# A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects

Zewen Li, Fan Liu, *Member, IEEE*, Wenjie Yang, Shouheng Peng, and Jun Zhou, *Senior Member, IEEE*

***Abstract*— A convolutional neural network (CNN) is one of the most significant networks in the deep learning field. Since CNN made impressive achievements in many areas, including but not limited to computer vision and natural language processing, it attracted much attention from both industry and academia in the past few years. The existing reviews mainly focus on CNN's applications in different scenarios without considering CNN from a general perspective, and some novel ideas proposed recently are not covered. In this review, we aim to provide some novel ideas and prospects in this fast-growing field. Besides, not only 2-D convolution but also 1-D and multidimensional ones are involved. First, this review introduces the history of CNN. Second, we provide an overview of various convolutions. Third, some classic and advanced CNN models are introduced; especially those key points making them reach state-of-the-art results. Fourth, through experimental analysis, we draw some conclusions and provide several rules of thumb for functions and hyperparameter selection. Fifth, the applications of 1-D, 2-D, and multidimensional convolution are covered. Finally, some open issues and promising directions for CNN are discussed as guidelines for future work.**

***Index Terms*— Computer vision, convolutional neural networks (CNNs), deep learning, deep neural networks.**

## I. Introduction

CONVOLUTIONAL neural network (CNN) has been making brilliant achievements. It has become one of the most representative neural networks in the field of deep learning. Computer vision based on CNN has enabled people to accomplish what had been considered impossible in the past few centuries, such as face recognitions, autonomous vehicles, self-service supermarkets, and intelligent medical treatments. To better understand modern CNN and make it better serve human beings, in this article, we present an overview of convolutions, introduce classic models and applications, and propose some prospects for CNN.

The emergence of CNNs cannot be separated from artificial neural networks (ANNs). McCulloch and Pitts [1] proposed the first mathematical model of neurons—the MP model. Rosenblatt [2], [3] proposed a single-layer perceptron model by adding learning ability to the MP model. However, single-layer perceptron networks cannot handle linear inseparable problems (such as XOR problems). Rumelhart *et al.* [4] proposed a multilayer feedforward network trained by the error backpropagation algorithm—backpropagation network (BP network), which addressed some problems that single-layer perceptron could not solve. Waibel *et al.* [5] proposed a time-delay neural network for speech recognition, which can be viewed as a 1-D CNN. Then, Zhang [6] proposed the first 2-D CNN—shift-invariant ANN. LeCun *et al.* [7] also constructed a CNN for handwritten zip code recognition and first used the term "convolution," which is the original version of LeNet. Various shallow neural networks were successively proposed, such as Chaotic neural networks proposed by Aihara *et al.* [8] and a general regression neural network proposed by Specht *et al.* [9]. The most famous one is LeNet-5 [10]. Nevertheless, when the number of layers of neural networks is increased, traditional BP networks would encounter problems, such as local optimum, overfitting, gradient vanishing, and gradient exploding. Krizhevsky *et al.* [11] proposed the following points: 1) multihidden layers ANNs have excellent feature learning abilities and 2) the "layerwise pretraining" can effectively overcome the difficulties of training deep neural networks, which brought about the study of deep learning. Krizhevsky *et al.* [11] achieved the best classification result at that time using deep CNN in the ImageNet Large Scale Visual Recognition Challenge (LSVRC), which attracted researchers much attention and greatly promoted the development of modern CNN.

Before our work, there exist several researchers who have reviewed CNN. Aloysius and Geetha [12] paid attention to frameworks of deep learning chronologically. Nevertheless, they did not fully explain why these architectures were better than their predecessors and how they achieved their goals. Dhillon and Verma [13] discussed the architectures of some classic networks, but there are many new-generation networks, after their work, that have been proposed, such as MobileNet v3, inception v4, and ShuffleNet series, which deserve researchers' attention. Besides, the work reviewed applications of CNN for object detection. Rawat and Wang [14] reviewed CNN for image recognition. Liu *et al.* [15] discussed CNN for image recognition. Ajmal *et al.* [16] discussed CNN for image segmentation. These reviews mainly focused on the applications of CNN in different scenarios without considering CNN from a general perspective. In addition, these reviews did not cover the newly proposed inspiring ideas.

In this article, we focus on analyzing and discussing CNN. In detail, the key contributions of this review are as follows: 1) we provide an overview of some inspiring convolutions, including basic building blocks of modern CNN, deformable convolution, group convolution, steerable convolution, and so on. 2) Some classic CNN-based models are covered, from

Manuscript received 31 March 2020; revised 21 February 2021; accepted 18 May 2021. Date of publication 10 June 2021; date of current version 1 December 2022. This work was supported in part by the Natural Science Foundation of Jiangsu Province under Grant BK20191298 and in part by the Fundamental Research Funds for the Central Universities under Grant B200202175. *(Corresponding author: Fan Liu.)*

Zewen Li, Fan Liu, Wenjie Yang, and Shouheng Peng are with the College of Computer and Information, Hohai University, Nanjing 210098, China (e-mail: servon@hhu.edu.cn; fanliu@hhu.edu.cn; vicent@hhu.edu.cn; shoehengpeng@hhu.edu.cn).

Jun Zhou is with the School of Information and Communication Technology, Griffith University, Nathan, QLD 4111, Australia (e-mail: jun.zhou@griffith.edu.au).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3084827.

Digital Object Identifier 10.1109/TNNLS.2021.3084827

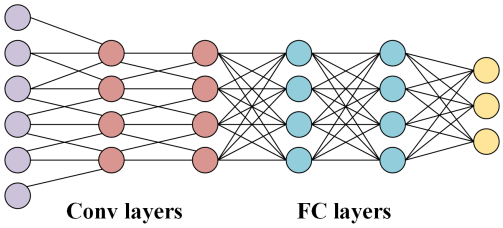



Fig. 1. Diagram of CNN layers and FC layers.

LeNet-5, AlexNet to MobileNet v3, and GhostNet. Innovations of these models are emphasized to help readers draw some useful experience from masterpieces. 3) Several representative activation functions, loss functions, optimizers, and hyperparameters are discussed. We have reached some conclusions about them through experiments. 4) Although applications of 2-D convolution are widely used, 1-D and multidimensional ones should not be ignored. Some typical applications are presented. 5) We raise several points of view on prospects for CNN. Parts of them are intended to refine existing CNNs, and the others design new networks from scratch.

We organize the rest of this article as follows: Section II overviews several convolutions and briefly introduces some classic CNN-based models. We mainly focus on the innovations of these models, rather than all details. Section III discusses some representative activation functions, loss functions, optimizers, and hyperparameters, which can help readers to select them appropriately. Section IV covers some applications of CNN from the perspective of different dimensional convolutions. Section V discusses current challenges and several promising trends of CNN for future work. Section VI concludes the survey by giving a bird's eye view of our contributions.

## II. Overview of Several Convolutions

The CNN is a kind of feedforward neural network that is able to extract features from data with convolution structures. Different from the traditional feature extraction methods [17]–[19], CNN does not need to extract features manually. The architecture of CNN is inspired by visual perception [20]. A biological neuron corresponds to an artificial neuron; CNN kernels represent different receptors that can respond to various features; activation functions simulate the function that only neural electric signals exceeding a certain threshold can be transmitted to the next neuron. Loss functions and optimizers are something people invented to teach the whole CNN system to learn what we expect. Compared with fully connected (FC) networks in Fig. 1, CNN possesses many advantages: 1) local connections—Each neuron is no longer connected to all neurons of the previous layer, but only to a small number of neurons, which is effective in reducing parameters and speed up convergence. 2) Weight sharing—a group of connections can share the same weights, which reduces parameters further. 3) Downsampling dimension reduction—a pooling layer harnesses the principle of image local correlation to downsample an image, which can reduce the amount of data while retaining useful information. It can also reduce the number of parameters by removing trivial features. These three appealing characteristics make CNN one of the most representative algorithms in the deep learning field.

To be specific, in order to build a CNN model, four components are typically needed. Convolution is a pivotal step for feature extraction. The outputs of convolution can be

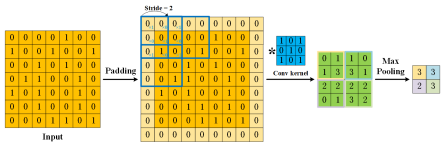



Fig. 2. Procedure of a 2-D CNN.

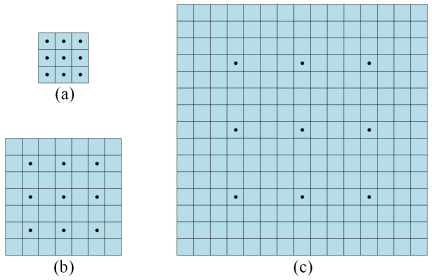



Fig. 3. Comparison between general convolution kernel and dilated convolution kernel. (a) General 3 × 3 convolution kernel. (b) Two-dilated 3 × 3 convolution kernel. (c) Four-dilated 3 × 3 convolution kernel.

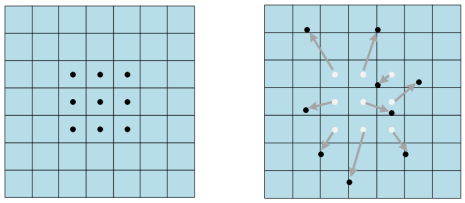
called feature maps. When setting a convolution kernel with a certain size, we will lose information on the border. Hence, padding is introduced to enlarge the input with zero value, which can adjust the size indirectly. Besides, to control the density of convolving, the stride is deployed. The larger the stride, the lower is the density. After convolution, feature maps consist of a large number of features that is prone to causing overfitting problem [21]. As a result, pooling [22] (a.k.a. downsampling) is proposed to obviate redundancy, including max pooling and average pooling. The procedure of a CNN is shown in Fig. 2.

