


A Generalized Attention Mechanism to Enhance the Accuracy Performance of Neural Networks

Pengcheng Jiang 

School of Software

Nanjing University of Information Science and Technology

Nanjing 210044, P. R. China

pcjiang@nuist.edu.cn


Ferrante Neri 

NICE Research Group

School of Computer Science and Electronic Engineering

University of Surrey, Guildford GU2 7XS, UK

f.neri@surrey.ac.uk

Yu Xue 

School of Software

Nanjing University of Information Science and Technology

Nanjing 210044, P. R. China

xueyu@nuist.edu.cn

Ujjwal Maulik 

Department of Computer Science and Engineering

Jadavpur University, Kolkata, India

ujjwal.maulik@jadavpuruniversity.in

Received 6 June 2024

Accepted 8 August 2024

Published Online 31 August 2024

In many modern machine learning (ML) models, attention mechanisms (AMs) play a crucial role in processing data and identifying significant parts of the inputs, whether these are text or images. This selective focus enables subsequent stages of the model to achieve improved classification performance. Traditionally, AMs are applied as a preprocessing substructure before a neural network, such as in encoder/decoder architectures. In this paper, we extend the application of AMs to intermediate stages of data propagation within ML models. Specifically, we propose a generalized attention mechanism (GAM), which can be integrated before each layer of a neural network for classification tasks. The proposed GAM allows for at each layer/step of the ML architecture identification of the most relevant sections of the intermediate results. Our experimental results demonstrate that incorporating the proposed GAM into various ML models consistently enhances the accuracy of these models. This improvement is achieved with only a marginal increase in the number of parameters, which does not significantly affect the training time.

Keywords: Convolutional neural networks; deep learning; deep neural networks.

*Corresponding author.

1. Introduction

Deep neural networks are a popular model for most prediction tasks, such as image classification,^{1,2} text classification,³ pattern recognition,^{4,5} feature analysis,⁶ and segmentation,^{7,8} etc. Because of its powerful capabilities in these fields, deep neural networks are often used in various industrial scenarios, including medical diagnosis,^{9–12} defect detection,^{13,14} recommendation systems,¹⁵ and more.^{16,17} In supervised learning, neural networks are trained using datasets, which allows them to learn and make predictions or decisions without being explicitly programmed. Neural networks need to be trained to achieve good performance. The back propagation (BP) algorithm^{18–20} is one of the most broadly used approaches for training deep neural networks. BP algorithm is a parameter search method based on gradient, which updates the parameters of the model by the back-propagated error signal between the real values and the predicted values. At present, there are many successful implementations of optimizers based on BP algorithm²¹ such as stochastic gradient descent (SGD)²² and adaptive moment estimation (Adam).²³

In the process of optimizing neural network parameters using gradient descent, the parameters corresponding to different features are updated simultaneously. However, some of these updates may actually harm the performance of the neural network. Recent research has even shown that removing certain neurons can lead to improved network performance. Furthermore, the success of the Dropout method highlights the challenge of achieving optimal results by treating all features equally during training.²⁴ Therefore, it's essential to train and utilize features selectively within neural networks. To address this, attention mechanisms (AMs) are employed to identify the most crucial sections of input data^{25–27} and process them accordingly in subsequent stages. For instance, AMs are used in natural language processing (NLP) to identify the most relevant words in a sentence,²⁸ and in image processing to identify the most important elements and isolate them from the background.²⁹

In its traditional implementation, the AM is an additional layer of a neural network typically positioned before the convolutional blocks to preprocess the input of each stage, leaving the task of

classification to the neural network using the pre-processed data. While this approach is beneficial, there is no reason why attention cannot be applied again before subsequent layers of the neural network. Intuitively, including an AM in the following layers may improve the accuracy of the model. For example, in image recognition, an AM applied to the feature maps could bolster the learning of the subsequent layer.

Although conceptually an AM could be placed before any layer, from an implementation standpoint, it cannot be directly located before just any layer. It should be adapted to function with the layer that follows it. Inspired by this idea, we propose in this paper a generalized attention mechanism (GAM) that can be positioned before any layer of any neural network to enhance the performance of the neural network. The proposed mechanism identifies at each step the most important sections that are inputted into the following layer.

The rest of this paper is organized as follows. Related works on neural networks and AMs are provided in Sec. 2. The implementation and theory of GAM are described in Sec. 3. Section 4 presents experiments and analysis. Finally, Sec. 5 concludes the paper.

2. Related Work

Machine learning (ML) has revolutionized numerous fields by providing algorithms and statistical models that enable computers to perform tasks without explicit instructions, relying instead on patterns and inference.^{30,31} Building on the foundations of ML, deep learning (DL) has emerged as a subset that uses neural networks with many layers to model complex patterns in large datasets.^{32,33} Many important research works in DL have proposed different neural network structures.³⁴ Among the various architectures in DL, convolutional neural networks (CNNs) have been particularly successful in tasks involving image and spatial data due to their ability to capture hierarchical patterns through convolutional layers. More recently, AMs have significantly advanced the field by allowing models to focus on relevant parts of the input data dynamically, improving performance in tasks like NLP and image captioning by providing context-aware processing

capabilities. In this section, the related works about AMs will be introduced.

2.1. Attention mechanism: Definition and notation

The AM is an element of a machine learning model that focuses on specific parts of the input sequence when generating each part of the output sequence. It dynamically computes a weighted sum of the input elements, where the weights are determined by a “compatibility” function that measures the relevance of the input elements to the current output element.

With the purpose of introducing the notation and laying the basis for the description of our method, we briefly outline the functioning of AM. Given an input sequence represented by a matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$, where each $\mathbf{x}_i \in \mathbb{R}^d$ is an input vector (whose elements are referred to as features) and a query vector $\mathbf{q} \in \mathbb{R}^d$, an AM performs the following three operations.

1. Score Calculation

Compute the scores that represent the similarity between the query vector \mathbf{q} and each input vector \mathbf{x}_i . This can be done using various scoring functions, such as dot product, additive, or scaled dot-product. For simplicity, let’s use the dot product:

$$e_i = \mathbf{q}^\top \mathbf{x}_i \quad \text{for } i = 1, 2, \dots, n. \quad (1)$$

Here, e_i is the unnormalized score for the i th input.

2. Softmax

Normalize the scores using the softmax function to obtain the attention weights α_i :

$$\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^n \exp(e_j)} \quad \text{for } i = 1, 2, \dots, n. \quad (2)$$

The attention weights α_i sum to 1 and indicate the relative importance of each input vector.

3. Context Vector

Compute the context vector \mathbf{c} as a weighted sum of the input vectors, using the attention weights α_i :

$$\mathbf{c} = \sum_{i=1}^n \alpha_i \mathbf{x}_i. \quad (3)$$

This mechanism allows the model to focus on the most relevant parts of the input sequence, effectively attending to different parts of the input when generating each part of the output sequence.

