2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)

$SinP[N]$: A Fast Convergence Activation Function for Convolutional Neural Networks

1st Ka-Hou Chan
Chinese-Portuguese-English Machine Translation Laboratory, Macao Polytechnic Institute,
Macao, China
chankahou@ipm.edu.mo

2nd Sio-Kei Im
Chinese-Portuguese-English Machine Translation Laboratory, Macao Polytechnic Institute,
Macao, China
marcusim@ipm.edu.mo

3rd Wei Ke
Macao Polytechnic Institute,
Macao, China
wke@ipm.edu.mo

4th Ngan-Lin Lei
Macao Polytechnic Institute,
Macao, China
vivianlei@ipm.edu.mo

***Abstract*—Convolutional Neural Networks (CNNs) are currently the most advanced machine learning architecture for visual data classification. The choice of activation functions has a significant impact on the performance of a training task. In order to overcome the vanishing gradient problem, we propose a new activation function for the classification system. The activation function makes use of the properties of periodic functions, where the derivative of a periodic function is also periodic. Furthermore, a linear combination is introduced to prevent the derivative from becoming zero. We verify this novel activation function by training an empirical analysis and comparing with the currently discovered activation functions. Experimental results show that our activation function $SinP[N](x) = \sin(x) + Nx$, leads to very fast convergence even *without* the normalization layer. As a result, this new activation function enhances the training accuracy significantly, and can be easily deployed in the current systems built upon the standard CNN architecture.**

***Index Terms*—Convolutional Neural Network, Fast Convergence, Periodic Function, Machine Learning**

I. Introduction

With the development of deep learning, neural networks have attracted enthusiastic interests in computer vision and pattern recognition. Convolutional Neural Networks (CNNs) have recently achieved great successes and are being widely applied to image classification [15], [21], [25]. A CNN is a sample processing system with the ability to filter out feature patterns from a set of visual data. However, these high-performance CNNs usually come with a large computational cost due to the use of multi-layer architectures for filtering, such as convolutional layers and pooling layers. This often requires accelerations of parallel GPUs, or highly optimized distributed CPU architectures to process large datasets.

In a CNN system, there are a series of filters applied to the visual image data to obtain and learn the identified features, which can then be used by the model for the classification purpose. Commonly, a CNN contains three major layers with a set of predefined activation functions.

Convolutional Layer This layer applies feature weights to convolution filters for training and obtain the features in the image. For each convolution region, the layer computes the sum of products as a single value in the output feature.

Pooling Layer This layer is to obtain the target sample of sub regions in order to increase the iterator performance. The commonly used pooling algorithm is max pooling, which only keeps the maximum value of the specified pixel tiles and discards all other values.

Fully Connected Layer This is a medium layer that sums up the features from the preceding layers, and then computes the specified number of nodes for each target class in the model (any possible class the model can predict).

In general, a CNN consists of a queue of modules to obtain features. Each module consists of one convolutional layer followed by one pooling layer. Finally, the last module is a connected layer to perform the classification. The output of a specified activation function *Softmax* [18] can be used to represent a probability vector $(p_0, p_1, \ldots, p_n)$, where each component p_i is in range $[0.0, 1.0]$, and the sum of all these components is equal to 1, i.e., $\sum_{i=0}^{n} p_i = 1.0$. Therefore, we can interpret the *Softmax* vector to determine how likely the given data item belongs to a target class, typically we select the index of the maximum probability component in the vector. Although most research in machine learning, including CNNs, usually focuses on the model architectures [8], [23], [24], it is also worth mentioning that the activation function always has its influence throughout the whole training process. Thus, the choice of the activation function has a significant impact on the training and testing accuracy of CNNs.