Furthermore, in order for convolution kernels to perceive larger areas, dilated convolution [23] was proposed. A general 3 × 3 convolution kernel is shown in Fig. 3(a), and a two-dilated 3 × 3 convolution kernel and a four-dilated 3 × 3 convolution kernel are shown in Fig. 3(b) and (c). Note that there is an empty value (filling with zero) between each convolution kernel point. Even though the valid kernel points are still 3 × 3, a two-dilated convolution has a 7 × 7 receptive field, and a four-dilated convolution has a 15 × 15 receptive fields.

### A. Deformable Convolution

In general, the neural network's geometric structure is fixed, which makes networks not robust to geometric transformation. In computer vision, it is critical that models adapt to the object's geometric transformation after the deformation or the change of poses. We can overcome the geometric transformation by adding training data or using data augmentation, but these solutions require heavy workloads. Instead, we can use features or algorithms robust to geometric transformation and need to select prior knowledge. Nevertheless, these manually chosen features are often not generalizable.

Dai *et al.* [24] introduced two modules that can enhance the ability of CNN for geometric transformation. The first one is the deformable convolution module. As shown in Fig. 4, deformable convolution was proposed to handle the scenario that the shapes of objects are usually irregular. Deformable convolution can only focus on what they are interested in, making the feature maps representative. Dai *et al.* [24] used a parallel convolution layer to learn offsets and added it to the convolution kernel's original position to realize scale transformation, such as translation and rotation, and then



(a) (b)

Fig. 4. Comparison between general convolution kernel and deformable convolution kernel. (a) General $3 \times 3$ convolution kernel. (b) Deformable $3 \times 3$ convolution kernel.

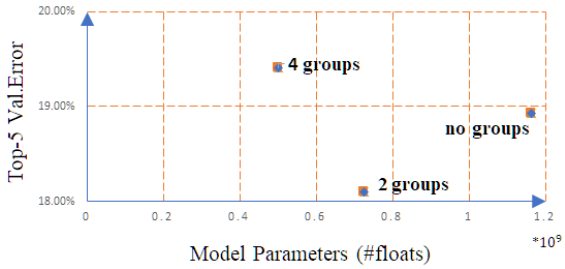



Fig. 5. Comparison of AlexNet with group convolution and without group convolution.

carried out convolution. These offsets adapt to the target object's scale transformation and increase the convolution kernel's receptive field. The second one is a deformable region of interest (ROI) pooling module. The deformable pooling module uses a full connection layer to learn offsets of features and then perform pooling.

Zhu *et al.* [25] found that although deformable convolution can adapt to geometric transformation and expand the receptive field by moving the convolution kernel unit, the training is affected by many features that are not related to the available domain. Therefore, it is necessary to enhance the deformable volume. Zhu *et al.* [25] proposed a deformable convNet v2. By expanding the use of deformable convolution in the network and adding a modulation mechanism to the deformable convolution, the network module can vary both the spatial distribution and the relative influence of its samples. In order to meet the hardware requirements, Dong *et al.* [26] combined depthwise convolution with deformable convolution. All the convolution layers to predict the offsets are replaced by depthwise convolution to further reduce the computational cost.

### *B. Group Convolution*

Group convolution was first applied to AlexNet to train deep CNNs with low GPU resource consumption. Replacing ordinary convolution with group convolution can build wider networks and reduce parameters. Moreover, group convolution can learn representations better and guarantee accuracy. As shown in Fig. 5, AlexNet without group convolution is less efficient and less accurate than the other two groups.

Why is group convolution better at learning features? As shown in Fig. 6, the filter bank of group convolution learns features in a sparse diagonal block structure on the channel dimension. Compared with the standard convolution in Fig. 7, it is less likely to lead to overfitting by reducing the number of parameters. Furthermore, operation analogous to regularization is used to learn more accurate and useful deep networks. As group convolution is more effective than the standard one, ResNeXt [27] overlaps smaller convolution blocks to approximate ResNet's convolution blocks, which can increase

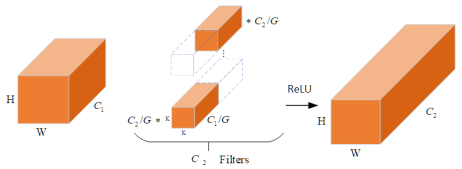



Fig. 6. Group Convolution Module. A convolution layer is divided into G groups of small filters, and the output of the packet convolution is composed of the production of the entire feature map.

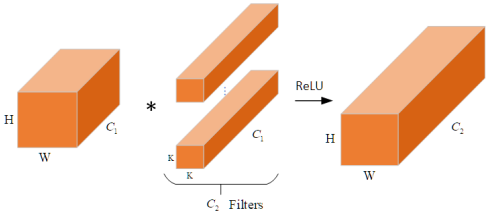



Fig. 7. Standard Convolution Module. The number of channels of the output feature map is the number of convolution kernels.

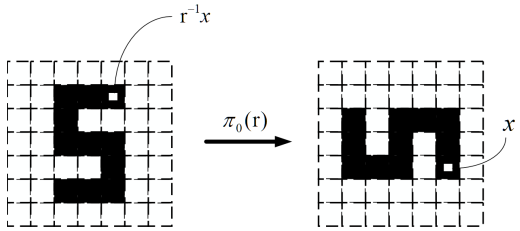



Fig. 8. Image uses the linear operator $\pi_0$ to rotate $r$.

the accuracy and reduce the model's complexity. To minimize the design burden, the smaller convolutional blocks used in ResNeXt have the same topology. The experimental results of ResNeXt also proved that increasing the number of stacking blocks is more effective than only increasing the width and depth of the network.

Huang *et al.* [28] divided group convolution training into several steps. The first step is the compression stage. After sparsity-inducing regularization training, the less essential filters with lower weights are pruned. The second step is the optimization stage. After every groups' weights are fixed, continuous training ensures that the same group's convolution shares the same sparsity pattern. However, this method of manually selecting group convolution still lacks superiority. Thus, Zhang *et al.* [29] proposed Groupable ConvNet to combine network architecture search (NAS) with group convolution, which can automatically learn the number of groups using an end-to-end approach.

### *C. Steerable Convolution*

In many visual tasks, it is crucial to enhance the equivariance of networks. We usually prefer networks to be equivariant, rather than invariant. As shown in Fig. 8, after rotating an image belonging to linear space, the invariant network no longer performs accurate recognition. More specifically, a face with abnormal facial features, for instance, in an abstract painting may still be recognized as a positive sample in an invariant convolution network. Therefore, we need CNNs to have equivariance to produce a predictable linear representation in input transformation. The filters adapt not only to position changes like a standard CNN but also to pose changes.

Cohen and Welling [30] advanced a general theory of manipulable representation and proposed steerable CNN that applies the theory to convolution operations.

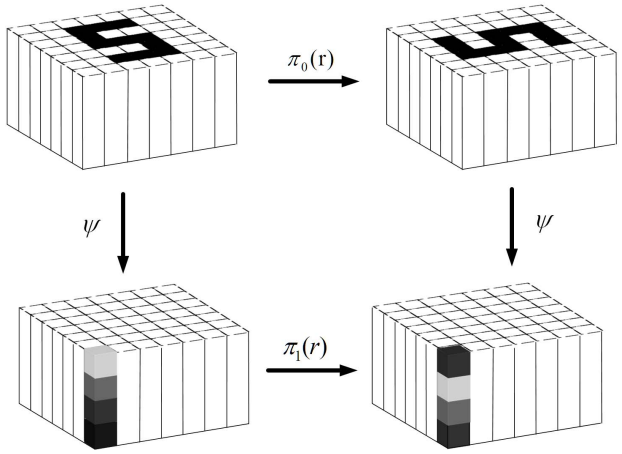



Fig. 9. $\psi$ a is the transformation with equivariance. Turn a list of features to a new position through $\pi_1$ and then use a permutation matrix equivalent to a rotation of 90° to cyclically exchange four channels.

As shown in Fig. 9, they introduced various forms of the linear transformation in a group. It helps to increase the flexibility of equivariant CNN and decouple groups' size and computational complexity. Besides, the steerable CNN can be easily extended to continuous settings, and evaluating steerable CNN for large and high-dimensional groups is an essential part of future work.

Based on steerable CNN, Weiler *et al.* [31] proposed the 3-D steerable CNN that is equivalent to rigid body motion, representing data in 3-D Euclidean space using scalar, vector, and tensor fields and mapping in these representations using equal-variant convolution. Besides, the Euclidean group E(2) [32] and its subgroups as kernel space's general solutions are given to adapt to the rotation and reflection of planar images.