2.2. Attention modules in natural language processing

AMs have revolutionized NLP by allowing models to focus on the most relevant parts of a sentence or document when processing information. An early work was the RNNsearch approach proposed by Bahdanau *et al.* which uses an AM to simultaneously translate and align on a machine translation task.³⁵ Luong *et al.* introduced the use of global and local attention to enable a finer alignment between source and target sentences in machine translation tasks.³⁶ Subsequently, the proposal of the self-AM led to the widespread use of Transformer based on this technique in NLP. It allows the model to focus on different positions in a single sequence to capture dependencies regardless of distance. This feature has made Transformer more successful on several NLP tasks.³⁷ BERT is one of the most successful studies of Transformer, which uses masked language model targets pre-trained in large corpora and migrated to specific scenarios.³⁸ Another well-known application is GPT, which generates coherent and rich text based on input, and its latest version has achieved optimal results on a large number of NLP tasks.³⁹

2.3. Attention modules in computer vision

As AMs have garnered considerable interest across disciplines, researchers have endeavored to apply this concept within the realm of computer vision.⁴⁰ In recent years, the main classifications of visual AMs include: channel attention, spatial attention, self-attention, etc. SENet is the first neural network that uses channel attention.⁴¹ They proposed Squeeze-and-Excitation (SE) Block to add an AM module to the positions that need to be necessary, in which they modified the method in the traditional AM to compute a query-like vector using the average value at the channel level. The formulaic expression of the SE Block is $f(\mathbf{X}) = \text{Sigmoid}(\text{MLP}(\text{GAP}(\mathbf{X})))$, where *MLP* is a small network with two layers of fully connected and ReLU activation functions, *GAP* is global average pooling, and \mathbf{X} is an input tensor

containing multiple channels. CBAM adds spatial attention to this, thus focusing on important information at the same channel level.⁴² The formulaic expression for spatial attention can be denoted as $f(\mathbf{X}) = \text{Sigmoid}(\text{conv}([\text{MP}(\mathbf{X}); \text{AP}(\mathbf{X})]))$, where conv is a convolutional kernel of size 7, MP denotes the maximum pooling and AP denotes the average pooling operation. In the following studies, more research has since been conducted based on these two types of AMs.^{43,44} With the successful application of the self-AM in the image area, Transformer networks are heavily used for visual tasks such as image classification.^{45–47} This approach constructs the query, key, and value matrices of the self-AM after patch embedding the image to be categorized. Then, the self-AM within following layers computes attention scores based on the similarity between all pairs of patches in the query, key, and value representations. Therefore, the network dynamically focuses on the most relevant parts of the image for the task at hand, such as distinguishing between categories in image classification tasks. These attention scores are then used to create a weighted combination of value vectors, effectively allowing the model to emphasize certain features over others. The outputs of the self-attention layers are then passed through feed-forward neural networks — another component of the Transformer layer. These networks apply further transformations to the data, adding nonlinearity and helping to refine the representation for the final classification layer.

Among the plethora of studies presenting machine learning and specifically neural systems endowed with AMs for computer vision, it is worth mentioning the medical imaging domain. Some examples include image classification,⁴⁸ pattern enhancement in neuroimaging,⁴⁹ EEG for epilepsy detection,⁵⁰ prediction,⁵¹ and classification.⁵² It also worth mentioning the use of the AM in image generation and translation.⁵³

3. Generalized Attention Mechanism

The proposed generalized attention mechanism (GAM) is a module that modifies the functioning outlined in Sec. 2 to be applied before any layer of a neural network. In this module, we use a trainable parameter to represent the importance of features

and make the model pay more attention to important features during the training process. The GAM can be used in both linear and convolutional layers and consists of two parts: a scaling vector \mathbf{s} consisting of trainable parameters and a tuning function $t(\cdot)$. The parameters will be updated during the BP process. For the linear layer, we consider neurons as features, while in the convolutional layer, we take the feature maps on each channel as features. The length of the scaling vector depends on the number of features in this layer. The tuning function in GAM is an activation function and is used to add nonlinearity to the AM. The value obtained after the scaling vector is calculated by the tuning function reflects the importance of the features. Therefore, the *score calculation* in GAM is given by the following equation:

$$e_i = t(\mathbf{u}), \quad u \sim \mathcal{N}(0, 1), \quad (4)$$

where e_i is the initial importance score, and it is calculated by a random vector \mathbf{u} . In order to prevent the effect of the scale of the value domain of $t(\cdot)$, we need to scale this importance representation to satisfy that the sum of the importance of all features is equal to the number of features. Thus, the attention coefficients can be represented through *Softmax* operations as

$$\alpha_i = N \times \frac{\exp(e_i)}{\sum_{j=1}^n \exp(e_j)} \quad \text{for } i = 1, 2, \dots, n, \quad (5)$$

where t is Sigmoid as we used in our method, N is the number of features in the current layer. The general structure of the module can be abstracted as Fig. 1. For the linear layer, suppose we are given an input vector \mathbf{x} , and the output \mathbf{o} can be represented as

$$o_i = \alpha_i \times x_i, \quad (6)$$

where x_i are the values of input neurons, o_i are the elements of a *context vector* to be used for subsequent layers to compute. Then, the following linear layer can calculate the output vector \mathbf{y} with

$$y_j = \sum_{i=1}^{\text{Num}_{in}} FC_{i,j} \times o_i, \quad (7)$$

where y_j is each element in \mathbf{y} , FC is the weight matrix of the linear layer. Therefore, for the fully connected neural networks, if the proposed GAM is added before the layer, a scaling vector is added to multiply each feature element by element to obtain a

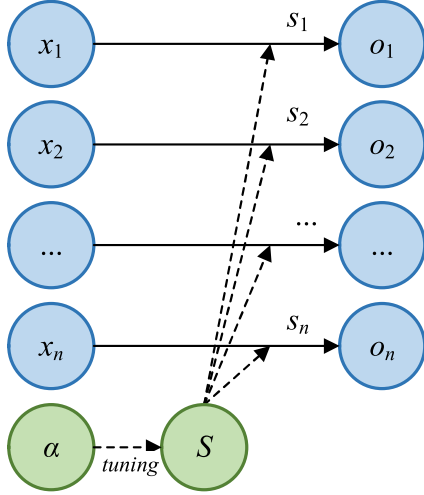


Fig. 1. An abstract representation of the method in this paper. x_i is different feature and s_i is scaling factor for the corresponding features. The scaling factor vector S is calculated by scaling parameter vector α through tuning function. o_i is scaled feature which is x_i times s_i .

new feature vector. The calculated features will replace the original vector for full connection operation. The details of calculation are represented in Fig. 2, and the whole module is named GAM-Linear.

For the convolution operation, commonly used convolution operations are divided into three categories based on the dimension of data, which include one-dimensional (1D) time series data with multi-channels, 2D images, and 3D space data. Let's take the GAM on 2D images as an example. In many research studies, including this paper, it's a common

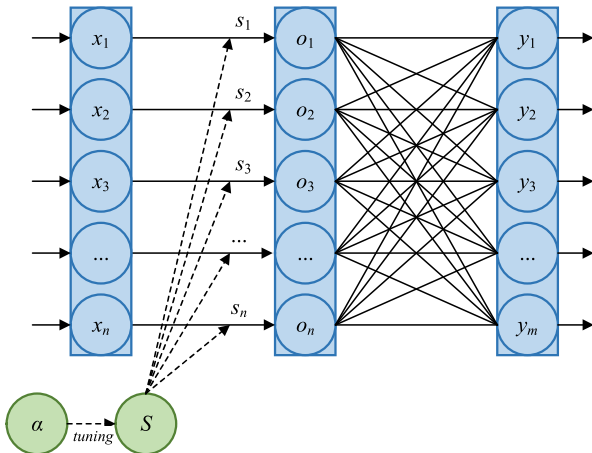


Fig. 2. GAM-Linear: GAM for a fully connected layer.

practice to consider inputs of the same channel as belonging to the same feature map.

