling value, we calculate the RunPool function with Algorithm 1.

Algorithm 1 describes how to calculate the network parameters to take $k_{map}$ value. The RunPool tries to find appropriate $k_{map}$ value based on the parameters of the current window. It harnesses parameter $k$ to obtain optimal k-max pooling value. In the proposed scheme, the model calculates the window parameter to obtain dynamic $k_{map}$ value. The $k_{map}$ function is derived from the input or other aspects of the network. The function calculates a pooling window by turning different length of vectors into the same length. The convolutional layers run the feature extraction to capture better representation in the hidden layer. To calculate $k_{map}$, the function obtains length of input matrices $C = C_1, C_2, ... C_n$ as the representation of 1D dataset including text, character, and numeric. Finally, we get final integer value of $k_{map}$ after multiplying it with a Gaussian random variable.

We activate the pooling function between the convolution and the fully connected layer (FC). FC is an efficient way of learning nonlinear combinations to classify the data into target (categories) $y = y_1, y_2$. It connects each of neuron layer to every neuron in another layer. The input to the FC layer is the flattened output from the final pooling or convolutional layer and then fed into the FC layer.

For calculating error loss of the model, the experiment utilizes cross-entropy (CE) loss function. We calculate the loss function with the function:

$$J(\theta) = -\frac{1}{T_{mb}} \sum_{T=0}^{T_{mb}-1} \sum_{t=0}^{F_{input}-1} \frac{\partial f}{y^{(i)}} \ln h_f^{(i)} \quad (10)$$

The above criterion function consists of $T_{train}$ as the number of training examples, $T_{mb}$ represents the number of training examples in a mini-batch, $h_f^{(i)}$ represents the hidden neuron of each layer, and $\frac{\partial f}{y^{(i)}}$ represents the derivative. In this experiment, we train the model by calculating the CE in binary classification problems because it is a mostly loss function for classification problems. CE is applied to obtain a probability distribution output and is a preferred loss function for FC activation like SoftMax.

3. EXPERIMENTAL SETUP

In this study, we experiment with a large amount of 1D datasets, including numerical and text datasets. We gather a large dataset to construct a benchmark sample for the training and testing model with the dataset. In this paper, we focus on the 1D dataset, including texts, characters, and numeric data, to measure the pooling performance in training the model.

![Figure 3](image-url)
3.1. OSN Classification

In this experiment, we collect a large online social network (OSN) as the primary dataset and train it as a binary classification problem. OSN is a large environment with a large amount of data with millions of users. OSN become a popular application for sharing a lot of data items including videos, photos, and messages. The main trait of OSN is the social relation. It contains a large number of accounts. They post, send, and share any data items across the platforms. Indeed, it is easy to gather much information across the sphere. OSN is a dense interconnection among the user of the network. In reality, OSN’s model is mostly multirelational that contains a large number of nodes. Basically, in the research area, the OSN depicts the nodes by a graph. The nodes represent actors who build diverse relationship type edges that reflect actor interactions [14]. In current years, OSN becomes an active research interest in several fields [15].

In the OSN environment, analyzing the data characteristic can produce patterns of knowledge of the underlying OSN activities and hidden regularities. Link prediction, as one of the hot topics in social network research and as one of the analysis problems since a decade, drives a big amount of the researcher’s attention from various disciplines [16]. However, the OSN environment remains as security issues and become the primary concern in the fast-growing OSN. Protection technique is one of the most critical aspects of an (OSN). The shortcomings of security and privacy-preserving model put the user data at risk. Many papers propose various techniques to address the OSN security issue [17,18].

Commonly, the OSN system conducts anomaly detection by monitoring a sudden user discrepancies activity. If the users’ activities keep on changing in a period abruptly, the OSN provider catches the suspicious account up by the changing of access pattern. If it fails to gain the information and behave with unusual pattern, the abnormal can infect the system with existing fraudulent [19]. However, suspicious activity exhibits different patterns, type of data, level of interaction, and the difference in time usage. To identify the anomalies, it needs to classify in detail the features of users. In this paper, we focus on building a classification model with supervised learning by computing the OSN dataset in the training process.