### *D. Graph Convolution*

Graph neural network (GNN) is one of the popular topics in recent years. The earliest GNN based on fixed point theory proposed by Gori *et al.* [33] is used to solve the problem. They converted the graph to a 1-D matrix in the preprocessing stage leading to topological information loss. Scarselli *et al.* [34] extended the existing neural network method to process the graph domain data. They proposed a GNN model that can directly process more types of graphs, such as cyclic graphs, directed graphs, and undirected graphs. Because convolution neural networks cannot learn data in non-Euclidean space, the previous GNN model cannot combine with convolution. It becomes a bottleneck that GNNs need to break through. Bruna *et al.* [35] found that CNN can use the local translation invariance of the signal class in its domain, which is very useful in image and audio recognition tasks. They proposed two types of GNN structures. One is based upon a hierarchical clustering of the domain and the other is based upon the spectrum of the graph Laplacian. Kipf and Welling [36] proposed a graph convolution network (GCN). They motivate the choice of convolution architecture via a localized first-order approximation of spectral graph convolutions. The GCN model linearly expands the graph's number of edges and learns the hidden layer representation of the coded local graph structure and node features. The graph's convolution methods can be divided into two types: spatial domain convolution and frequency domain convolution.

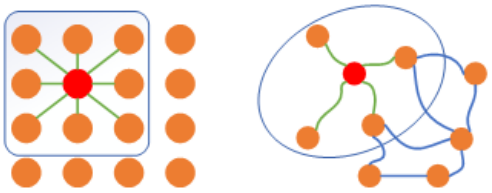


Fig. 10. Comparison between standard convolution and graph convolution on spatial domain.

Compared with convolution on the spatial domain in Fig. 10, convolution on the frequency domain has a factual theoretical basis for graph signal processing. GCN [36] is a method based on frequency domain convolution, which filters a convolution network and maps processed signals by Fourier transform to the frequency domain. From the spatial domain perspective, GCN is considered aggregating feature information from a node's neighborhood and gradually improved by finding alternative symmetric matrices. Adaptive GCN proposed by Li *et al.* [37] constructs a residual graph using a learnable distance function with two-node features as input. The adjacency matrix of the residual graph can learn unspecified hidden structural relationships. Zhuang and Ma [38] implemented a dual-GCN (DGCN) by parallelizing two graph convolution layers. By integrating outputs from dual-graph convolution layers, DGCN captures both local and global structural information without leveraging multiple graph convolution layers.

Traditional CNN convolves pixels on the image, while graph convolutions based on spatial domain define convolution by the spatial relations of nodes in graphs. As shown in Fig. 10, it convolves the representation of the central node with that of neighboring nodes and gets the updated representation of the central node. The node information of the spatial domain convolution essentially propagates along the edges. Micheli [55] proposed the first graph convolution neural network based on spatial domain, called neural network for graphs (NN4G). It performs convolutions by accumulating a neighbor node's information directly. To record each layer's features, NN4G [55] uses residual structure and skip connections.

### *E. Several Classic CNN Models*

Since AlexNet [11] was proposed, researchers have invented a variety of CNN models—they are deeper, wider, and lighter. Part of well-known models can be seen in Fig. 11 and Table I in the Supplementary Material. Due to the limitation of article length, this section aims to briefly overview several representative models and discusses the innovations of them to help readers draw lessons from the key points and have the ability to propose their own promising ideas.

## III. Discussion and Experimental Analysis

### *A. Activation Function*

*1) Discussion of Activation Function:* CNNs can harness different activation functions to express complex features. Similar to the function of the neuron model of the human brain, the activation function here is a unit that determines which information should be transmitted to the next neuron. Each neuron in the neural network accepts the output value of the neurons from the previous layer as input and passes the processed value to the next layer. In a multilayer neural network, there is a function between two layers. This function is called the activation function, whose structure is shown in Fig. 12.

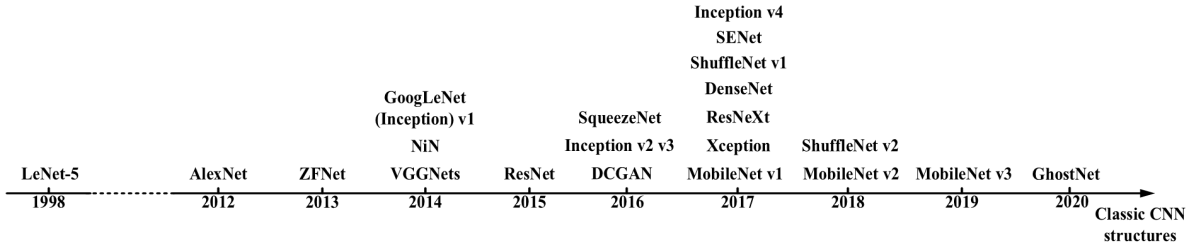



Fig. 11. Part of classic CNN models. NiN: network in network; ResNet: residual netwrok; DCGAN: deep convolutional generative adversarial network; SENet: squeeze-and-excitation network.

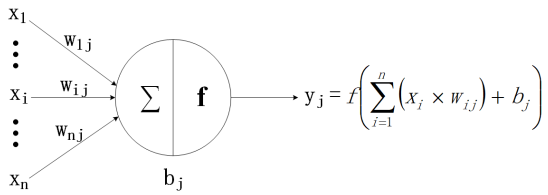



Fig. 12. General activation function structure.

In Fig. 12, $x_i$ represents the input feature; $n$ features are input to the neuron $j$ at the same time; $w_{ij}$ represents the weight value of the connection between the input feature $x_i$ and the neuron $j$; $b_j$ represents the internal state of the neuron $j$, which is the bias value; and $y_j$ is the output of the neuron $j$. $f(\cdot)$ is the activation function, which can be sigmoid function, tanh function [10], rectified linear unit (ReLU) [56], and so on [57].

If an activation function is not used or a linear function is used, the input of each layer will be a linear function of the output of the previous layer. In this case, He *et al.* [58] verify no matter how many layers the neural network has, the output is always a linear combination of the input, which means hidden layers have no effect. This situation is the primitive perceptron [2], [3], which has the limited learning ability. For this reason, the nonlinear functions are introduced as activation functions. Theoretically, deep neural networks with nonlinear activation functions can approximate any function, which greatly enhances the ability of neural networks to fit data.

In this section, we mainly focus on several frequently used activation functions. Sigmoid function is one of the most typical nonlinear activation functions with an overall $S$-shape [see Fig. 13(a)]. As $x$ value approaching 0, the gradient becomes steeper. The sigmoid function can map a real number to (0, 1); therefore, it can be used for binary classification problems. In addition, SENet [51] and MobileNet v3 [48] need to transform the output value to (0, 1) for attention mechanism, in which sigmoid is an appropriate way to implement.

Different from sigmoid, tanh function [10] [see Fig. 13(b)] can map a real number to $(-1, 1)$. Since the mean value of the output of tanh is 0, it can achieve a kind of normalization. This makes the next layer easier to learn.

In addition, ReLU [56] [see Fig. 13(c)] is another effective activation function. When $x$ is less than 0, its function value is 0; when $x$ is greater than or equal to 0, its function value is $x$ itself. Compared with the sigmoid function and tanh function, a significant advantage of using the ReLU function is that it can speed up learning. Sigmoid and tanh involve in exponential operation that requires division while computing derivatives, whereas the derivative of ReLU is a constant. Moreover, in the sigmoid and tanh function, if the value of $x$ is too large or too small, the gradient of the function is small, which can cause the function to converge slowly. However, when $x$ is less than 0, the derivative of ReLU is 0, and when $x$ is greater than 0, the derivative is 1; so, it can obtain an ideal convergence effect. AlexNet [11], the best model in ILSVRC-2012, uses ReLU as the activation function of the CNN-based model, which mitigates gradient vanishing problem when the network is deep and verifies that the use of ReLU surpasses sigmoid in deep networks.

Based on the discussion earlier, we can find that ReLU does not include the upper limit. In practice, we can set an upper limit, such as ReLU6 [59].

However, when $x$ is less than 0, the gradient of ReLU is 0. It means the back-propagated error will be multiplied by 0, resulting in no error being passed to the previous layer. In this scenario, the neurons can be regarded as inactivated or dead. Therefore, some improved versions are proposed. Leaky ReLU [see Fig. 13(d)] can reduce neuron inactivation. When $x$ is less than 0, the output of leaky ReLU is $x/a$, instead of zero, where $a$ is a fixed parameter in the range of $(1, +\infty)$.

Another variant of ReLU is PReLU [58] [see Fig. 13(e)]. Unlike leaky ReLU, the slope of the negative part of PReLU is based upon the data, not a predefined one. He *et al.* [58] reckon that PReLU is the key to surpassing the level of human classification on the ImageNet 2012 classification data set.

Exponential linear units (ELU) function [60] [see Fig. 13(f)] is another improved version of ReLU. Since ReLU is nonnegatively activated, the average value of its output is greater than 0. This problem leads to the offset of the next layer unit. ELU function has a negative value; therefore, the average value of its output is close to 0, making the rate of convergence faster than ReLU. However, the negative part is a curve, which demands lots of complicated derivatives.

In addition to sigmoid, tanh, and ReLU-likes, Google proposed another activation function called Swish [61] [see Fig. 13(g)]. Swish is $f(x) = x \cdot \sigma(\beta x)$, where $\sigma(z)$ represents the sigmoid function and $\beta$ is a hyperparameter. If $\beta = 0$, it becomes $f(x) = x/2$. As $\beta \to \infty$, it becomes similar to ReLU function. Although it has not been proved mathematically yet, experiments show that Swish tends to perform better than ReLU on deeper models across a number of challenging data sets.

Some activation functions, such as sigmoid and tanh, are prone to gradient saturation as the gradient changes little in a certain range. Inspired by swish, mish [62] [see Fig. 13(h)] was proposed by Misra as a novel self-regularized nonmonotonic activation function, which can be mathematically defined as $f(x) = x \cdot \tanh(\text{softplus}(x))$. Its nonboundary and smooth properties provide better gradient flow, accuracy, and generalization. Experimental results in this article show that Mish consistently outperforms Swish and ReLU across common standard architectures.