For the 2D convolutional layer, let's suppose that the input we obtain is a tensor \mathbf{X} , which has dimensions $C \times W \times H$, representing its number of channels, length, and width of the feature maps, respectively. We can represent the context tensor \mathbf{O} as

$$\mathbf{O}_{i,*,*} = \alpha_i \times \mathbf{X}_{i,*,*}, \quad (8)$$

where $\mathbf{O}_{i,*,*}$ and $\mathbf{X}_{i,*,*}$ denote the i th feature map of the input and output tensor, respectively. Then, the output of the convolutional layer \mathbf{Y} can be calculated as $\mathbf{Y}_{j,*,*} = \text{conv}_j * \mathbf{O}$, where conv_j is the j th convolutional kernel of the current layer, and “ $*$ ” represents the convolutional operator. This process is illustrated in Fig. 3, and the entire module is named GAM-Conv2D.

The GAM plugins for the other two types of data dimensions are almost the same as those shown above, with the exception that Eq. (8) would become $\mathbf{O}_{i,*} = \alpha_i \times \mathbf{X}_{i,*}$ and $\mathbf{O}_{i,*,*,*} = \alpha_i \times \mathbf{X}_{i,*,*,*}$ for GAM-Conv1D and GAM-Conv3D, respectively.

3.1. Attention module embedding and computational reduction

This module is minimally invasive, only slightly altering the structure and functionality of the neural network it is embedded in. We can integrate it without significantly modifying the network, and it can be completely removed after training without affecting inference performance. Given the prevalence of pre-trained models used for model transfer,⁵⁴ rebuilding the model with GAM plugins and re-training the augmented model is inefficient.⁵⁵ Therefore, this paper proposes directly replacing basic neural network operations with operations that include these GAM plugins before each layer of the network. In previous sections, the GAM plugins are always placed before the base operations, combining the two modules to create new operations such as GAM-Linear and GAM-Conv2D. GAM-Linear has two components. First, GAM is added to the input side, with the number of scaling parameters equal to the number of input neurons. Then, the original linear layer is extracted and put at the position following the GAM, retaining its weights and biases. GAM-Conv2D also consists of two parts.

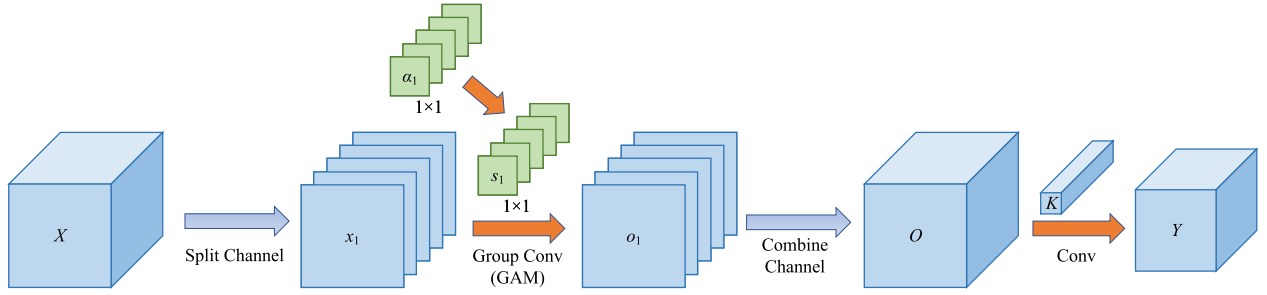


Fig. 3. GAM-Conv2D: GAM for a 2D convolutional layer.

First, a 1×1 depth-wise separable convolution layer is created and added to the input end of the module, with the number of groups and output channels equal to the number of feature maps. Second, the original convolution layer is added to the module, retaining its parameters and biases.

During the inference process, the model parameters have already been determined, so the GAM is no longer necessary to reduce computation. Therefore, this paper also proposes a reduction method to completely eliminate excess computation from this module without affecting the overall network results. In the inference process, an element e_i in the score vector only affects the magnitude of the corresponding feature. This influence can be replaced by scaling the parameters at the corresponding position. For GAM-Linear, y_i is calculated as

$$y_i = \sum_{k=1}^n w_{ik} e_k x_k.$$

The parameters for each feature can be directly scaled to simplify the inference calculation, as shown in the following equation:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} e_1 w_{11} & e_2 w_{12} & \cdots & e_n w_{1n} \\ e_1 w_{21} & e_2 w_{22} & \cdots & e_n w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_1 w_{m1} & e_2 w_{m2} & \cdots & e_n w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \quad (9)$$

Here, the middle matrix represents the actual weight matrix during the inference process.

For GAM-Conv2D, the method to eliminate the effect of the plugin is the same as in GAM-Linear. Suppose that the weights of the GAM module in the GAM-Conv2D layer are W_{GAM} . The weight

$W_{\text{GAM},i}$ is extracted and multiplied by the corresponding weights in the subsequent basic convolution operations, yielding the actual convolution parameters.

3.2. Usage of the GAM

In this section, a detailed usage of GAM will be introduced with reference to algorithm given in Fig. 4. First, the network is initialized. If pre-trained weights exist, they are loaded. After loading pre-trained weights, the added GAM needs to be initialized with the same scores to maintain the effectiveness of pre-trained weights. Parameters in GAM added to networks without loaded pre-trained weights are randomly initialized. Subsequently, the newly constructed network needs to be trained or fine-tuned. After training, the weights in GAM obtain scores through function t . When removing GAM, this score is multiplied onto the corresponding weights. Finally, we obtain a final network that is not affected by GAM.

Input: The original network config Net_config , training dataset D_{train} .
Output: The final network Net^* .
1: Initialise the original network Net with Net_config .
2: if exist pre-trained weights then
3: Load the pre-trained weights into the Net .
4: Add GAM modules before the original layers and get Net' .
5: Initialise the u in the GAM modules with same values.
6: else
7: Add GAM modules before the original layers and get Net' .
8: Initialise the u in the GAM modules with random values.
9: end if
10: Train/Fine-tune the Net' with training dataset D_{train} .
11: Multiply the scores onto the corresponding weights.
12: Remove the GAM modules and get the final network Net^* .

Fig. 4. Pseudocode of applying the GAM in neural networks.

4. Experiments and Analysis

4.1. Datasets and implementation

We conducted experiments on several datasets, selected with consideration for a balance between sample size and feature dimensionality. We selected five datasets from the University of California, Irvine (UCI) Machine Learning Repository: “Wine”,^a “Letter Recognition”,^b “dorothea”,^c “Dry Bean Dataset”,^d and “Shill Bidding Dataset”.^e These datasets vary in size, with the smallest containing approximately 200 samples and the largest containing 20,000 samples. To assess the model’s performance on medium-sized datasets, we conducted experiments on the “MNIST”^f and “Fashion MNIST”^g datasets, each comprising 32×32 gray-scale images with 60,000 samples divided into 10 categories. Additionally, we utilized two large datasets from the UCI Machine Learning Repository to evaluate the model’s performance: “Swarm Behavior Data”^h and “Winnipeg Dataset”.ⁱ For testing the model’s performance on convolutional layers, we selected “CIFAR-10”^j and “CIFAR-100”^k datasets, each comprising 60,000 color images of size 32×32 . We used 10,000 images for testing, 10,000 for validation during training, and the remaining 40,000 for training the model. “CIFAR-10” consists of 10 categories, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck, while “CIFAR-100” contains 100 categories, each with 600 images. We split each dataset, reserving 20% for testing, 20% for validation, and the remainder for training. In the subsequent experiments, multi-layer perceptrons (MLPs) were utilized for experiments and comparisons on datasets excluding CIFAR. For CIFAR-10 and CIFAR-100, we employed several common CNN architectures. The runs have all run for 200 epochs. All models were implemented using

PyTorch and tested on an NVIDIA RTX 2080Ti GPU.