In this paper, we discover a novel class of activation functions based on sinusoid curves. We focus on solving the vanishing gradient problem, with the purpose to provide the momentum for the convergence everywhere ($x \in \mathbb{R}$) in SGD. We make use of the properties of periodic functions, i.e., $p(x) = p(x + T)$, in particular, the derivative of a periodic function is also periodic, i.e., $p'(x) = p'(x + T)$. A linear combination $p(x) + Nx$ is used to prevent the derivative from becoming zero. We conclude the acceptance range of N that is able to produce stable and satisfying results. We evaluate several periodic functions to further validate the performance by comparing with currently discovered activation functions. The best activation function that we have discovered, called $SinP[N]$, is defined as $SinP[N](x) = \sin(x) + Nx$, where N

978-1-7281-0359-4/18/$31.00 ©2018 IEEE
DOI 10.1109/UCC-Companion.2018.00082

is a kernel parameter to be discussed in the upcoming sections. Experiments show that $SinP[N]$ gains faster convergence on CNNs when applied to image classification. Given the same time period of iteration training, the accuracy improvement is significant compared with mainstream activate functions.

a) Related Work: In order to increase the convergence speed of CNNs, early solutions include constraining the network complexity by using fewer weights or sharing weights [15], [17]. To avoid the vanishing gradient problem is a great challenge to the training of effective and robust deep CNNs. Many learning algorithms have been extended to deal with different kinds of activation functions, including linear units [2], [13], kernel weights [5], [20] and layer analysis [3], [11]. Most notably, almost all activation functions involved in the training of complex models require nonintervention, thus, standard complex derivatives is hard to be used in runtime optimization algorithms. In recent years, typical CNNs prefer to use the Rectified Linear Unit (*ReLU*) [9] activation function in the convolutional and connected layer to employ nonlinearities into the training process. Later, some promising results are reported in [19] with the *Swish* function, which is similar to the *ReLU* function, but with the advantage of being smooth as well as having a non-zero left tail derivative. These two activation functions achieve the same goal through the sparse representation property. The sparse features are more likely to be linearly separable and have a smaller dependency on non-linear mapping mechanisms. Furthermore, there are also many regularization methods for network structure design that have been developed to avoid vanishing gradient. Batch normalization [12] can enable networks with raise sensitive layer to start convergence for Stochastic Gradient Descent (SGD) with back propagation. This effectively decouples each layer's parameters from those of other layers, leading to significantly faster convergence. It also generates better-conditioned optimization solutions [4], [16]. For initial weight optimization, a technique is presented in [7] that uses the Fast Fourier Transform (FFT) to approximate DFT, then uses the obtained value to initialize the sinusoid units of a neural network. This enables the weight to shift toward with time-series data, and demonstrates that it is both practical and effective. Later, an approach is shown in [10] that is even more effective at modeling when combined with other activation functions.

b) Outline: The rest of the paper is organized as follows. Section II formally defines the class of $SinP[N]$ functions and their derivatives. A couple of instances are illustrated to give an analysis of the properties of the $SinP[N]$ functions and derivatives, including the choice of the N kernel parameter and how it overcomes the vanishing gradient problem. Next, in Section III, we give the idea of how the new activation functions can be integrated, followed by the configurations of our experiment and evaluation environment. The results and improvements are illustrated and discussed in detail in this section as well. Finally, Section IV concludes the paper.

II. Properties of $SinP[N]$

As we know that neural networks do not use any linear equation as activation function. Because a neural network with a linear activation function effectively produces only the linear results, no matter how complex the architecture is. The input to a network is usually taken by a linear transformation as the neural operation,

$$f\left(weight \times input + bias\right).$$

However, problems in the real world are mostly formulated in non-linear forms. Thus, we employ the sinusoid as our non-linear activation function, i.e., $\sin\left(x\right)$. A major reason for this choice of sinusoid is that the curve is defined everywhere along the X-axis, this property can better coincide with more patterns through fewer iteration steps. Particularly for the *Softmax* function, the wave peak corresponds to the result with the the highest probability. Another reason is that sinusoid, as a periodic function, can prevent the gradient from becoming unstable when the input is very large. Similarly, the $\cos\left(x\right)$ function possesses all the properties of the $\sin\left(x\right)$ function, so it can also be considered as an activation function. However, by the nature that the cosine curve does not go through the origin, i.e., nonzero-centered. we are not going to discuss it further in this work. Nonetheless, cosine may still become a new activation function potentially.