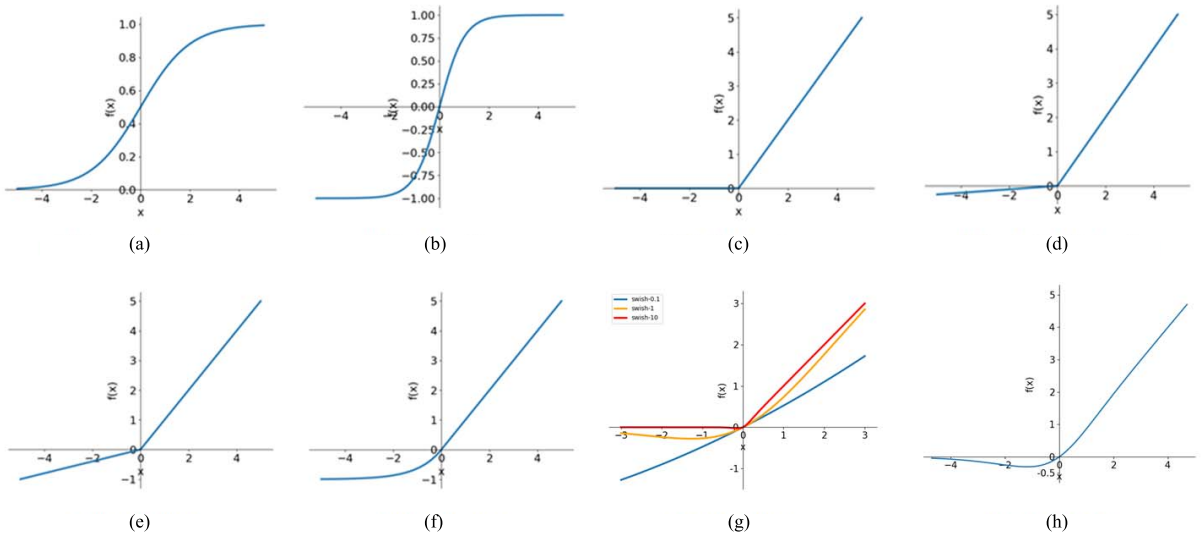



Fig. 13. Diagrams of sigmoid, tanh, ReLU, leaky ReLU, PReLU, ELU, swish, and mish. (a) Sigmoid function. (b) Tanh function. (c) ReLU function. (d) Leaky ReLU function. (e) PReLU function. (f) ELU function. (g) Swish functions. (h) Mish function.

*2) Experimental Evaluation:* To compare aforementioned activation functions, two classic CNN models, LeNet-5 [10] and VGG-16 [39], are tested on four benchmark data sets, including the Mixed National Institute of Standards and Technology (MNIST) database [10], fashion-MNIST [63], CIFAR-10 [64], and CIFAR-100 [64]. LeNet-5 is the first modern but relatively shallow CNN model. In the following experiments, we train LeNet-5 from scratch. VGG-16 is a deeper, larger, and more frequently used model. We conduct our experiments on the basis of a pretrained VGG-16 model on ImageNet [65] without the last three layers.

Both LeNet-5 and VGG-16 deploy softmax at the last layer for multiclassification. All experiments are tested on Intel Xeon E5-2640 v4 (X2), NVIDIA TITAN RTX (24 GB), Python 3.5.6, and Keras 2.2.4.

*a) MNIST and fashion-MNIST:* MNIST is a data set of handwritten digits consisting of ten categories, which has a training set of 60 000 examples and a test set of 10 000 examples. Each example is a $28 \times 28$ grayscale image, associated with a label from ten classes, from 0 to 9. Fashion-MNIST data set is a more complicated version of MNIST, sharing the same image size, structure, and split. These two data sets are trained on LeNet-5 and VGG-16, and results are exhibited in Table II in the Supplementary Material and Fig. 14. From the results, we can draw some meaningful conclusions.

1) Linear activation function indeed leads to the worst performance. Therefore, when building a deep neural network (more than one layer), we need to add a nonlinear function. Otherwise, multiple layers, theoretically, are equivalent to one layer.
2) Sigmoid converges slowest among all the activation functions. Usually, the sigmoid does not perform well. As a result, if we expect a fast convergence, sigmoid is not the best solution.
3) From the perspective of accuracy, ELU possesses the best accuracy, but it is only a little better than ReLU, leaky ReLU, and PReLU. In terms of training time, from Table II in the Supplementary Material, ELU is prone to consume more time than ReLU and leaky ReLU.
4) ReLU and leaky ReLU have better stability during training than PReLU and ELU.

*b) CIFAR-10 and CIFAR-100:* CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images data set [64], which are more complex than MNIST and Fashion-MNIST. CIFAR-10 data set consists of 60 000 $32 \times 32$ RGB images in ten classes, with 6000 images per class. The whole data set is divided into 50 000 training images and 10 000 test images, i.e., each class has 5000 training images and 1000 test images. CIFAR-100 is similar to the CIFAR-10, except it has 100 classes containing 600 images per class. And each class has 500 training images and 100 test images. Similarly, we evaluate LeNet-5 and VGG-16 with different activation functions on these two data sets. The results can be seen in Table II in the Supplementary Material and Fig. 15, from which we can draw some conclusions.

1) Tanh, PReLU, ELU, Swish, and Mish activation functions are more likely to bring about oscillation at the end of the training.
2) When training a deep CNN model with pretrained weights, it is hard to converge by using sigmoid and tanh activation functions.
3) The models trained by leaky ReLU and ELU have better accuracy than the others in the experiments. Nevertheless, ELU may make networks learn nothing sometimes. More often than not, leaky ReLU has a better performance in terms of accuracy and training speed.

*3) Rules of Thumb for Selection:*

1) For binary classification problems, the last layer can harness sigmoid; for multiclassification problems, the last layer can harness softmax.
2) Sigmoid and tanh functions sometimes should be avoided because of the gradient vanishment. Usually, in hidden layers, ReLU or leaky ReLU is a better choice.
3) If you have no idea about choosing activation functions, feel free to try ReLU or leaky ReLU.
4) If a lot of neurons are inactivated in the training process, please try to utilize leaky ReLU, PReLU, and so on.
5) The negative slope in leaky ReLU can be set to 0.02 to speed up training.

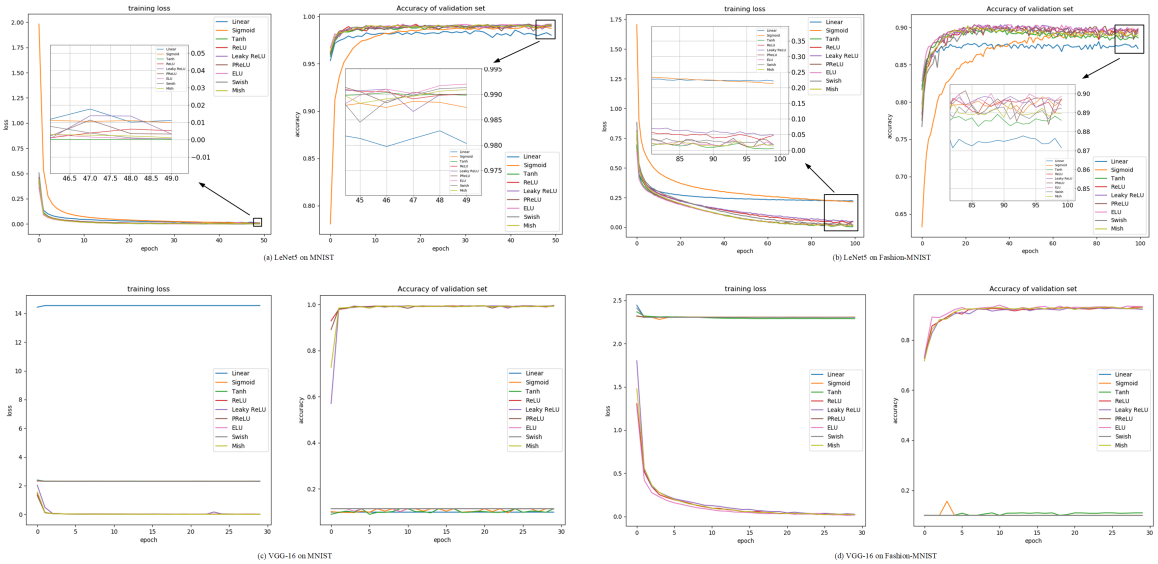



Fig. 14. Experimental results on nine activation functions, respectively. (a) Training loss and accuracy of validation set on MNIST trained by LeNet-5. (b) Training loss and accuracy of validation set on fashion-MNIST trained by LeNet-5. (c) Training loss and accuracy of validation set on MNIST trained by VGG-16. (d) Training loss and accuracy of validation set on Fashion-MNIST trained by VGG-16.

### *B. Loss Function*

The loss function or cost function is harnessed to calculate the distance between the predicted value and the actual value. The loss function is usually used as a learning criterion of the optimization problem. Loss function can be used with CNNs to deal with regression problems and classification problems, the goal of which is to minimize the loss function. Commonly used loss functions include mean absolute error (MAE), mean square error (MSE), cross entropy, and so on. Some of them are briefly described in Table I.

*1) Loss Function for Regression:* In CNNs, we are likely to use MAE or MSE to deal with regression problems.

MAE calculates the mean of the absolute error between the predicted value and the actual value; MSE calculates the mean of square error between them.