4.2. Experiments on small datasets

To showcase the effectiveness of GAM in fully connected layers, we constructed a classifier consisting solely of fully connected layers. When building the network, we dynamically adjust its architecture based on the number of features and categories. In our experiments, MLPs with five hidden layers were constructed, with the number of nodes in each hidden layer set to half the sum of the number of features and categories. ReLU was employed as the activation function, and dropout was not utilized. The test results for these four datasets are presented in Table 1. These results are based on 10 independent runs. We conducted a *t*-test on these results with a confidence level of 0.05. The mean and standard deviation are displayed in the “Accuracy” column, with statistically significant results highlighted with an underline. It can be observed that, with this setting, the insertion of GAM results in statistically significant improvements in all cases under study. We observed that incorporating GAM had minimal impact on the efficiency of MLPs, particularly for larger network structures. For datasets with numerous features or classes, this method resulted in an increase of less than 0.1% in computational overhead. For the grayscale image datasets “MNIST” and “Fashion MNIST”, the increase was less than 0.3%. Despite the slight increase in computational cost, the accuracy of these datasets improved after incorporating GAM. Notably, our approach led to a significant 4% improvement in accuracy on the “Swarm Behavior Data” dataset.

^a<https://archive.ics.uci.edu/ml/datasets/Wine>.

^b<https://archive.ics.uci.edu/ml/datasets/Letter+Recognition>.

^c<https://archive.ics.uci.edu/ml/datasets/dorothea>.

^d<https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset>.

^e<https://archive.ics.uci.edu/ml/datasets/Shill+Bidding+Dataset>.

^f<http://yann.lecun.com/exdb/mnist/>.

^g<https://github.com/zalandoresearch/fashion-mnist>.

^h<https://archive.ics.uci.edu/ml/datasets/Swarm+Behaviour>.

ⁱ<https://archive.ics.uci.edu/ml/datasets/Crop+mapping+using+fused+optical-radar+data+set>.

^j<https://www.cs.toronto.edu/~kriz/cifar.html>.

^k<https://www.cs.toronto.edu/~kriz/cifar.html>.

Table 1. Experiments on 9 small datasets with five hidden-layer MLP. The statistical significant results are underlined in the ‘‘Accuracy’’ column.

Dataset	Method	Accuracy (%)	FLOPs	Params
Wine	Baseline	90.57±4.05	384	427
	GAM	<u>97.14±2.21(+6.57%)</u>	437(+13.80%)	480(+12.41%)
Letter Recognition	Baseline	95.67±1.97	2646	2777
	GAM	<u>98.11±1.96(+2.45%)</u>	2767(+4.57%)	2898(+4.36%)
MNIST	Baseline	98.11±0.22	945654	947649
	GAM	<u>98.81±0.31(+0.70%)</u>	948423(+0.29%)	950418(+0.29%)
Fashion MNIST	Baseline	89.83±0.36	945654	947649
	GAM	<u>91.05±0.25(+1.22%)</u>	948423(+0.29%)	950418(+0.29%)
Swarm Behavior Data	Baseline	71.80±0.56	8654406	8660413
	GAM	<u>75.87±0.87(+4.06%)</u>	8662811(+0.10%)	8668818(+0.10%)
WinnipegDataset	Baseline	99.19±0.11	48690	49147
	GAM	<u>99.54±0.06(+0.34%)</u>	49314(+1.28%)	49771(+1.27%)
dorothea	Baseline	96.50±1.38	645832	647474
	GAM	<u>98.19±0.86(+1.69%)</u>	648127(+0.36%)	649769(+0.35%)
DryBeanDataset	Baseline	91.77±0.28	737	799
	GAM	<u>94.45±0.19(+2.68%)</u>	808(+9.63%)	870(+8.89%)
ShillBiddingDataset	Baseline	95.02±0.50	155	182
	GAM	<u>97.17±0.24(+2.15%)</u>	189(+21.94%)	216(+18.68%)

4.3. Experiments on CIFAR-10 and CIFAR-100

We selected classical network structures as baselines and integrated the GAM plugin into these networks. We considered networks of various sizes and structures for model selection. For the linear network structure, we opted for the widely recognized VGG-Net.⁵⁶ For network structures with skip-connections, we chose ResNet⁵⁷ as the baseline. In addition, we have conducted experiments on some recent networks to illustrate the generalization of GAM, including some ViT networks. In the implementation of these algorithms, we integrated the plugin into existing models to compare their performance before and after plugin integration.¹ To highlight the positive impact of the GAM plugin on feature control, we omitted the learning rate schedule during training and used a relatively large learning rate. Consequently, our final experimental results may slightly differ from published baselines in other studies. We selected a maximum learning rate between 0.1 and 0.5, ensuring no invalid parameters were produced during testing. All comparisons were conducted using paired learning rates within the same group.

For ViT networks, we modified the original network with a final layer to apply to CIFAR training and prediction. In order to keep the performance of ViT from degrading too much, we scaled the image size of CIFAR to 64. The accuracy curves on the test dataset are depicted in Fig. 5, while the experimental results are presented in Tables 2 and 3.

From the displayed results, we observed that the GAM plugin effectively adjusts parameter updates for different features during the training process, resulting in earlier flattening of the learning curves. Furthermore, as depicted in Fig. 5, we observed that the original version of AlexNet exhibits significant overfitting during training on the CIFAR-100 dataset, characterized by a notable decline in validation accuracy midway through training, with difficulty in returning to an optimal state thereafter. Upon investigation, we discovered that for this image size, AlexNet lacks fully-connected layers and does not include Dropout operations. This observation suggests that our approach has a modest capacity for mitigating overfitting. Additionally, our method demonstrates a substantial improvement when combined with SENet on the CIFAR-100 dataset. We hypothesize that this enhancement may be

¹The implementation of comparison algorithms can be found at <https://github.com/kuangliu/pytorch-cifar>, <https://github.com/BIGBALLON/CIFAR-ZOO> and timm package.