A. First Derivative of $SinP[N]$

The kind of *ReLU*, *Swish* and *softplus* [6] is one-sided, and the gradient on the right side can keep existing (always equal or close to 1), but rapidly vanish on the left side (always equal or close to 0). In contrast, by using $\sin(x)$ as our activation function, the gradient $\cos(x)$ will cause the deep training process to become unstable. Because the gradient alternates between positive and negative, and the vanishing gradient problem also occurs intermittently. In order to overcome these problems and keep the advantages of *ReLU*-like activation functions, we introduce a linear combination with sinusoid as follows,

$$SinP[N](x) = \sin(x) + Nx, \tag{1}$$

thus we have the first derivative of $SinP[N]$ below,

$$\frac{d}{dx}SinP[N](x) = \cos(x) + N. \tag{2}$$

Here, $N \in \mathbb{R}$ is the kernel parameter. We can effectively alleviate the vanishing gradient problem by adjusting N. When $|N| > 1$, the derivative in (2) becomes either constantly positive or constantly negative, making $SinP[N]$ either monotonically increasing ($N > 1$) or monotonically decreasing ($N < -1$). It means that the neuron can always be active when $|N| > 1$. Note that our discussion only focuses on the $N > 1$ case, the other case ($N < -1$) can be handled similarly. Fig. 1 and 2 show the curves of three $SinP[N]$ functions and their corresponding derivatives when N taking $0, 1$ and 2.

As shown in Fig. 1, these wave curves are growing along the lines $y = 0$, $y = x$ and $y = 2x$ respectively, while keeping

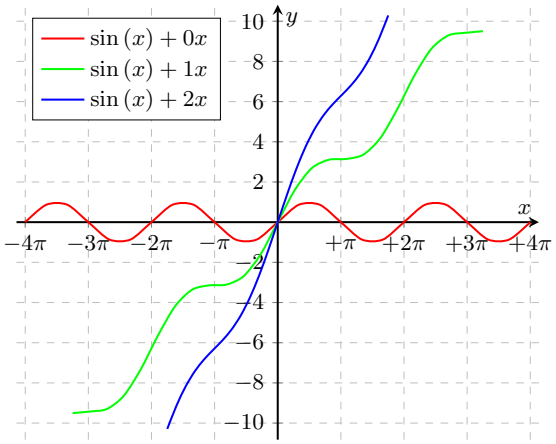

Fig. 1. The $SinP[N]$ activation function with $N \in \{0, 1, 2\}$.

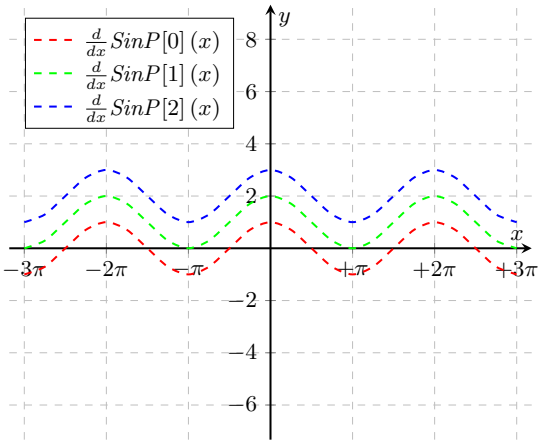

Fig. 2. First derivatives of $SinP[N]$ with $N \in \{0, 1, 2\}$.

nonlinear. Also as shown in Fig. 2, the corresponding first derivatives are along the lines $y' = 0$, $y' = 1$ and $y' = 2$ respectively. More generally, it can be seen that for $N \in \mathbb{R}$, activation function $SinP[N]$ is always along the line $y = Nx$, and its first derivative is always along the line $y' = N$.

B. Overcoming Vanishing Gradient

Compared with *ReLU*-like activation functions, the most striking property of $SinP[N]$ is that the first derivative is also a periodic function. The vanishing gradient problem can be effectively alleviated by adjusting the N kernel parameter. We can expect that having any $N > 1$ is effective in theory, and the experiments show that $SinP[N]$ practically improves the training performance on CNNs. In the training process, the N values are chosen between 1.0 and 2.0 and become divergent above 2.0.