MAE is more robust to outliers than MSE because MSE calculates the square error of outliers. However, the result of MSE is derivable so that it can control the rate of update. The result of MAE is nonderivable, the update speed of which cannot be determined during optimization.

Therefore, if there are lots of outliers in the training set and they may have a negative impact on models, MAE is better than MSE. Otherwise, MSE should be considered.

*2) Loss Function for Classification:* In CNNs, there are many loss functions to handle when it comes to classification tasks.

The most typical one, called cross entropy loss, is used to evaluate the difference between the probability distribution obtained from the current training and the actual distribution. This function compares the predicted probability with the actual output value (0 or 1) in each class and calculates the penalty value based upon the distance from them. The penalty is logarithmic; therefore, the function provides a smaller score (0.1 or 0.2) for smaller differences and a bigger score (0.9 or 1.0) for larger differences.

Cross entropy loss is also called softmax loss, which indicates that it is always used in CNNs with a softmax layer. For example, AlexNet [11], inception v1 [40], and ResNet [44] uses cross entropy loss as the loss function in their original article, which helped them reach state-of-the-art results.

However, cross entropy loss has some flaws. Cross entropy loss only cares about the correctness of the classification, not the degree of compactness within the same class or the margin between different classes. Hence, many loss functions are proposed to solve this problem.

Contrastive loss [66] enlarges the distance between different categories and minimizes the distance within the same categories. It can be used in dimensionality reduction in CNNs. After dimensionality reduction, the two originally similar samples are still similar in the feature space, whereas the two originally different samples are still different. In addition, the contrastive loss is widely used with CNNs in face recognition. It was first used in SiameseNet [67] and later was deployed in DeepID2 [68], DeepID2+ [69], and DeepID3 [70].

After the contrastive loss, triplet loss was proposed by Schroff *et al.* [71]. In FaceNet [71], with which the CNN model can learn better face embeddings. The definition of the triplet loss function is based upon three images, i.e., anchor image, positive image, and negative image. The positive image and the anchor image are from the same person, whereas the negative image and the anchor image are from different people. Minimizing triplet loss is to make the distance between the anchor image and the positive one closer and make the distance between the anchor and the negative one further. Triplet loss is usually used with CNNs for fine-grained classification at the individual level, which requires models can distinguish different individuals from the same category. CNNs with triplet loss or its variants can be used in identification problems, such as face identification [71]–[73], person re-identification [74], [75], and vehicle re-identification [76].

Another one is the center loss [77], which is an improvement based upon cross entropy. The purpose of center loss is to focus on the uniformity of the distribution within the same class. In order to make it evenly distributed around the center of the class, center loss adds an additional constraint to minimize the intraclass difference. Center loss was used

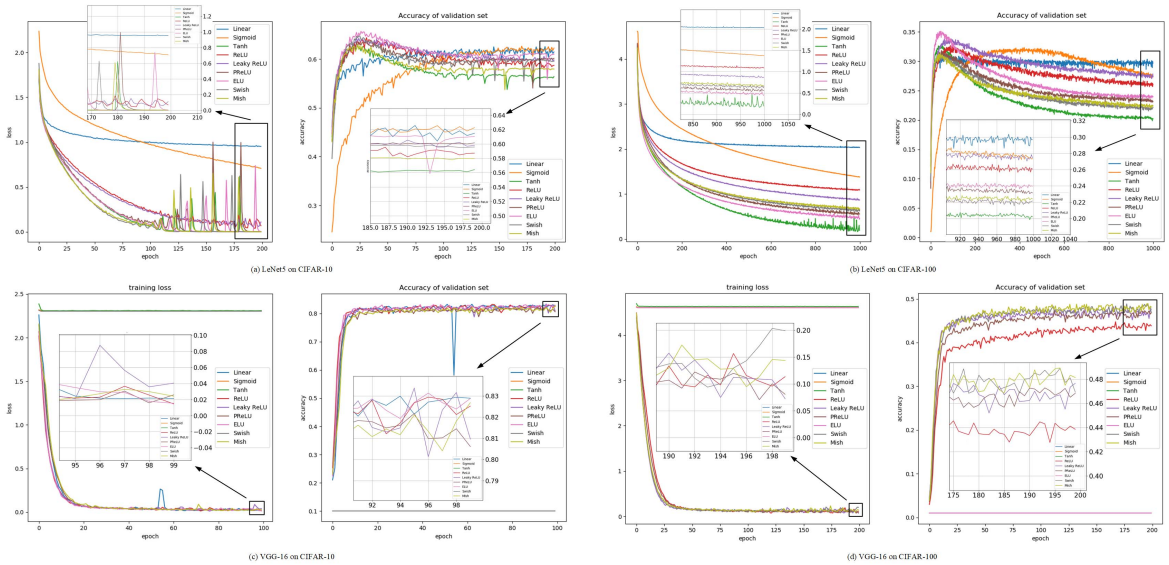



Fig. 15. Experimental results on nine activation functions, respectively. (a) Training loss and accuracy of the validation set on CIFAR-10 trained by LeNet-5. (b) Training loss and accuracy of the validation set on CIFAR-100 trained by LeNet-5. (c) Training loss and accuracy of the validation set on CIFAR-10 trained by VGG-16. (d) Training loss and accuracy of the validation set on CIFAR-100 trained by VGG-16.

TABLE I

COMMON LOSS FUNCTIONS FOR CNN

| Loss | Brief Description |
|---|---|
| Mean absolute error | Calculate the mean of absolute error of samples. |
| Mean square error | Calculate the mean of square error of samples. |
| Cross entropy loss | Calculate the difference between the probability distribution and the actual distribution. [11] [33] [37] |
| Contrastive loss | Enlarge the distance between different categories and minimize the distance within the same categories. [60] [61] [62] [63] |
| Triplet loss | Minimize the distance between anchor samples and positive samples, and enlarge the distance between anchor samples and negative samples. [64] [65] [66] [67] [68] |
| Center loss | Minimize intra-class distance. [70] [71] [72] [73] |
| Large-margin softmax loss | Focus on intra-class compression and inter-class separation. [74] [75] [76] |

with CNN in face recognition [77], image retrieval [78], person re-identification [79], speaker recognition [80], and so on.

Another variant of cross entropy is the large-margin softmax loss [81]. The purpose of it is also for intraclass compression and interclass separation. Large-margin softmax loss adds a margin between different classes and introduces the margin regularity through the angle of the constraint weight matrix. Large-margin softmax loss was used in face recognition [81], emotion recognition [82], speaker verification [83], and so on.

*3) Rules of Thumb for Selection:*

1) When using CNN models to deal with regression problems, we can choose L1 loss or L2 loss as the loss function.
2) When dealing with classification problems, we can select the rest of the loss functions.
3) Cross entropy loss is the most popular choice, usually appearing in CNN models with a softmax layer in the end.
4) If the compactness within the class or the margin between different classes is concerned, the improvements based upon cross entropy loss can be considered, such as center loss and large-margin softmax loss.
5) The selection of loss function in CNNs also depends on the application scenario. For example, when it comes to face recognition, contrastive loss and triplet loss are turned out to be the commonly used ones nowadays.

### C. Optimizer

In CNNs, we often need to optimize nonconvex functions. Mathematical methods require huge computing power; therefore, optimizers are used in the training process to minimize the loss function for getting optimal network parameters within an acceptable time. Common optimization algorithms include momentum, Root-mean-square prop (RMSprop), adaptive moment estimation (Adam), and so on.

*1) Gradient Descent:* There are three kinds of gradient descent methods that we can use to train our CNN models: batch gradient descent (BGD), stochastic gradient descent (SGD), and mini-BGD (MBGD).

The BGD indicates a whole batch of data needs to be calculated to get a gradient for each update, which can ensure convergence to the global optimum of the convex plane and the local optimum of the nonconvex plane. However, it is too slow to use BGD because the average gradient of the whole batch samples needs to be calculated. In addition, it can be tricky for data that is not suitable for in-memory calculation. Hence, BGD is hardly utilized in training CNN-based models in practice.

On the contrary, SGD only uses one sample for each update. It is apparent that the time of SGD for each update greatly less than BGD because only one sample's gradient is needed to calculate. In this case, SGD is suitable for online learning [84]. However, SGD is quickly updated with high variance, which will cause the objective function to oscillate severely. On the one hand, the oscillation of the calculation can make the gradient calculation jump out of the local optimum and

finally reach a better point; on the other hand, SGD may never converge because of endless oscillation.

In terms of convergence rate, assuming that the standard deviation of each sample for the real distribution is $\sigma$, the standard deviation of $n$ samples is $\sigma/\sqrt{n-1}$. When we use samples to estimate the gradient, one sample brings the standard deviation of $\sigma$, but using $n$ samples to estimate the gradient does not make the standard deviation decrease linearly, while the amount of calculation of n samples increases linearly. On the premise of the same amount of computation, the convergence rate of using the whole sample set is much slower than using a small number of samples. In other words, in order to converge to the same optimal point, when using the whole training set, although the number of iterations is small, the time of each iteration is long. Hence, the total cost of time is greater than using a small number of samples for multiple iterations.

Theoretically, it is true that the smaller the number of samples is, the faster the convergence rate will be when using a single-core CPU. However, when using GPU for training in engineering, due to the large number of cores and excellent parallel computing ability of GPU, it takes the same time to calculate a sample as to calculate dozens or even hundreds of samples. Therefore, in engineering practice, based on BGD and SGD, MBGD was proposed and frequently used. It combines the advantages of BGD and SGD, and the size of the batch depends on GPU's memory and computing core. MBGD uses a small batch of samples for each update so that it can not only perform gradient descent more efficiently than BGD but also reduce the variance, making the convergence more stable.