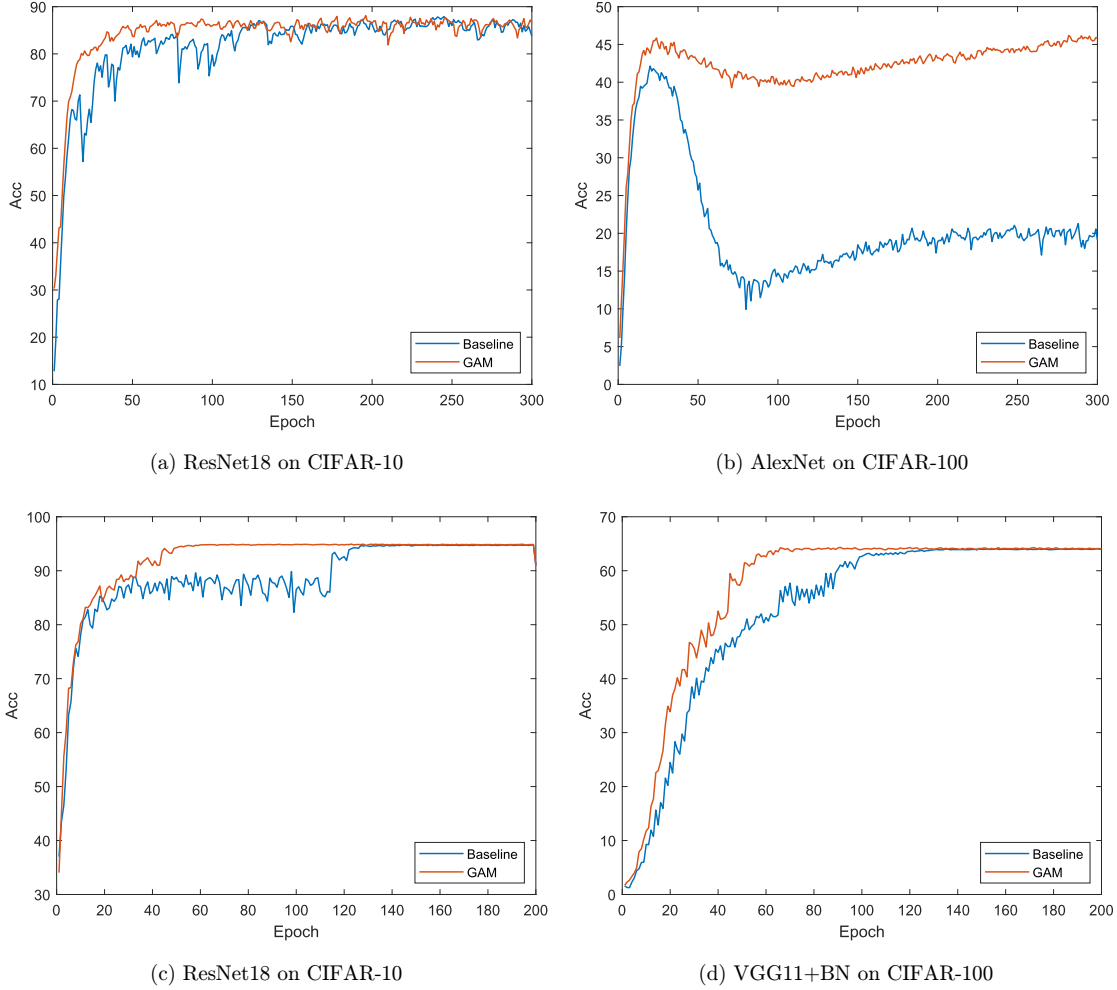


Fig. 5. The curve of accuracy during training.

attributed to the channel AM being further refined by GAM, enabling a more pronounced emphasis on critical features.

It must be remarked that, besides its use for images, the proposed GAM, following some modifications and redesign, could be integrated within the Multi-Head AMs of a transformer or a large language model. In the case of text analysis/prediction, instead of feature maps, there would be numeric vectors, and instead of convolution operators, there would be linear operators.

4.4. Feature visualization analysis

To delve deeper into the impact of GAM on neural network training, Grad-CAM was employed to visualize attention regions on various images.

We present 27 image sets for comparison in Fig. 6, obtained from comparative experiments conducted on ResNet-18 with and without generalized attention mechanism (GAM) on the CIFAR-10 dataset. Each set comprises three images: the original image from the dataset on the left, the attention region of the baseline model in the middle, and the attention region of ResNet-18 with GAM incorporated on CIFAR-10 on the right. Upon observation, it becomes evident that the baseline model often focuses on incorrect regions in many images. Additionally, the baseline model struggles to accurately identify features corresponding to the target category, as indicated by large areas of yellow or red in certain images. In contrast, the addition of GAM enhances the neural network’s ability to accurately localize the recognition target and extract relevant

Table 2. Experiments on CIFAR-10 with eleven backbone networks (including CNNs and ViTs).

	Method	Top-1 (%)	Top-5 (%)	FLOPs	Params
Resnet18 ⁵⁷	Baseline	89.17	99.64	556.6515M	11.1740M
	GAM	89.55(+0.38%)	99.69(+0.05%)	556.6597M(+39.0080K)	11.1749M(+3.8430K)
Resnet50 ⁵⁷	Baseline	91.29	99.64	1.3047G	23.5208M
	GAM	92.07(+0.78%)	99.80(+0.16%)	1.3047G(+223.3280K)	23.5247M(+22.5310K)
VGG11+BN ⁵⁶	Baseline	82.02	98.68	153.2247M	9.2311M
	GAM	84.16(+2.14%)	98.79(+0.96%)	153.2340M(+9.3120K)	9.2334M(+2.2430K)
VGG19+BN ⁵⁶	Baseline	90.14	99.54	399.0461M	20.0405M
	GAM	90.71(+0.57%)	99.54(+0.00%)	399.0698M(+23.6480K)	20.0455M(+4.9950K)
SENet18 ⁴¹	Baseline	90.64	99.51	556.7867M	11.3065M
	GAM	91.10(+0.46%)	99.69(+0.18%)	556.8282M(+41.5600K)	11.3129M(+6.3950K)
ShuffleNetV2 ⁵⁸	Baseline	87.3	99.53	556.7867M	11.3065M
	GAM	89.06(+1.76%)	99.62(+0.09%)	556.8282M(+41.5600K)	11.3129M(+6.3950K)
DLA ⁵⁹	Baseline	92.47	99.81	1.0333G	16.2914M
	GAM	93.54(+1.07%)	99.81(+0.00%)	1.0334G(+101.9840K)	16.3015M(+10.1470K)
RegNetX (200MF) ⁶⁰	Baseline	92.95	99.84	226.5599M	2.3552M
	GAM	94.42(+1.47%)	99.88(+0.04%)	226.6706M(+110.6720K)	2.3653M(+10.1550K)
EfficientViT (b0) ⁶¹	Baseline	82.78	93.37	9.7326M	2.1418M
	GAM	85.63(+2.85%)	95.91(+2.54%)	9.9645M(+0.2319M)	2.1589M(+0.0171M)
ConvMixer (768) ⁶²	Baseline	89.30	96.81	1.6518G	20.3489M
	GAM	90.22(+0.92%)	97.42(+0.61%)	1.6557G(+3.9936M)	20.4472M(+0.0983M)
FastViT (t8) ⁶³	Baseline	90.23	97.56	43.6769M	3.2427M
	GAM	91.55(+1.32%)	98.01(+0.45%)	44.2353M(+0.5584M)	3.2744M(+0.0317M)

Table 3. Experiments on CIFAR-100 with eleven backbone networks (including CNNs and ViTs).