III. Implementation of $SinP[N]$ in CNNs

Succinctly, $SinP[N]$ can be implemented with slight modification in most deep learning libraries. Comparing to some optimization methods such as the BatchNorm [12] and the Dropout [22], our training architecture can be constructed with much simpler layers. We employ the MNIST [17] and CIFAR-10 [14] as our image datasets for training. There are more than 50,000 images for training and testing. We use the raw data directly without any pre-processing. The result is from the original view test without any enhancement. All our experiments were conducted on an Nvidia Quadro M2000 with 4GB of video memory.

A. Overall Architecture

According to the above discussion, we use a plain architecture for training process as shown in Table I and II. These are two sets of convolutional and pooling layers, with the activation functions following the corresponding layers. In both experiments, we evaluate classification performance on the same CNNs with different activation functions — $SinP[N]$, *ReLU* and *Swish*. Note that we use the same activation function from start to finish in each experiment. In order to show the gradient activity, we use the most gradient related optimizer, the SGD method, for the approximation.

We implement the model in Tensorflow [1] and switch the activation function between $SinP[N]$, *ReLU* and *Swish*. For more detail about $SinP[N]$, we discuss it in three cases: $SinP[1]$, $SinP[1.5]$ and $SinP[2]$. We use the same data augmentation with 100 batch size per iteration step. The initial learning rate is set to 0.01 for MNIST and 0.003 for CIFAR-10 without momentum. For more detail, the complete python source code can be found at github.com/ChanKaHou/SinPN.

B. Result and Discussion

We first analyze the convergence speed of each individual activation function. Our experiments processed 100,000 iteration steps for each activation function. We conducted our experiments on both CNN models performance and a median of 4 turns. It is notable that the performance results of *ReLU* and *Swish* are almost the same, with a very slight gain by *Swish*, thus we only show the result of *Swish* in the following figures. We can see a clear difference between $SinP[N]$, *ReLU* and *Swish*.

As shown in Fig. 3, the advantages of these $SinP[N]$s are not very obvious, nonetheless it can be seen that $SinP[N]$ can provide a faster learning process than *Swish*, which shows the cross entropy required to reach consumes only 50% to 30% of the iteration steps with $SinP[N]$ on the MNIST dataset for the CNNs in Table I. In contrast, the CIFAR-10 dataset is more complex that the advantages of $SinP[N]$ can be obviously visualized in Fig. 4. As expected, the convergence will speed up with the increasing N ($N > 1$). The numbers listed in Fig. 4 shows the cross entropy required to reach consumes only 20% to 10% iteration steps with $SinP[N]$ on the CIFAR-10 dataset for the CNNs in Table II. These are significant improvements.

Furthermore, we compare the accuracies of different activation functions. The model configuration is the same as the previous implementation and we find the test accuracy (%)

TABLE I
MNIST CNNs STRUCTURE.

Input Size	Layer:	$\{parameter = value\}$	Activation Function
$28 \times 28 \times 01$	convolution:	$\{kernel = 7 \times 7, filters = 8\}$	$SinP[N]$, *ReLU*, *Swish*
$28 \times 28 \times 08$	pooling:	$\{kernel = 2 \times 2, strides = 2 \times 2\}$	none
$14 \times 14 \times 08$	convolution:	$\{kernel = 7 \times 7, filters = 16\}$	$SinP[N]$, *ReLU*, *Swish*
$14 \times 14 \times 16$	pooling:	$\{kernel = 2 \times 2, strides = 2 \times 2\}$	none
$07 \times 07 \times 16$	connected:	$\{units = 10\}$	$SinP[N]$, *ReLU*, *Swish*
10	softmax:	$\{\}$	*Softmax*

TABLE II
CIFAR-10 CNNs STRUCTURE.