Among these three methods, MBGD is the most popular one. Lots of classic CNN models use it to train their networks in original articles, such as AlexNet [11], VGG [39], inception v2 [41], ResNet [44] and DenseNet [85]. It has also been leveraged in FaceNet [71], DeepID [68], and DeepID2 [86].

*2) Gradient Descent Optimization Algorithms:* On the basis of MBGD, a series of effective algorithms for optimization are proposed to accelerate the model training process. A proportion of them is presented as follows.

Qian [87] proposed momentum algorithm. It simulates physical momentum, using the exponentially weighted average of the gradient to update weights. If the gradient in one dimension is much larger than the gradient in another dimension, the learning process will become unbalanced. The momentum algorithm can prevent oscillations in one dimension, thereby achieving faster convergence. Some classic CNN models, such as VGG [39], inception v1 [42], and residual networks [44], use momentum in their original article.

However, for the momentum algorithm, blindly following gradient descent is a problem. Nesterov accelerated gradient (NAG) algorithm [88] gives the momentum algorithm predictability that makes it slow down before the slope becomes positive. By getting the approximate gradient of the next position, it can adjust the step size in advance. NAG has been used to train CNN-based models in many tasks [89]–[91].

The Adagrad algorithm [92] is another optimization algorithm based upon gradients. It can adapt the learning rate to parameters, performing smaller updates (i.e., a low learning rate) for frequent feature-related parameters and performing larger step updates (i.e., a high learning rate) for infrequent ones. Therefore, it is suitable for processing sparse data. One of the main advantages of Adagrad is that there is no need to adjust the learning rate manually. In most cases, we just use 0.01 as the default learning rate [53]. FaceNet [71] uses Adagrad as the optimizer in training.

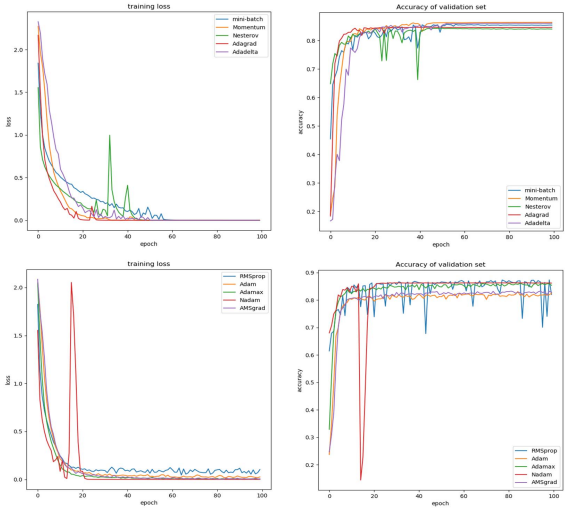



Fig. 16. Accuracy of validation set and training loss on CIFAR-10 trained by VGG-16 with ten different optimizers, respectively.

The Adadelta algorithm [93], an extension of the Adagrad, is designed to reduce its monotonically decreasing learning rate. It does not merely accumulate all squared gradients but sets a fixed size window to limit the number of accumulated squared gradients. At the same time, the sum of gradients is recursively defined as the decaying average of all previously squared gradients, rather than directly storing the previously squared gradients. Adadelta are leveraged in many tasks [94]–[96].

The RMSprop algorithm [97] is also designed to solve the problem of the radically diminishing learning rate in the Adagrad algorithm. MobileNet [46], inception v3 [42], and inception v4 [43] achieved their best models using RMSprop.

Another frequently used optimizer is adaptive moment estimation [98]. It is essentially an algorithm formed by combining the momentum with the RMSprop. Adam stores both the exponential decay average of the past square gradients like the Adadelta algorithm and the average exponential decay average of the past gradients like the momentum algorithm. The practice has proved that the Adam algorithm works well on many problems and is applicable to many different CNN structures [99]–[101].

The AdaMax algorithm [98] is a variant of Adam that makes the boundary range of the learning rate simpler, and it has been used to train CNN models [101], [102].

Nesterov-accelerated Adam estimation (Nadam) [103] is a combination of Adam and NAG. Nadam has a stronger constraint on the learning rate and a direct impact on the update of the gradient. Nadam is used in many tasks [104]–[106].

AMSGrad [107] is an improvement on Adagrad. The author of the AMSGrad algorithm found that there were errors in the update rules of the Adam algorithm, which caused it to fail to converge to the optimal in some cases. Therefore, the AMSGrad algorithm uses the maximum value of the past squared gradient instead of the original exponential average to update the parameters. AMSGrad has been used to train CNN in many tasks [108]–[110].

*3) Experimental Evaluation:* In the experiment, we tested ten kinds of optimizers—MBGD, momentum, Nesterov, Adagrad, Adadelta, RMSprop, Adam, Adamax, Nadam, and AMSgrad on the CIFAR-10 data set. The last nine optimizers

TABLE II
COMPARATIVE RESULTS OF DIFFERENT OPTIMIZERS

| Model | Last 3 layers | Dataset | Data aug | Loss | Activation func | Batch size | epochs | Optimizer | Val acc | Train time (s) |
|---|---|---|---|---|---|---|---|---|---|---|
| VGG-16 (pre-trained) | 512, 256, 10 | CIFAR-10 | - | Cross entropy | ReLU | 512 | 100 | MBGD | 85.70% | 926.24 |
| | | | | | | | | Momentum | 86.37% | 947.18 |
| | | | | | | | | Nesterov | 84.32% | 945.92 |
| | | | | | | | | Adagrad | 84.68% | 950.72 |
| | | | | | | | | Adadelta | 86.06% | 965.90 |
| | | | | | | | | RMSprop | **87.32%** | 959.33 |
| | | | | | | | | Adam | 83.09% | 953.46 |
| | | | | | | | | Adamax | 86.18% | 960.83 |
| | | | | | | | | Nadam | 86.26% | 968.72 |
| | | | | | | | | AMSgrad | 83.25% | 951.28 |

TABLE III
STRENGTHS AND WEAKNESSES OF SOME POPULAR OPTIMIZERS

| Optimizer | State Memory (bytes) | # Parameters | Strengths | Weaknesses |
|---|---|---|---|---|
| SGD | 0 | 1 | 1) Excellent generalization (after sufficient training) | 1) Prone to saddle points or local optima<br>2) Slow convergence rate<br>3) Sensitive to hyper-parameter initialization and learning rate selection |
| Momentum | 4n | 2 | 1) Accelerate in directions of steady descent<br>2) Not prone to falling into local optima | 1) Sensitive to hyper-parameter initialization and the values of learning rate and momentum |
| Adagrad | ∼4n | 1 | 1) Adjust learning rate adaptively<br>2) Convergence rate is faster on data with sparse features | 1) Worse generalization and likely to converge to sharp minimum<br>2) Gradient vanishing due to aggressive scaling |
| RMSprop | ∼4n | 3 | 1) Adjust learning rate adaptively but less aggressive than Adagrad<br>2) Convergence rate is faster on data with sparse features<br>3) Built in Momentum | 1) Worse generalization and likely to converge to sharp minimum |
| Adam | ∼8n | 3 | 1) Automatically decay the learning rate<br>2) Convergence rate is faster on data with sparse features<br>3) Combines the advantages of Momentum, Adagrad, and RMSprop | 1) Worse generalization and likely to converge to sharp minimum |

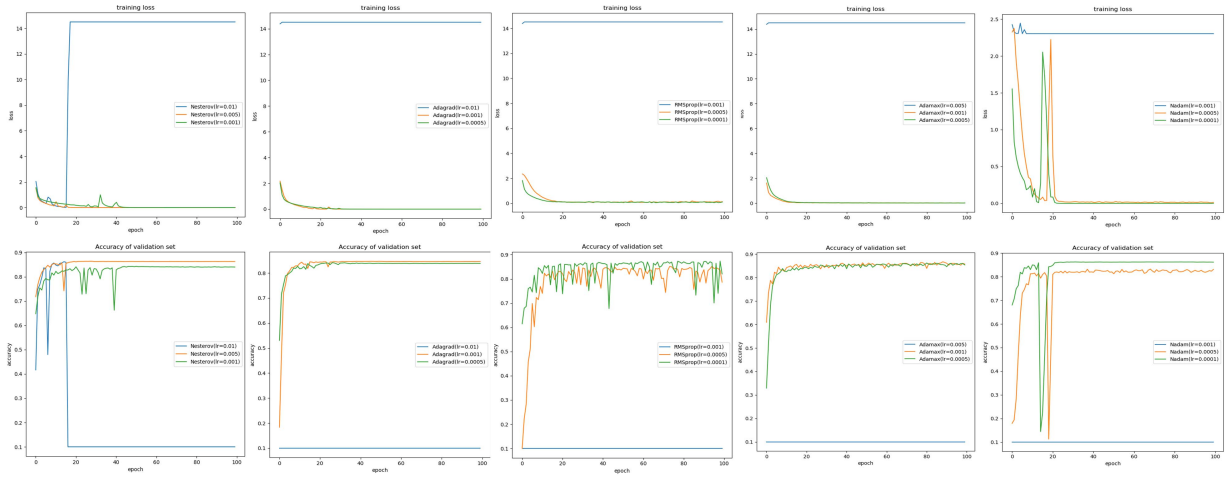



Fig. 17. Accuracy of validation set and training loss on CIFAR-10 trained by VGG-16 with Nesterov, Adagrad, RMSprop, Adamax, or Nadam optimizer with different learning rates, respectively.

are based upon mini batch. The format of the CIFAR-10 data set is the same as the experiment in Section III-A. We also do our experiments on the basis of a pretrained VGG-16 model without the last three layers on ImageNet [65]. The results can be seen in Table II and Fig. 16, from which we can draw some conclusions.