	Method	Top-1 (%)	Top-5 (%)	FLOPs	Params
AlexNet ⁶⁴	Baseline	42.14	71.80	63.1542M	23.6410M
	GAM	46.18(+4.04%)	75.63(+3.83%)	63.1672M(+3.9360K)	23.6511M(+0.0899K)
Resnet50 ⁵⁷	Baseline	55.68	80.62	1.3049G	23.7053M
	GAM	56.09(+0.41%)	79.63(-0.99%)	1.3049G(+223.3280K)	23.7053M(+22.5310K)
VGG11+BN ⁵⁶	Baseline	66.06	88.41	153.2708M	9.2773M
	GAM	66.34(+0.28%)	88.39(-0.02%)	153.2801M(+9.3120K)	9.2795M(+2.2430K)
ShuffleNetV1 ⁶⁵	Baseline	60.84	86.24	41.0913M	887.5820K
	GAM	61.09(+0.25%)	87.24(+1.00%)	41.1663M(+74.9760K)	898.0970K(+10.5150K)
ShuffleNetV2 ⁵⁸	Baseline	58.70	85.29	11.4232M	352.0420K
	GAM	59.28(+0.58%)	86.17(+0.88%)	11.4493M(+26.0800K)	356.1890K(+4.1470K)
DLA ⁵⁹	Baseline	68.1	89.46	1.0334G	16.3376M
	GAM	70.6(+2.50%)	91.09(+1.63%)	1.0335G(+101.9840K)	16.3477M(+10.1470K)
RegNetX (200MF) ⁶⁰	Baseline	71.11	91.94	226.5599M	2.3552M
	GAM	73.8(+1.47%)	93.06(+0.04%)	226.6706M(+110.6720K)	2.3653M(+10.1550K)
SENet18 ⁴¹	Baseline	64.39	88.24	226.5599M	2.3552M
	GAM	72.6(+8.21%)	91.93(+3.69%)	226.6706M(+110.6720K)	2.3653M(+10.1550K)
EfficientViT (b0) ⁶¹	Baseline	68.98	89.67	9.8478M	2.2571M
	GAM	69.51(+0.53%)	90.19(+0.52%)	10.0797M(+0.2319M)	2.2742M(+0.0171M)
ConvMixer (768) ⁶²	Baseline	73.07	92.25	1.6518G	20.4181M
	GAM	73.91(+0.84%)	93.88(+1.63%)	1.6558G(+3.9936M)	20.5164M(+0.0983M)
FastViT (t8) ⁶³	Baseline	72.07	91.17	43.7460M	3.3120M
	GAM	73.40(+1.33%)	92.99(+1.82%)	44.3044M(+0.5584M)	3.3436M(+0.0317M)

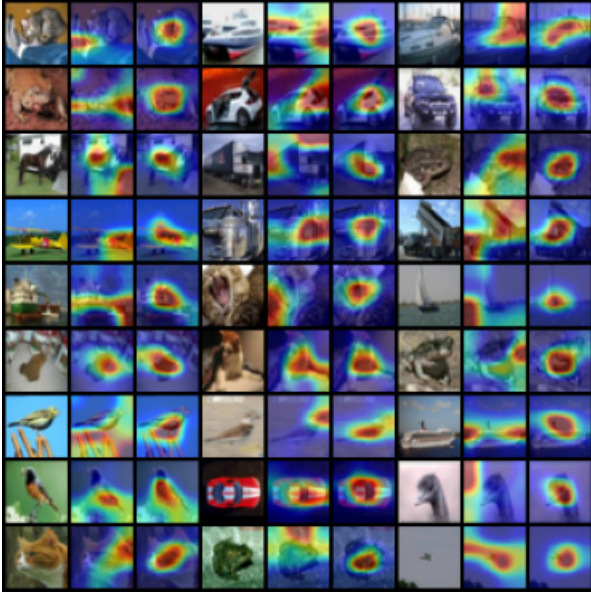


Fig. 6. Feature heatmaps using ResNet-18 on CIFAR-10. The left of each set is the original picture, the middle are results with the original model, and the right are results with our model.

features. In some instances, the network can even precisely focus on the entire recognition target. For example, on small objects (such as the 9th in the last column), the GAM can accurately focus on the region where the target object is located. For images with complete forms, such as some bird images shown in the display, GAM can focus on the entire body of the bird, but the attention capability of the original model is relatively poor. For more complex images, such as images with only partial structures of category objects (such as the ostrich and the cat in the last two rows), GAM can focus on the head of the object to more accurately capture features.

4.5. Ablation study

To demonstrate the advantages of our module compared to others that use AMs, we designed an ablation experiment. We have added different attention modules based on ResNet-18 backbone network, including SE module, CBAM module, separated attention module, and our proposed GAM module. In this part of the experiment, we used a learning rate decay strategy to better fit the model to the dataset. The results of the experiments in this section on CIFAR-10 are shown in Table 4. From this table, we

Table 4. The ablation study on the attention modules.

Attention methods	Accuracy (%)	FLOPs (M)	Params (M)
SE	96.11	557.97	11.26
CBAM	95.16	558.33	11.26
Split-attention	96.02	1107.30	15.04
GAM	96.33	556.66	11.17

can observe that our method has significant advantages compared to other attention-based approaches, both in terms of performance and resource consumption (FLOPs and parameter count).

In order to further explore the impact of GAM on computational cost, we conducted additional experiments on the basic single-layer module. First, we utilized linear layers and varied the number of input and output neurons. According to the results in Table 5, we found that when GAM was used to process input data, the additional computation was only related to the number of input neurons. The proportion of additional computation was also influenced by the size of weights. In addition, we conducted similar experiments on the convolutional layer. Table 6 illustrates that although the additional computational cost of GAM is influenced by the hyperparameters of each convolutional layer, even when the weight size is very small, the proportion of additional computational cost does not exceed 2%. Combining these two tables, we can easily conclude that when GAM is applied to neural networks, the computational cost is not significantly affected. Especially in large networks, the impact of GAM on computational cost can be almost negligible.

Table 5. The increase rate of the FLOPs when introducing the GAM in one linear layer.

#in	#out	Baseline	GAM	↑
100	100	10 K	10.1 K	0.1 K (1%)
100	200	20 K	20.1 K	0.1 K (0.5%)
100	300	30 K	30.1 K	0.1 K (0.33%)
200	100	20 K	20.2 K	0.2 K (1%)
200	200	40 K	40.2 K	0.2 K (0.5%)
200	300	60 K	60.2 K	0.2 K (0.33%)
300	100	30 K	30.3 K	0.3 K (1%)
300	200	60 K	60.3 K	0.3 K (0.5%)
300	300	90 K	90.3 K	0.3 K (0.33%)

Table 6. The increase rate of the FLOPs when introducing the GAM in one convolutional layer.

Different sets of hyperparameters		Baseline	GAM	↑
out=10	in=10	0.1764M	0.17896M	1.45%
$S_f=16$	in=20	0.3528M	0.35792M	1.45%
$S_k=3$	in=40	0.7056M	0.71584M	1.45%
in=10	out=10	0.1764M	0.17896M	1.45%
$S_f=16$	out=20	0.3528M	0.35536M	0.73%
$S_k=3$	out=40	0.7056M	0.70816M	0.36%
in=10	$S_f=8$	0.0324M	0.03304M	1.98%
out=10	$S_f=16$	0.1764M	0.17896M	1.45%
$S_k=3$	$S_f=32$	0.81M	0.82024M	1.26%
in=10	$S_k=3$	0.1764M	0.17896M	1.45%
out=10	$S_k=5$	0.36M	0.36256M	0.71%
$S_f=16$	$S_k=7$	0.49M	0.49256M	0.52%

5. Conclusion


In this paper, we proposed a modified attention module called GAM, which can adaptively identify important features or feature maps during the training process and focus on training the relevant parameters. Our method demonstrates good results on various datasets and can accelerate the convergence of the training curve by leveraging this property during training. However, when faced with overfitting, although this method has some preventive effects, its mechanism is difficult to determine. Therefore, it requires further research on how GAM can prevent overfitting and how to enhance its ability to prevent overfitting. In the future, it may be necessary to introduce some random noise as an augmentation strategy to improve this performance. Furthermore, our method has shown outstanding performance in neural networks with AMs. However, there are various types of AMs in the existing networks, and it is still unclear whether GAM can perform well in all types of attention modules. In particular, in certain specific networks, GAM seems to achieve greater performance improvements. Therefore, more detailed research needs to be conducted to ensure that GAM can be more appropriately used in the network. In conclusion, our proposed GAM module is a promising approach for aiding the training process in DL. It achieves good results on various datasets, accelerates the convergence of the training curve, and performs well in neural networks with AMs. However, further


research is needed to better understand its mechanism and enhance its ability to prevent overfitting.