3.2. Dataset

In this experiment, we provide the OSN dataset as a 1D dataset and feed the features into the supervised learning algorithm. At the first stage, we collect the OSN dataset consists of some high-level features of OSN, such as user profiles $P_u$ and relation $R_u$. User profiles $P_u$ refers to a real natural member. It is an individual account in the OSN that consists of user representation, including username, age, location, pictures information, URL information, and other attributes. $R_u$ represents the user’s social connections in the OSN. $R_u$ including how many friends are in their network.

In this paper, we define OSN as G and is represented by $G(P, R)$ where $P = \{p_1, p_2, ..., p_n\}$ is the set of user identity, and the collection of the OSN relation is described as $R = \{R_1, R_2, ..., R_n\}$. The user's relationship has a normal relation (blue lines), malicious relation (yellow lines), and fake relation (red lines). Figure 4 depicts OSN graph $G(P, R)$ with types of relation, and user properties.

![Figure 4](image_url)

3.2.1. OSN link features

Current research discusses various DL methods to conduct link prediction [20,21]. In the OSN communities, anomalies link become one of the primary concerns of OSN security research. Initially, the source of menace comes from suspicious user activities. The infected user automatically spread fake information or deceive user for targeting other regular users. If the crumbly system cannot adopt the threat model, it becomes an easy target for phishing or compromising attack. An anomaly has a diverse type of threat like vertical attack and horizontal attack. A vertical is usually a threat to the providers, and the horizontal one poses attacks to other OSN users [22].

The current study discusses fake account detection in social networks by using the learning method. DeepScan proposes malicious account detection with DL in Location-Based Social Networks (LBSN) [23]. Another scheme utilizes the similarity of the user’s friends to calculate the adjacency matrix of the graph to identify the user as benign or fake [24]. Analyzing social relations is a new way of determining anomalous users by mapping social ties and geographical among users. It calculates OSN factors like friendship, common interest, or alliance to determine chosen friends. Many researchers have proposed methods to construct the OSN security model. Therefore, in this paper, we build a learning classifier as a new way of protection by analyzing the OSN dataset.

We gather the OSN link dataset of the OSN from several public OSN, especially the undirected OSN graph. In the security case, link prediction is a crucial part of analyzing the structure in the OSN as a security scheme. Instead of exploring the entire graph, we gather the neighborhood link information by aggregating the features. For this experiment, we collect topology information of the network which is available from the data domain. We gather the undirected OSN dataset from Facebook and Flickr samples to build the benchmark dataset. Table 2 displays the dataset which is gathered for the experiment.
This corpus contains the undirected network link of OSN users. In the dataset, a node denotes a user, and an edge indicates a friendship between two users. The datasets contain a subset sample of the total friendship graph. Notably, measuring social relation with supervised learning such as CNN becomes a new approach in OSN malicious link detection. The main problem of node relation analysis is to quantify links among the user and classify the user categories. Finally, we feed the sample into the learning model to train the OSN attributes (user relations and profiles features) to classify the OSN account.

### 3.2.2. OSN profile features

In this part, we gather OSN user-profiles information as the experiment dataset that contains many features can be trained for building a classification model. The public OSN usually obtain the attributes for computing purposes. It is a way to handle the user and activity anomalies problem. Activity variations among accounts with contrast behavior from different OSN groups cause horizontal anomalies [25]. Thus, we take advantage of the vast user information as the sample to construct an effective learning model as the OSN protection.

Firstly, we define OSN as G and is represented by \( G = (P, R) \) where \( P = \{p_1, p_2, \ldots, p_n\} \) is the set of the user profile as attributes. We provide some essential user features that represent interaction among OSN users.

In this study, we provide sample OSN profiles with more than 3000 unique users. We build a dataset by decomposing some user profiles information. For the experiment dataset, the paper observes user data around 3000 OSN members. We provide 1500 data with a normal user profile and 1500 users with fake profiles label. We split the training data and testing dataset to achieve real accuracy and loss.

### 3.3. Data Preprocessing

The aim of data preprocessing is to convert the raw dataset at hand to the input vector for NN training. At the preprocessing, we separate the several features into parts to preserve the context of each token. A feature gets a different value when is extracted from the identity, hostname, location, time, and other attributes of OSN. The stage includes vectorization, normalization, handling missing values, and feature extraction.