Input Size	Layer:	$\{parameter = value\}$	Activation Function
$32 \times 32 \times 03$	convolution:	$\{kernel = 8 \times 8, filters = 24\}$	$SinP[N]$, *ReLU*, *Swish*
$32 \times 32 \times 24$	pooling:	$\{kernel = 2 \times 2, strides = 2 \times 2\}$	none
$16 \times 16 \times 24$	convolution:	$\{kernel = 8 \times 8, filters = 48\}$	$SinP[N]$, *ReLU*, *Swish*
$16 \times 16 \times 48$	pooling:	$\{kernel = 2 \times 2, strides = 2 \times 2\}$	none
$08 \times 08 \times 48$	connected:	$\{units = 10\}$	$SinP[N]$, *ReLU*, *Swish*
10	softmax:	$\{\}$	*Softmax*

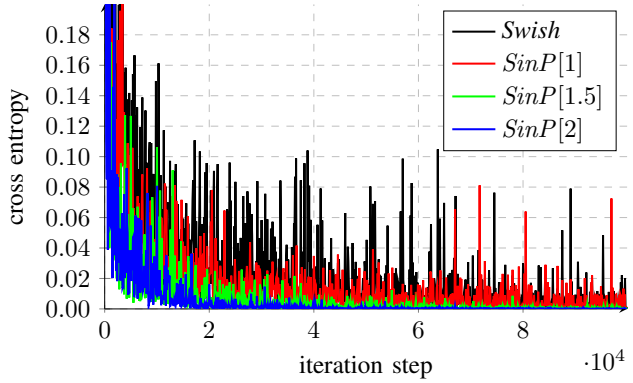

Fig. 3. MNIST, the trend of cross entropy with 100-thousand iteration steps.

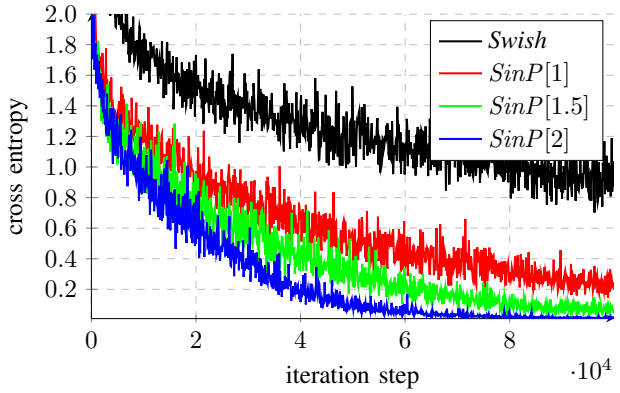

Fig. 4. CIFAR-10, the trend of cross entropy with 100-thousand iteration steps.

after completing of 200 epochs. For the MNIST dataset, all test accuracies can reach 98.8% at the first epoch within one thousand iterations. The more variant results on the CIFAR-10 dataset are shown in Fig. 5.

In Fig. 5, we can see that $SinP[N]$ outperforms both *ReLU* and *Swish*, significantly. The improvement delta provided by $SinP[N]$ is 0.3 relative to *ReLU* and 0.2 relative to *Swish*. It is very exciting that the strong performance of $SinP[N]$ poses a great challenge to the conventional method with *ReLU*-like activation functions. In all fairness, since our experiment models do not use any optimization method, e.g., BatchNorm, the results of the discovered activation function should be worse than other works, but our proposed $SinP[N]$ consistently still outperform others on CNNs without the optimization layers. However, the initial learning rate is difficult to define when $N > 1.5$. Because N has a major influence on the gradient, and the practice in our experiments shows that the occurrences of divergent will increase with an increasing N, therefore, we do not think it appropriate to set $N > 2.0$, otherwise the training process will become unstable. Practically, setting $N = 1.5$ is more suitable and highly recommended.

IV. Conclusion

We present a novel activation function suitable for CNNs to approximate the *Softmax* results for the purpose of speeding up convergence of SGD. Our method maintains the nonlinear property through the use of sinusoid. By introducing the linear combination $\sin(x)+Nx$, we are able to prevent the derivation from becoming zero. We show that the choice of kernel parameter N can influence the convergence speed. To validate the effectiveness of the new activation function, we use models with simple CNN architectures and compare the training processes with different activation functions. Our results show that $SinP[N]$ can outperform both *ReLU* and *Swish* from start to finish in convergence speed. This improvement over the discovered activation functions has been consistent and significant throughout our experiments, although we haven't