1) Almost all optimizers that we tested can make the CNN-based model converge at the end of the training.
2) The convergence rate of MBGD is slowest, even if it can converge at the end.
3) In the experiment, we find that Nesterov, Adagrad, RMSprop, Adamax, and Nadam oscillate and even cannot converge during training. Considering that initial values of parameters may affect experimental results, we change several random seeds to initialize parameters, but they do not converge as well. Hence, we conduct further experiments (see Fig. 17) and find that learning rate has a huge impact on convergence. It is a wise move to consider decreasing the learning rate when models do not converge.
4) Nesterov, RMSprop, and Nadam are likely to create oscillation, but it is this characteristic that may help models jump out of local optima.

*4) Rules of Thumb for Selection:*

1) MBGD should be used in order to make a tradeoff between BGD and SGD, i.e., the computing cost and the accuracy of each update.

TABLE IV
COMMON HYPERPARAMETER TUNING

| Hyper-parameter | Description | Strategy |
|---|---|---|
| learning rate | Learning rate refers to the step size of updating network weights. It can be constant, or variable. As mentioned in optimizer part, optimization algorithms determine different learning rates. In order to make the gradient descent perform better, the value of learning rate should be set in an appropriate range. If the learning rate is too small, the convergence speed will be slow; if it is too large, the parameters will oscillate back and forth on both sides of the optimal solution. | Usually, the initial value of parameters is far away from the optimal value. With iterations, the value will be closer to the optimal value. The basic idea of the adjustment strategy is to make learning rate gradually decrease with the training, i.e., using a relatively large learning rate at the beginning to speed up training, and then using a small learning rate to improve training stability, to avoid skipping the optimal value. |
| epoch | The epoch refers to the number of times that the whole training set is input to the neural network for training. When the gap of accuracy between training set and validation set is small, current epoch is considered appropriate. Otherwise, if the gap decreases, it means that the epoch is too small, resulting in underfitting; if the gap increases, the epoch is too large, resulting in overfitting. | If computing resources permit, set epoch as large as possible at first, and then stop training when training loss is stable or training accuracy is similar to validation accuracy. If computing resources are limited, train as many epochs as possible. Note that too small epoch will cause model underfitting, and too large epoch will cause model overfitting. |
| mini-batch size | Mini batch size is the number of samples sent to the model in each training. In the process of network optimization, too small batch size means that the number of samples input into the network is too small, i.e., not representative, and the noise increases correspondingly, which makes the network difficult to converge. Too large batch size makes the gradient direction nearly stable and easy to fall into the local optimum or saddle point. | In practice, mini-batch size is usually set to dozens to hundreds, depending on GPU's memory and computing core, and is usually set to the power of 2, because it can take full advantage of GPU performance. Moreover, when using the second-order optimization algorithm, such as Conjugate Gradient (CG) and L-BFGS, it is often necessary to use a large mini-batch size because if the first derivative is not well estimated, there will be a huge error in estimating the second derivative. |
| number of conv layers and conv kernels | Each conv layer usually contains different-level features. Shallow layers can detect edge features, local features, and other low-level features of the image, while deep layers can detect global features. Usually, networks with more layers and kernels have ability to represent more complex features, but meanwhile, are harder to train. | Unfortunately, to my best knowledge, there is no theoretical support or universal rule to determine the number of layers and kernels beforehand. Many proposed fancy structures depend on trial and error with the help of personal experience. |
| size of conv kernels | On the premise of the same receptive field, the smaller the convolution kernel is, the less the parameters and computational complexity are. Specifically, when the convolution kernel size is larger than 1, it can increase the receptive field. If the convolution kernel with even size is used, even if padding is symmetrically added, the size of input feature map and output feature map cannot be kept unchanged. | How to determine convolution kernel size of each layer has no solid theoretical basis. Researchers usually set it based on experience. Fortunately, many literatures have proved that several small convolution kernels can reach the receptive field of a large convolution kernel with fewer parameters. Hence, based on the left description, 3×3 kernel is frequently used. Besides,many novel kernels, such as 1×1, 1×n, and n×1 kernels, are worth trying. |

2) The performance of optimizers is closely related to data distribution. Based on the No Free Lunch Theorem, no single optimizer can surpass the others in all scenarios. It is a wise move to choose optimizers based on their characteristics. Some strengths and weaknesses of optimizers are shown in Table III, which can be leveraged as a reference while selection.
3) If excessive oscillation or divergence occurs, lowering the learning rate may be considerable.

### *D. Hyperparameter*

When constructing CNN, apart from selecting activation function, loss function, and optimizer, we have to tune many other hyperparameters, which greatly impact model performance. As we all know, there is no fixed set of hyperparameters that can guarantee an optimal solution all the time. Hence, a suite of experience and rules are significant while tuning hyperparameters. Table IV shows how some common hyperparameters affect the performance of models.

*1) Hyperparameter Priority:* When we train models, different hyperparameters have different tuning priorities. From our own experience, the priority of hyperparameters can be divided into four categories. The most significant hyperparameter is the learning rate, which means that it plays the most pivotal role in training and should be tuned first. The second tier includes acceleration value in the momentum optimizer, the number of convolution kernels, and mini-batch size. The third tier includes the number of layers, learning rate decay, and so on. The last tier includes the other hyperparameters of the optimizer. Note that this rule is not a panacea, and thus, researchers need to make specific analyses for different problems.

*2) Hyperparameter Search Strategy:* While tuning a model, it is easy to think of searching all hyperparameters for the best model. In detail, if two hyperparameters need to tuning and each has five candidates, 25 experiments have to be conducted, while with the increase of the number of hyperparameters, the number of experiments will explode exponentially. In addition, if we have no idea about which hyperparameter is more important (it is agnostic in most cases), then it is hard to determine which hyperparameter should be tuned first. Due to the limited computing resources, it is impractical to train numbers of giant models, each of which has millions of and even more trainable parameters. Hence, this kind of grid search algorithm is hard to implement, and accordingly, more efficient strategies or algorithms are strongly needed.

A common strategy is from coarse to fine. As mentioned earlier, pure grid search is challenging to conduct. An alternative way is leveraging coarse grid search to locate promising hyperparameter's approximate scope. Then, in this small area, fine searching, such as random search or small-step grid search, is harnessed to pinpoint the target.

Note that random search for hyperparameters does not mean to take values randomly and uniformly within the range, but to take random values with appropriate scale. For instance, when tuning the number of layers of CNN and the given range is from 4 to 6, then 4, 5, and 6 are all applicable. Nevertheless, given the tunable learning rate from 0.0001 to 1, if we take values evenly, there will be a 90% probability between 0.1 and 1, which is apparently unreasonable. Alternatively, if logarithmic coordinates are deployed, 0.0001 and 1 can be represented as $10^{-4}$ and $10^{0}$, respectively, so that we can choose a value from $-4$ to 0, denoted as $n$, using $10^{n}$ as the learning rate. Moreover, the same hyperparameter may possess different sensitivities in different ranges. For example, the momentum value in the momentum optimizer is more sensitive in the range of 0.9990 to 0.9995 than 0.9000 to

0.9005, and accordingly, more probability should be assigned to the former.

In order to efficiently find promising hyperparameters, a nonsmooth noisy problem, an increasing number of researchers and practitioners pay their attention to derivative-free optimization (DFO), a.k.a., gradient-free optimization or zeroth-order optimization. Unlike the aforementioned grid search or random search, these DFO algorithms iteratively leverage the information about the optimal solution to approximate the optimal solution. DFO algorithms can be roughly divided into genetic algorithms, particle swarm algorithms, simulated annealing algorithms, and so on. Some commonly used algorithms and toolkits include bayesian optimization [111], Nevergrad [112], and ZOOpt [113].

## IV. Application of CNN

CNN is one of the crucial concepts in the field of deep learning. In the era of big data, different from traditional approaches, CNN is able to harness a massive amount of data to achieve a promising result. Hence, there are lots of applications emerge. It can be used not only in the processing of 2-D images but also in 1-D and multidimensional scenarios.

### A. Applications of 1-D CNN

1-D CNN (1D CNN) typically leverages 1-D convolutional kernels to process 1-D data. It is effective when extracting the feature from a fixed-length segment in the whole data set, where the position of the feature does not matter. Therefore, 1D CNN can be applied to time series prediction and signal identification, for example.

*1) Time Series Prediction:* 1D CNN can be applied to time series prediction of data, such as electrocardiogram (ECG) time series, weather forecast, and traffic flow prediction. Erdenebayar *et al.* [114] proposed a method based on 1D CNN to predict atrial fibrillation using short-term ECG data automatically. Harbola and Coors [115] proposed 1-D Single CNN for predicting dominant wind speed and direction. Han *et al.* [116] applied 1D CNN on short-term highway traffic flow prediction. 1D CNN is used to extract spatial features of traffic flow, which is combined with temporal features to predict the traffic flow.