In the data preprocessing, we split the dataset into a training and test part. For training the model, we employ the training data and evaluate the model on the validation data. After finishing the validation model, we test the final model on the test data. For testing the model, we need to use a wholly different and unseen dataset to evaluate the model with the test dataset. We set the model to be unable to gain access to any information about the test dataset.

This study suffers from a not balanced dataset because the data contains about 90%–95% that belong to the same majority class (benign account). If we conduct training on such data, it can cause the ignorance of the clarification of minority clusters (fake account). Thus, it can influence the overall accuracy of classifications. To deal with this issue, we utilize the Synthetic Minority Oversampling Technique (SMOTE) to balance the dataset [26]. It is a relevant approach if we want to obtain the similarity feature of the nodes and avoid to diminish information.

In the initial prepreprocessing, we utilize a function to convert integer and string dataset into the input matrix. We need the technique to convert the features into a vector because the input may be an array of data like integers or strings that cannot be processed in neural network training. In the final phase of preprocessing, we separate the dataset into training and testing datasets to test the model on unseen data. We have two labels for the evaluation dataset, and these are benign and malicious targets.

We feed a few noises data to make it difficult for the model to be bias-free and reflects a real condition in a classification problem. Thus, the model will consider the noisy data patterns, and it pushes the model toward overfitting in the training and testing process. After the preprocessing stage, the model feeds the features into the graph for classification. To train the model, we feed the dataset into the proposed CNN to make predictions on new unseen data with binary classification tasks.

### 4. EXPERIMENT RESULT

Based on the experimental setup, we have conducted training and testing by gathering two groups of OSN dataset. In this section, we have shown how the generic pooling layer’s performance in link classification and fake profile detection in the benchmark OSN dataset. In this part, we have displayed the accuracy and loss of the pooling scheme. The experiment also has provided an evaluation of the performance model by using cross-validation. It is to calculate Recall, Precision, F1 Score, Receiver Operating Characteristic (ROC) [27].

#### 4.1. OSN Link Classification

In the link prediction problem, we have gained a trade-off between the accuracy and the performance time. By tuning diverse hyperparameters, we have undergone the neural network performance. In the training and testing process, we have tuned epoch = 500, batch size = 50, and with learning rate (lr = 0.01). Table 3 describes the proposed CNN performance, especially in the training and testing process.

We have tested gradient descent and tune optimization algorithms with different hyperparameters by computing this sample. We have employed the gradient descent to minimize the objective function \( j(\Theta) \) with the model’s parameters \( \Theta \in \mathbb{R}^d \). The model updates the parameters in the opposite direction of the gradient of the objective

| Properties | Facebook Dataset | Flickr Dataset |
|------------|------------------|----------------|
| Size       | 63,731 vertices  | 1,715,255 vertices |
| Volume     | 817,035 edges (users) | 15,551,250 edges (links) |
| Average degree | 25.640 edges / vertex | 18.133 edges / vertex |
| Size of LCC | 63,731 vertices (users) | 1,624,991 vertices (users) |
| Gini coefficient | 64.3% | 88.2% |
| Relative edge | 93.1% | 92.2% |
| distribution entropy | 0.17702 | 0.01528 |
| Assortativity | 14.8% | 11.2% |

LCC, Largest Connected Component.
function \( \nabla \theta (\phi) \). The study has set the learning rate \( \eta (lr = 0.01) \) to determine the stride size to reach a (local) minimum. In the training process, we have gained the best value of optimization when training the model with Adam optimizer. The experiment has fed Adam optimizer into the graph with different amounts of hidden layers. The result has shown a dynamic optimizer can achieve a small loss in a large number of hidden layers.