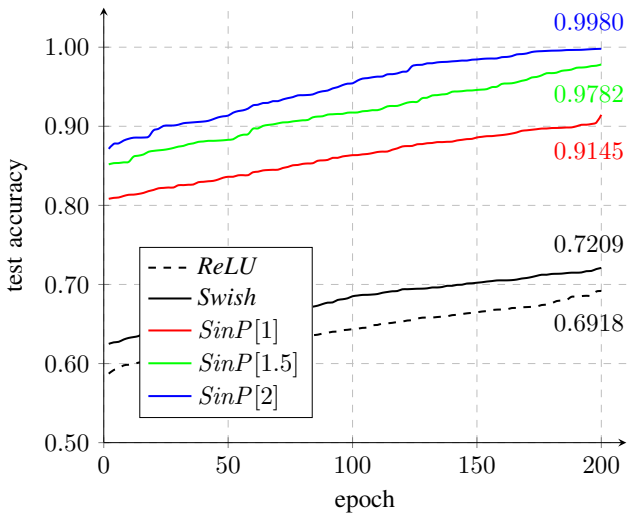

Fig. 5. Test accuracy rates for CIFAR-10 after 200 epochs of training.

integrated the state-of-art optimization layer. Further work can be done to apply this new activation function to other neural networks and with different datasets.

ACKNOWLEDGEMENT

This work was supported by the Macao Science and Technology Development Fund through Project 138/2016/A3.

REFERENCES

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)
[2] Bouboulis, P., Theodoridis, S.: Extension of wirtinger's calculus to reproducing kernel hilbert spaces and the complex kernel lms. IEEE Transactions on Signal Processing 59(3), 964–978 (2011)
[3] Chua, L.O., Roska, T.: The cnn paradigm. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 40(3), 147–156 (1993)
[4] Cireşan, D., Meier, U., Masci, J., Schmidhuber, J.: A committee of neural networks for traffic sign classification. In: The 2011 International Joint Conference on Neural Networks (IJCNN). pp. 1918–1921. IEEE (2011)
[5] Ciresan, D.C., Meier, U., Masci, J., Maria Gambardella, L., Schmidhuber, J.: Flexible, high performance convolutional neural networks for image classification. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence. vol. 22, p. 1237. Barcelona, Spain (2011)
[6] Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., Garcia, R.: Incorporating second-order functional knowledge for better option pricing. In: Advances in neural information processing systems. pp. 472–478 (2001)
[7] Gashler, M.S., Ashmore, S.C.: Training deep fourier neural networks to fit time-series data. In: International Conference on Intelligent Computing. pp. 48–55. Springer (2014)
[8] Girshick, R.: Fast r-cnn. arXiv preprint arXiv:1504.08083 (2015)
[9] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. pp. 315–323 (2011)
[10] Godfrey, L.B., Gashler, M.S.: A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks. In: 2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K). vol. 1, pp. 481–486. IEEE (2015)
[11] Huang, J.T., Li, J., Gong, Y.: An analysis of convolutional neural networks for speech recognition. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 4989–4993. IEEE (2015)
[12] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
[13] Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., Yan, S.: Deep learning with s-shaped rectified linear activation units. In: AAAI. pp. 1737–1743 (2016)
[14] Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
[15] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
[16] Le, Q.V.: Building high-level features using large scale unsupervised learning. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 8595–8598. IEEE (2013)
[17] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
[18] Nasrabadi, N.M.: Pattern recognition and machine learning. Journal of electronic imaging 16(4), 049901 (2007)
[19] Ramachandran, P., Zoph, B., Le, Q.V.: Swish: a self-gated activation function. arXiv preprint arXiv:1710.05941 (2017)
[20] Sankaradas, M., Jakkula, V., Cadambi, S., Chakradhar, S., Durdanovic, I., Cosatto, E., Graf, H.P.: A massively parallel coprocessor for convolutional neural networks. In: Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on. pp. 53–60. IEEE (2009)
[21] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
[22] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)
[23] Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., Xu, W.: Cnn-rnn: A unified framework for multi-label image classification. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2285–2294. IEEE (2016)
[24] Wang, L., Cao, Z., de Melo, G., Liu, Z.: Relation classification via multi-level attention cnns. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). vol. 1, pp. 1298–1307 (2016)
[25] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision. pp. 818–833. Springer (2014)