Acknowledgments


This work was partially supported by the National Natural Science Foundation of China (62376127, 61876089, 61876185, 61902281), the Natural Science Foundation of Jiangsu Province (BK20141005), and the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (14KJB520025), Jiangsu Distinguished Professor Programme.

ORCID

Pengcheng Jiang  <https://orcid.org/0000-0002-9625-697X>

Ferrante Neri  <https://orcid.org/0000-0002-6100-6532>

Yu Xue  <https://orcid.org/0000-0002-9069-7547>

Ujjwal Maulik  <https://orcid.org/0000-0003-1167-0774>

References

1. Y. Pei, Y. Huang, Q. Zou, X. Zhang and S. Wang, Effects of image degradation and degradation removal to CNN-based image classification, *IEEE Trans. Pattern Anal. Mach. Intell.* **43**(4) (2021) 1239–1253.
2. G. Mirzaei and H. Adeli, Machine learning techniques for diagnosis of Alzheimer disease, mild cognitive disorder, and other types of dementia, *Biomed. Signal Proc. Control* **72** (2022) 103293.
3. J. Shi, Z. Li, W. Lai, F. Li, R. Shi, Y. Feng and S. Zhang, Two end-to-end quantum-inspired deep neural networks for text classification, *IEEE Trans. Knowl. Data Eng.* **35**(4) (2023) 4335–4345.
4. N. Mammone, C. Ieracitano, H. Adeli and F. C. Morabito, Autoencoder filter bank common spatial patterns to decode motor imagery from EEG, *IEEE J. Biomed. Health Inform.* **27**(5) (2023) 2365–2376.
5. M. H. Rafiei, L. V. Gauthier, H. Adeli and D. Takabi, Self-supervised learning for electroencephalography, *IEEE Trans. Neural Netw. Learn. Syst.* **35**(2) (2024) 1457–1471.
6. F. Colonnese, F. D. Luzio, A. Rosato and M. Panella, Bimodal feature analysis with deep learning for autism spectrum disorder detection, *Int. J. Neural Syst.* **34**(2) (2024) 2450005.
7. Z. Zhou, J. Zhang and C. Gong, Hybrid semantic segmentation for tunnel lining cracks based on swin transformer and convolutional neural network,

- Comput.-Aided Civ. Infrastruct. Eng.* **38** (2023) 2491–2510.
8. Z. Wang, Y. Zhang, K. M. Mosalam, Y. Gao and S. Huang, Deep semantic segmentation for visual understanding on construction sites, *Comput.-Aided Civ. Infrastruct. Eng.* **37** (2022) 145–162.
 9. H. S. Nogay and H. Adeli, Machine learning (ML) for the diagnosis of autism spectrum disorder (ASD) using brain imaging, *Rev. Neurosci.* **31** (2020) 825–841.
 10. H. S. Nogay and H. Adeli, Multiple classification of brain MRI autism spectrum disorder by age and gender using deep learning, *J. Med. Syst.* **48** (2024) 15.
 11. H. S. Nogay and H. Adeli, Detection of epileptic seizure using pretrained deep convolutional neural network and transfer learning, *Eur. Neurol.* **83**(6) (2020) 602–614.
 12. H. Selcuk Nogay and H. Adeli, Diagnostic of autism spectrum disorder based on structural brain MRI images using, grid search optimization, and convolutional neural networks, *Biomed. Signal Proc. Control* **79** (2023) 104234.
 13. B. K. Oh, S. H. Yoo and H. S. Park, A measured data correlation-based strain estimation technique for building structures using convolutional neural network, *Integr. Comput.-Aided Eng.* **30** (2023) 395–412.
 14. T. Siriborvornratanakul, Pixel-level thin crack detection on road surface using convolutional neural network for severely imbalanced data, *Comput.-Aided Civ. Infrastruct. Eng.* **38** (2023) 2300–2316.
 15. G. B. Martins, J. P. Papa and H. Adeli, Deep learning techniques for recommender systems based on collaborative filtering, *Expert Syst.* **37** (2020) e12647.
 16. L. Ruiz, S. Diaz, J. M. González and F. Cavas, Improving the competitiveness of aircraft manufacturing automated processes by a deep neural network, *Integr. Comput.-Aided Eng.* **30** (2023) 341–352.
 17. J. Urdiales, D. Martín and J. M. Armingol, An improved deep learning architecture for multi-object tracking systems, *Integr. Comput.-Aided Eng.* **30** (2023) 121–134.
 18. D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, *Nature* **323**(6088) (1986) 533–536.
 19. G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science* **313**(5786) (2006) 504–507.
 20. R. Groenendijk, L. Dorst and T. Gevers, Geometric back-propagation in morphological neural networks, *IEEE Trans. Pattern Anal. Mach. Intell.* **45** (2023) 14045–14051.
 21. M. R. Zhang, J. Lucas, G. Hinton and J. Ba, Lookahead optimizer: k steps forward, 1 step back, in *Advances in Neural Information Processing Systems*, Vol. 32 (Curran Associates, 2019), pp. 9593–9604.
 22. B. Woodworth, K. K. Patel, S. Stich, Z. Dai, B. Bullins, B. McMahan, O. Shamir and N. Srebro, Is local SGD better than minibatch SGD? in *Proc. 37th Int. Conf. Machine Learning, Proc. Machine Learning Research*, Vol. 119 (PMLR, 2020), pp. 10334–10343.
 23. D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, in *Int. Conf. Learning Representations*, San Diego, CA, USA (2015).
 24. P. Jiang, Y. Xue and F. Neri, Continuously evolving dropout with multi-objective evolutionary optimisation, *Eng. Appl. Artif. Intell.* **124** (2023) 106504.
 25. M. S. Shahabi, A. Shalbaf, B. Nobakhsh, R. Rostami and R. Kazemi, Attention-based convolutional recurrent deep neural networks for the prediction of response to repetitive transcranial magnetic stimulation for major depressive disorder, *Int. J. Neural Syst.* **33**(2) (2023) 2350007.
 26. Y. Xue, C. Zhang, F. Neri, M. Gabbouj and Y. Zhang, An external attention-based feature ranker for large-scale feature selection, *Knowl. Based Syst.* **281** (2023) 111084.
 27. C. Zhang, Y. Xue, F. Neri, X. Cai and A. Slowik, Multi-objective self-adaptive particle swarm optimization for large-scale feature selection in classification, *Int. J. Neural Syst.* **34**(3) (2024) 2450014.
 28. L. Zhang, Z. Zhou, P. Ji and A. Mei, Application of attention mechanism with prior information in natural language processing, *Int. J. Artif. Intell. Tools* **31**(4) (2022) 2240008.
 29. M. Guo, T. Xu, J. Liu, Z. Liu, P. Jiang, T. Mu, S. Zhang, R. R. Martin, M. Cheng and S. Hu, Attention mechanisms in computer vision: A survey, *Comput. Vis. Media* **8**(3) (2022) 331–368.
 30. A. Hassanpour, M. Moradikia, H. Adeli, S. R. Khayami and P. Shamsinejadbabaki, A novel end-to-end deep learning scheme for classifying multi-class motor imagery electroencephalography signals, *Expert Syst.* **36** (2019) e12494.
 31. D. R. Pereira, M. A. Piteri, A. N. Souza, J. P. Papa and H. Adeli, FEMa: A finite element machine for fast learning, *Neural Comput. Appl.* **32** (2020) 6393–6404.
 32. K. M. R. Alam, N. Siddique and H. Adeli, A dynamic ensemble learning algorithm for neural networks, *Neural Comput. Appl.* **32** (2020) 8675–8690.
 33. M. H. Rafiei and H. Adeli, A new neural dynamic classification algorithm, *IEEE Trans. Neural Netw. Learn. Syst.* **28** (2017) 3074–3083.
 34. M. Matinfar, N. Khaji and G. Ahmadi, Deep convolutional generative adversarial networks for the generation of numerous artificial spectrum-compatible earthquake accelerograms using a limited number of ground motion records, *Comput.-Aided Civ. Infrastruct. Eng.* **38** (2023) 225–240.