Moreover, we have tested RunPool CNN with another gradient descent like stochastic gradient descent SGD with a momentum that also produces a competitive result. The SGD with momentum \( (m = 0.5) \) can provide a better result if we tune in larger epoch and big regulizer. However, the experiment has found that adding more hidden layers for SGD cannot reduce the loss. Using a large number of hidden layers has caused overhead computation in a particular device, especially in CPU processing. Testing the NN by adding more layers and neuron numbers of CNN could not engage in improving the predictive capability. Because of the limitation of neuron number, it enlarges computing resources. Thus, using the RunPool pooling function can increase the accuracy result without adding more hidden layers in a neural network.

In this experiment, we also calculate the Recall function to define the percentage of the datasets that belong to the malicious and identifies it as malicious. The precision determines the rate of the datasets which identifies as malicious, but actually, do not include the malicious. Finally, we want to seek a balance between Precision and Recall. We need to measure the OSN account classification who have the malicious link by accepting a low precision if the testing result is not significant. Hence, we require an F1 score as an optimal blend of Precision and Recall. Tables 4 and 5 show the Recall, Precision, and F1 Score in both OSN datasets.

Table 3 | CNN with RunPool layer performance in link classification with different parameter setting.

| Input | Hyperparameter | Optimizer | Training Loss | Testing Loss |
|-------|----------------|-----------|---------------|--------------|
| FB link #50000 | Dropout \( p = 0.6 \) | Adam | 0.5033 | 0.5069 |
| | \( H1 = 64 \) | SGD, \( m = 0.5 \) | 0.5068 | 0.5070 |
| | \( H2 = 32 \) | RMSProp | 0.5176 | 0.5223 |
| | \( k_{top} = 5 \) | | | |
| Flickr #50000 | Dropout \( p = 0.6 \) | Adam | 0.5052 | 0.5060 |
| | \( H1 = 64 \) | SGD, \( m = 0.5 \) | 0.5178 | 0.5160 |
| | \( H2 = 32 \) | Adagrad | 0.5075 | 0.5074 |
| | \( k_{top} = 5 \) | RMSProp | 0.5092 | 0.5087 |

FB, Facebook.

Table 4 | Accuracy and F1 Score with several machine learning algorithms for the Facebook dataset.

| Classification Model | Accuracy | Average Precision and Recall | F1 Score |
|----------------------|----------|-----------------------------|----------|
| Bernoulli            | 78.77    | 0.2123                      | -        |
| Logistic regression  | 91.49    | 0.6803                      | 0.7563   |
| SVM (rs = 40)        | 91.95    | 0.6974                      | 0.7724   |
| SVM (rs = 50)        | 91.99    | 0.6990                      | 0.7745   |
| S-neural network     | 94.10    | 0.9120                      | 0.9210   |
| Proposed approach    | 98.04    | 0.9775                      | 0.9595   |

Table 5 | Accuracy and F1 Score with several machine learning algorithms for the Flickr dataset.

| Classification Model | Accuracy | Average Precision and Recall | F1 Score |
|----------------------|----------|-----------------------------|----------|
| Bernoulli            | 78.77    | 0.2123                      | -        |
| Logistic regression  | 92.86    | 0.7048                      | 0.7787   |
| SVM (rs = 40)        | 92.47    | 0.7217                      | 0.7897   |
| SVM (rs = 50)        | 92.52    | 0.7257                      | 0.7938   |
| S-neural network     | 94.10    | 0.9280                      | 0.9321   |
| Proposed approach    | 98.34    | 0.9732                      | 0.9846   |

Table 6 | Classification Model Accuracy Average Precision and Recall F1 Score

4.2. OSN Account Classification

In this section, we test the proposed RunPool with several learning parameters to classify the user categories based on OSN profile attributes. In the training process, this study was tested with some popular optimizers, including SGD with momentum, Adam, Adagrad, and RMSProp. The study calculates the gradient of loss concerning the weight with various learning rates.

In this experiment, we tune the gradient descent and optimization algorithms with different hyperparameters. In NN training, choosing the value of a hyperparameter is one of the most important parts to increase the network training quality. By tuning appropriate hyperparameters in the CNN, we obtain the highest performance of the neural network. In the training and testing process, we have put the local variable concept to compute the loss function and compute the gradient of the loss concerning the weight with various learning rates.