35. D. Bahdanau, K. H. Cho and Y. Bengio, Neural machine translation by jointly learning to align and translate, in *Int. Conf. Learning Representations*, San Diego, CA, USA (2015).
36. M.-T. Luong, H. Pham and C. D. Manning, Effective approaches to attention-based neural machine translation, in *Proc. 2015 Conf. Empirical Methods in Natural Language Processing* (The Association for Computational Linguistics, 2015), pp. 1412–1421.
37. C. Zhu, W. Ping, C. Xiao, M. Shoeybi, T. Goldstein, A. Anandkumar and B. Catanzaro, Long-short transformer: Efficient transformers for language and vision, *Adv. Neural Inf. Process. Syst.* **34** (2021) 17723–17736.
38. J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in *Proc. 2019 Conf. North*, eds. J. Burstein, C. Doran and T. Solorio (Association for Computational Linguistics, 2019), pp. 4171–4186.
39. T. Brown et al., Language models are few-shot learners, *Adv. Neural Inf. Process. Syst.* **33** (2020) 1877–1901.
40. S. Jung, J. Jeoung, H. Kang and T. Hong, 3D convolutional neural network-based one-stage model for real-time action detection in video of construction equipment, *Comput.-Aided Civ. Infrastruct. Eng.* **37** (2022) 126–142.
41. J. Hu, L. Shen, S. Albanie, G. Sun and E. Wu, Squeeze-and-excitation networks, *IEEE Trans. Pattern Anal. Mach. Intell.* **42**(8) (2020) 2011–2023.
42. S. Woo, J. Park, J.-Y. Lee and I. So Kweon, CBAM: Convolutional block attention module, in *Proc. European Conf. Computer Vision (ECCV)* (Springer, 2018), pp. 3–19.
43. S. Sang, Y. Zhou, M. T. Islam and L. Xing, Small-object sensitive segmentation using across feature map Attention, *IEEE Trans. Pattern Anal. Mach. Intell.* **45**(5) (2023) 6289–6306.
44. H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li and A. Smola, ResNeSt: Split-attention networks, in *2022 IEEE/CVF Conf. Computer Vision and Pattern Recognition Workshops (CVPRW)* (IEEE Press, 2022), pp. 2735–2745.
45. Z. Liu, X. Yang, H. Tang, S. Yang and S. Han, FlatFormer: Flattened window attention for efficient point cloud transformer, in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE Computer Society, 2023), pp. 1200–1211.
46. L. Zhu, X. Wang, Z. Ke, W. Zhang and R. Lau, BiFormer: Vision transformer with bi-level routing attention, in *2023 IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2023), pp. 1–10.
47. T. Yao, Y. Li, Y. Pan, Y. Wang, X.-P. Zhang and T. Mei, Dual vision transformer, *IEEE Trans. Pattern Anal. Mach. Intell.* **45**(9) (2023) 10870–10882.
48. H. Zhu, J. Wang, S. Wang, R. Raman, J. M. Górriz and Y. Zhang, An evolutionary attention-based network for medical image classification, *Int. J. Neural Syst.* **33**(3) (2023) 2350010.
49. J. E. Arco, A. Ortiz, N. Gallego-Molina, J. M. Górriz and J. Ramírez, Enhancing multimodal patterns in neuroimaging by siamese neural networks with self-attention mechanism, *Int. J. Neural Syst.* **33**(4) (2023) 2350019.
50. Z. Wang, S. Hou, T. Xiao, Y. Zhang, H. Lv, J. Li, S. Zhao and Y. Zhao, Lightweight seizure detection based on multi-scale channel attention, *Int. J. Neural Syst.* **33**(12) (2023) 2350061.
51. D. Liu, X. Dong, D. Bian and W. Zhou, Epileptic seizure prediction using attention augmented convolutional network, *Int. J. Neural Syst.* **33**(11) (2023) 2350054.
52. Y. Zhao, J. He, F. Zhu, T. Xiao, Y. Zhang, Z. Wang, F. Xu and Y. Niu, Hybrid attention network for epileptic EEG classification, *Int. J. Neural Syst.* **33**(6) (2023) 2350031.
53. Y. Xue, Y. Zhang and F. Neri, A method based on evolutionary algorithms and channel attention mechanism to enhance cycle generative adversarial network performance for image translation, *Int. J. Neural Syst.* **33**(5) (2023) 2350026.
54. N. Mammone, C. Ieracitano, R. Spataro, C. Guger, W. Cho and F. C. Morabito, A few-shot transfer learning approach for motion intention decoding from electroencephalographic signals, *Int. J. Neural Syst.* **34**(2) (2024) 2350068.
55. M. Zhao, C. Zhang, J. Zhang, F. Porikli, B. Ni and W. Zhang, Scale-aware crowd counting via depth-embedded convolutional neural networks, *IEEE Trans. Circuits Syst. Video Technol.* **30**(10) (2020) 3651–3662.
56. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *Int. Conf. Learning Representations*, San Diego, CA, USA (2015).
57. K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2016), pp. 770–778.
58. N. Ma, X. Zhang, H.-T. Zheng and J. Sun, ShuffleNet V2: Practical guidelines for efficient CNN architecture design, in *Proc. European Conf. Computer Vision (ECCV)* (Springer International Publishing, 2018), pp. 116–131.
59. F. Yu, D. Wang, E. Shelhamer and T. Darrell, Deep layer aggregation, in *2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition* (IEEE, 2018), pp. 2403–2412.

60. I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He and P. Dollar, Designing network design spaces, in *2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2020), pp. 10425–10433.
61. H. Cai, J. Li, M. Hu, C. Gan and S. Han, EfficientViT: Lightweight multi-scale attention for high-resolution dense prediction, in *2023 IEEE/CVF Int. Conf. Computer Vision (ICCV)* (IEEE, 2023), pp. 17256–17267.
62. A. Trockman and J. Z. Kolter, Patches are all you need? *Trans. Mach. Learn. Res.* (2023).
63. P. K. Anasosalu Vasu, J. Gabriel, J. Zhu, O. Tuzel and A. Ranjan, FastViT: A fast hybrid vision transformer using structural reparameterization, in *2023 IEEE/CVF Int. Conf. Computer Vision (ICCV)* (IEEE, 2023), pp. 5785–5795.
64. A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, *Commun. ACM* **60**(6) (2017) 84–90.
65. X. Zhang, X. Zhou, M. Lin and J. Sun, ShuffleNet: An extremely efficient convolutional neural network for mobile devices, in *2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition* (IEEE, 2018), pp. 6848–6856.