In this experiment, the accuracy result has shown that supervised machine learning produces diverse accuracy. There are Bernoulli Naïve Bayes (86.90%), Gradient Boosting (GB) (90.66%),
LR (90.49%), SVM (90.04%), NN (92.10%). The proposed CNN architecture with the RunPool pooling layer can produce 94.00% in several times of the testing process with the benchmark dataset. We have found that the proposed pooling layer can harvest the highest accuracy than other common machine learning in the benchmark OSN user-profiles dataset. At the final stage, we also calculate the ROC curve as a binary classification. The ROC curve shows that the proposed model can produce an AUC area of more than 0.9. It reflects the model that it can be a good classifier for training the dynamic dataset such as OSN.

### 4.3. Pooling Comparison

In this part, we provide the comparison result among RunPool Pooling, Max Pooling, and Average Pooling to calculate model accuracy. We calculate the accuracy of the model combined with different pooling functions to measure the performance of each pooling in the classification task. This study tested the accuracy among RunPool Pooling, Max Pooling, and Average Pooling in the same hyperparameter tuning and environment. It is to convince the performance of each pooling in the classification task. Table 8 displays a comparison result among different pooling function by using 2254 training samples and 545 validation samples.

In the first experiment, the study trains the classifier without a pooling layer that means CNN replaces the pooling layer by convolution with stride. However, CNN without pooling cannot produce a good result in diverse training periods. In this case, CNN requires pooling layers to reduce the dimensions of output features and render some amount of invariance. Based on the training result, CNN with pooling layers are better than without pooling.

To ensure the performance among RunPool Pooling, Max Pooling, and Average Pooling, we add the pooling layers into the CNN. As we know, Max Pooling and Average Pooling are deterministic pooling types that take the pooling value upon the maximum and average score of a pooling window. Instead of using ordinary deterministic pooling, we test the RunPool as a nondeterministic pooling to train the CNN model in the classification task. The experiment shows that the proposed CNN with RunPool Pooling can achieve a promising accuracy for an account classification task. However, this experiment suffers from decreasing accuracy when trained with a large epoch (many iterations). It can be solved by providing more dataset for training the model.

At the final stage, we evaluate the classifier by using ROC and AUC scores. This technique is used to assess the false alerts of malicious detection. Based on the experiment, the graph depicts a comparison among several pooling layers in calculating the AUC score and False Positive Rate (FPR) score. Figure 5 depicts the AUC comparison among three pooling functions in malicious classification by using OSN features as the dataset.

Figure 4 depicts that the AUC scores with the proposed approach (a) are higher than the conventional pooling layer (a) and (b). In this part, we plot AUC as a binary classification by adopting problem distributed observations (X1; Y1) and (X2; Y2). The AUC illustrates the percentage of the area under the ROC curve with range between 0–1. The proposed model can achieve a promising result with $AUC = 0.9889$. Thus, the AUC reflects that the classifier can discriminate between positive and negative classes (malicious and benign label) effectively.

![Figure 5](image_url)

**Figure 5** | Comparison result of the AUC score among Max Pooling, Average Pooling, and RunPool Pooling layer in the malicious classification task.

### Table 6 Classification result of the proposed CNN with RunPool layer in with different parameter setting.

| Parameter | Optimizer | Val-Loss | Val-Accuracy |
|-----------|-----------|----------|--------------|
| Dropout p = 0.5 | Adam | 0.3633 | 0.9131 |
| H1 = 32 | SGD, m = 0.9 | 0.2087 | 0.9291 |
| H2 = 64 | Adagrad | 0.3620 | 0.9025 |
| $k_{top}$ = 7 | RMSProp | 0.4049 | 0.8794 |

| Dropout p = 0.3 | Adam | 0.4018 | 0.8954 |
| H1 = 16 | SGD, m = 0.9 | 0.2143 | 0.9344 |
| H2 = 32 | Adagrad | 0.3559 | 0.8898 |
| $k_{top}$ = 5 | RMSProp | 0.4017 | 0.8865 |

### Table 7 Comparison result between some common classification algorithms and the proposed approach in the OSN dataset.

| Classification Model | Precision | Recall | F1 Score |
|----------------------|-----------|--------|----------|
| Bernoulli (Naïve Bayes) | 86.90 | 86.94 | 87.00 |
| Gradient Boosting (n estim=50) | 90.66 | 90.69 | 91.00 |
| Logistic Regression | 90.47 | 90.59 | 90.61 |
| SVM (rs = 31.6) | 90.04 | 87.34 | 87.24 |
| S-Neural Network | 92.10 | 92.80 | 92.91 |
| Proposed Approach | 94.00 | 93.21 | 93.42 |

### Table 8 Comparison of validation accuracy among pooling functions when training with the same hyperparameter tuning and dataset.

| Pooling Type | Epoch = 7 | Epoch = 10 | Epoch = 15 |
|--------------|-----------|------------|------------|
| Average Pooling | 91.13% | 91.49% | 87.94% |
| Max Pooling | 91.21% | 91.31% | 89.89% |
| Max. Pool + RunPool | 91.67% | 91.67% | 89.36% |
| Avg. Pool + RunPool | 91.40% | 91.89% | 88.48% |
| RunPool Pooling | 92.38% | 92.54% | 91.08% |
To sum up the benefit of the proposed pooling layer in the CNN architecture with the 1D dataset, we have presented the training and validation result by calculating the huge sample (OSN features). We find that the proposed CNN with RunPool can achieve an efficient result and increase the learning performance of the neural network. In DL study, the proposed pooling technique can be an alternative way to train a DL model not only for the 1D dataset (text) but also for a 2D dataset (image) and 3D dataset (video and voice).

5. CONCLUSION

In this paper, we introduce a new pooling layer called RunPool to find different optimal pooling values within the pooling window. We propose the pooling function to deal with deterministic pooling issues by harnessing some network parameters to obtain an informative feature map. It calculates some layer parameters and Gaussian distribution numbers based on $k_{isp}$ initialization. Then, the function compares $k_{isp}$ and $k_{top}$ to achieve the best pooling value in each hidden layer, respectively.

In the test process, we obtain a dynamic optimizer that is better to achieve high accuracy and small loss. On the other hand, the SGD with momentum also produces a competitive result. It can provide an accurate and minimum loss if we tune it with a larger epoch and big regulizer. SGD with momentum optimizer become an excellent option to train the CNN RunPool model. In fake OSN account classification, we compare the proposed CNN accuracy to the supervised ML, including Bernoulli Naïve Bayes (86.90%), GB (90.66%), LR (90.49%), and SVM (90.04%). The proposed CNN can achieve the highest score by producing 94.00%.

Demonstrated by experiment with 1D OSN dataset, it is confirmed that RunPool can improve the performance of the CNN to address the drawback of common max pooling and average pooling. This type of stochastic pooling can be combined with the common pooling layers to produce better accuracy with tiny loss. By implementing CNN with RunPool, we harvest an increased level in the graph performance and gain better accuracy in the validation process for the classification case. Not just calculate the accuracy and loss, we also conduct the evaluation metric to measure the performance of the classifier. By calculating the Precision, Recall, and F1 Score, we obtain a promising result because the proposed model can achieve the highest AUC score.

As the future directions of malicious classification study with DL, it needs an exploration for handling accounts and malware hierarchy links that reflect the source of the threat. Then, neural network computation is necessary to calculate with a new technique like adaptive loss function rather than a regular loss function. We also suggest the next study to test this model by utilizing a 2D dataset (image) and 3D (video or audio) dataset for measuring training and testing accuracy.

CONFLICT OF INTEREST

The authors declare they have no conflicts of interest.

AUTHORS’ CONTRIBUTIONS

All authors have contribution in this research.

ACKNOWLEDGMENTS

This paper is conducted in the Institute of Research in Information Processing Laboratory, Harbin University of Science and Technology under CSC Scholarship (No. 2016BSZ263). This project is available on Github.

REFERENCES

[1] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv/1409.1556, 2014.
[2] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in Advances in Neural Information Processing Systems (NIPS), Lake Tahoe, Nevada, 2012, pp. 1106–1114.
[3] C. Szegedy, S. Ioffe, V. Vanhoucke, A.A. Alemi, Inception-v4, inceptionresnet and the impact of residual connections on learning, in Association for the Advancement of Artificial Intelligence (AAAI), San Francisco, 2017, pp. 4278–4284.
[4] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, Cambridge, MIT Press, 2016.
[5] M.D. Zeiler, R. Fergus, Stochastic pooling for regularization of deep convolutional neural networks, arXiv/1301.3557, 2013.
[6] C. Vincent, L. Spranger, M. Seuret, A. Nicolau, P. Král, A. Maier, Deep generalized max pooling, arXiv: 1908.05040, 2019.
[7] E. Hayoung, H. Choi, Alpha-pooling for convolutional neural networks, arXiv: 1811.03436v1 [cs.LG], 2018.
[8] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature. 521 (2015), 436–444.
[9] L. Marcelo, T. Issa, W. Isaac, S. Mohammad, Authorship verification using deep belief network system, Int. J. Commun. Syst. 30 (2017), e3259.
[10] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: closing the gap to human-level performance in face verification, in Computer Vision and Pattern Recognition (CVPR), IEEE Conference on Pattern Recognition, Columbus, 2014, pp. 1701–1708.
[11] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE. 86 (1998), 2278–2324.
[12] D. Yu, H. Wang, P. Chen, Z. Wei, Mixed pooling for convolutional neural networks, in International Conference on Rough Sets and Knowledge Technology, Shanghai, China, 2014.
[13] N. Kalchbrenner, E. Grefenstette, P. Blunsom, A convolutional neural network for modelling sentences, in Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Baltimore, 2014.
[14] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D.U. Hwang, Complex networks: structure and dynamics, Phy. Rep. 424 (2006), 175–308.
[15] P. Wanda, H.J. Jie, URLDeep: continuous prediction of malicious URL with dynamic deep learning in social networks, Int. J. Netw. Secur. 21 (2019), 971–978.
[16] P. Wang, B. Xu, Y. Wu, X. Zhou, Link prediction in social networks: the state-of-the-art, Sci. China Inf. Sci. 58 (2015), 1–38.
[17] W. Putra, S. Selo, B.S. Hantono, Model of secure P2P mobile instant messaging based on virtual network, in 2014 International Conference on Information Technology Systems and Innovation (ICITSI), Bandung, 2014, pp. 81–85.
[18] P. Wanda, S. Selo, B.S. Hantono, Efficient message security based HyperElliptic Curve Cryptosystem (HECC) for mobile instant messenger, in 2014 The 1st International Conference on Information Technology, Computer, and Electrical Engineering, Semarang, 2014, pp. 245–249.

[19] M.G. Vigliotti, C. Hankin, Discovery of anomalous behaviour in temporal networks, Soc. Netw. 41 (2015), 18–25.

[20] T. Li, B. Wang, Y. Jiang, Y. Zhang, Y. Yan, Restricted Boltzmann machine-based approaches for link prediction in dynamic networks, IEEE Access. 6 (2018), 29940–29951.

[21] T. Li, J. Zhang, P.S. Yu, Y. Zhang, Y. Yan, Deep dynamic network embedding for link prediction, IEEE Access. 6 (2018), 29219–29230.

[22] P. Bindu, S. Thilagam, Mining social networks for anomalies: methods and challenges, J. Netw. Comput. Appl. 68 (2016), 213–229.

[23] Q. Gong, Y. Chen, X. He, Z. Zhuang, T. Wang, H. Huang, X. Wang, X. Fu, DeepScan: exploiting deep learning for malicious account detection in location-based social networks, IEEE Commun. Mag. 56 (2018), 21–27.

[24] M. Mohammadrezaei, M.E. Shiri, A.M. Rahmani, Identifying fake accounts on social networks based on graph analysis and classification algorithms, Hindawi Secur. Commun. Netw. 2018 (2018), 1–8.

[25] D. Savage, X. Zhang, X. Yu, P. Chou, Q. Wang, Anomaly detection in online social networks, Social Netw. 39 (2014), 62–70.

[26] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002), 321–357.

[27] J.A. Swets, Signal Detection Theory and ROC Analysis in Psychology and Diagnostics: Collected Papers, Lawrence Erlbaum Associates, Mahwah, 1996.