*2) Signal Identification:* Signal identification is to discriminate the input signal according to the feature that CNN learned from training data. It can be applied to ECG signal identification, structural damage identification, and system fault identification. Zhang *et al.* [117] proposed a multiresolution 1-D CNN structure to identify arrhythmia and other diseases based on ECG data. Abdeljaber *et al.* [118] proposed a direct damage identification method based on 1D CNN that can apply to the original environmental vibration signals. Abdeljaber *et al.* [119] designed a compact 1D CNN used in fault detection and severity identification of ball bearings.

### B. Applications of 2-D CNN

*1) Image Classification:* Image classification is the task of classifying an image into a class category. CNN represents a breakthrough in this field.

LeNet-5 [10] is regarded as the first application used in hand-written digits classification. AlexNet [11] made CNN-based classification approaches get off the ground. Then, Simonyan and Zisserman [39] emphasize the importance of depth, but these primitive CNNs are not more than ten layers. Afterward, deeper network structures emerged, such as GoogLeNet [40] and VGGNets [39], which significantly improve the accuracy in classification tasks.

He *et al.* [120] proposed the SPP-Net that inserts a pyramid pooling layer between the last convolution layer and the FC layer, making the size of different input images get the same size outputs. He *et al.* [44] proposed ResNet to solve the degradation problems and made it possible to train deeper neural networks. Chen *et al.* [121] proposed a double path network (DPN) for image classification by analyzing the similarities and differences between ResNet [44] and DenseNet [122]. DPN not only shares the same image features but also ensures the flexibility of structure feature extraction by a double path. Facebook opened the source code of ResNeXt-101 [27] and extended the number of layers of ResNeXt to 101, which achieved state-of-the-art results on ImageNet.

In addition, CNN can be deployed in medical image classification [123], [124], traffic scenes related classification [125], [126], and so on [127], [128]. Li *et al.* [123] designed a custom CNN with shallow convolution layers to the classification of interstitial lung disease. Jiang *et al.* [124] proposed a method based on SE-ResNet modules to classify breast cancer tissues. Bruno and Osorio [125] applied inception networks to the classification of traffic signal signs. Madan *et al.* [126] proposed a different preprocessing method to classify traffic signals.

*2) Object Detection:* Object detection is based on image classification. Systems not only need to identify which category the input image belongs to but also need to mark it with a bounding box. The development process of object detection based on deep learning is shown in Fig. 18. The approaches of object detection can be divided into one-stage approaches, such as you only look once (YOLO) [129]–[131], single shot multibox detector (SSD) [132], and CornerNet [133], [134], and two-stage approaches, such as R-CNN [135], fast R-CNN [136], and faster R-CNN [137].

In the two-stage object detection, the region proposals are selected in advance and then the objects are classified by CNN. Girshick *et al.* [135] used region proposal and CNN to replace the sliding window and manual feature extraction used in traditional object detection and designed the R-CNN framework, which made a breakthrough in object detection. Then, Girshick [136], summarizing the shortcomings of R-CNN [135] and drawing lessons from the SPP-Net [120], proposed fast R-CNN, which introduced the ROI pooling layer, making the network faster. Besides, fast R-CNN shares convolution features between object classification and bounding box regression. However, fast R-CNN still retains the selective search algorithm of R-CNN's region proposals. Ren *et al.* [137] proposed faster R-CNN, which adds the selection of region proposals to make it faster. An essential contribution of faster R-CNN is introducing an RPN network at the end of the convolutional layer. Lin *et al.* [138] added feature pyramid network (FPN) to faster R-CNN, where multiscale features can be fused through the feature pyramid in the forward process.

In one stage, the model directly returns the category probability and position coordinates of the objects. Redmon *et al.* [129] regarded object detection as a regression problem and proposed YOLO v1, which directly utilizes a single-neural network to predict bounding boxes and

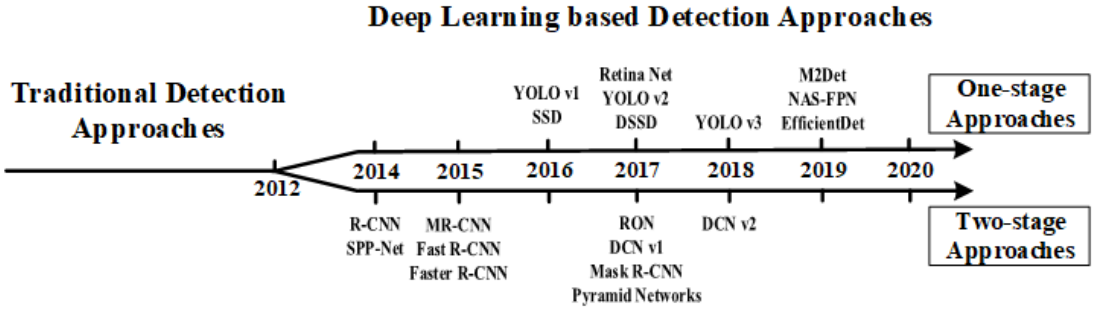



Fig. 18. Object detection milestones based on deep learning.

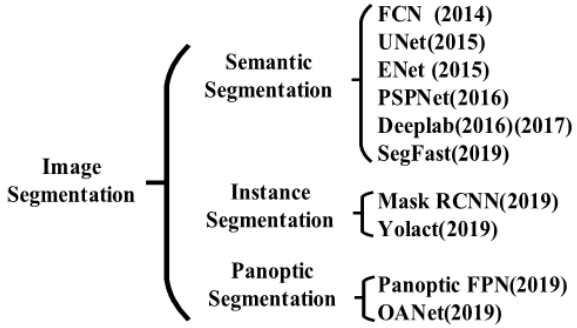



Fig. 19. Applications of CNN in image segmentation.

the category of objects. Afterward, YOLO v2 [130] proposed a new classification model, darknet-19, which includes 19 convolutional layers and five max-pooling layers. Batch normalization layers are added after each convolution layer, which is beneficial to stable training and fast convergence. YOLO v3 [131] was proposed to remove the max-pooling layers and the FC layers, using $1 \times 1$ and $3 \times 3$ convolution and shortcut connections. Besides, YOLO v3 borrows the idea from FPN to achieve multiscale feature fusion. For the benefits of the structure of YOLO v3, many classic networks replace the backend of it to achieve better results. All of the aforementioned approaches leverage anchor boxes to determine where objects are. Their performance hinges on the choice of anchor boxes, and a large number of hyperparameters are introduced. Therefore, Law *et al.* [133] proposed CornerNet, which abandons anchor boxes and directly predicts the top-left corner and bottom-right corner of bounding boxes of objects. In order to decide which two corners in different categories are paired with each other, an embedding vector is introduced. Then, CornerNet-Lite [134] optimized CornerNet in terms of detection speed and accuracy.

*3) Image Segmentation:* Image segmentation divides an image into different areas. It has to mark the boundaries of different semantic entities in an image. The image segmentation task completed by CNN is shown in Fig. 19.

Long *et al.* [139] proposed the concept of fully convolutional networks and applied CNN structures to image semantic segmentation for the first time. Ronneberger *et al.* [140] proposed U-Net, which has more multiscale features and has been applied to medical image segmentation. Besides, ENet [141], PSPNet [142], and so on [143], [144] were proposed to handle specific problems.

In terms of instance segmentation tasks, He *et al.* [145] proposed mask-RCNN that shares convolution features between two tasks through the cascade structure. Considering real time, Bolya *et al.* [146] based on RetinaNet [147] harnessed ResNet-101 and FPN to fuse multiscale features.

Panoptic segmentation was first proposed by Kirillov *et al.* [148]. They proposed panoramic FPN [138], which combines the FPN network with mask-RCNN to generate a branch of semantic segmentation. Liu *et al.* [149] proposed OANet that also introduced the FPN based on the mask-RCNN. The difference is that they designed an end-to-end network.

*4) Face Recognition:* Face recognition is a biometric identification technique based on the features of the human face. The development history of deep face recognition is shown in Fig. 20. DeepFace [150] and DeepID [68] achieved excellent results on the LFW [81] data set, surpassing humans for the first time in the unconstrained scenarios. Henceforth, deep learning-based approaches received much more attention. The process of DeepFace proposed by Taigman *et al.* [150] is detection, alignment, extraction, and classification. After detecting the face, using 3-D alignment generate a $152 \times 152$ image as the input of CNN. Taigman *et al.* [150] leveraged a Siamese network to train the model, which obtained state-of-the-art results. Unlike DeepFace, DeepID directly inputs two face images into CNN to extract feature vectors for classification. DeepID2 [86] introduces classification loss and verification loss. Based upon DeepID2 [86], DeepID2+ [69] adds the auxiliary loss between convolutional layers. DeepID3 [70] proposed two kinds of structures, which can be constructed by stacked convolutions of VGGNet or inception modules.

The aforementioned approaches harness the standard softmax loss function. More recently, improvements in face recognition are basically focused on the loss function. FaceNet [71] proposed by Google utilizes 22-layer CNN and 200 million pictures, including eight million people, to train a model. In order to learn more efficient embeddings, FaceNet replaces softmax with triplet loss. Besides, VGGFace [72] also deploys triplet loss to train the model. Besides, there are various loss functions harnessed to reach better results, such as L-softmax loss, SphereFace, ArcFace, and large margin cosine loss, which can be seen in Fig. 20 [81], [151]–[153].

### C. Application of Multidimensional CNN

In theory, CNN can be leveraged in data with any dimension. However, since high-dimensional data are hard to understand for humans, multidimensional CNN is not common in over three dimensions. Therefore, to better explain the key points, we take applications of 3-D CNN (3D CNN) for instance. However, it does not mean that higher dimensions are infeasible.

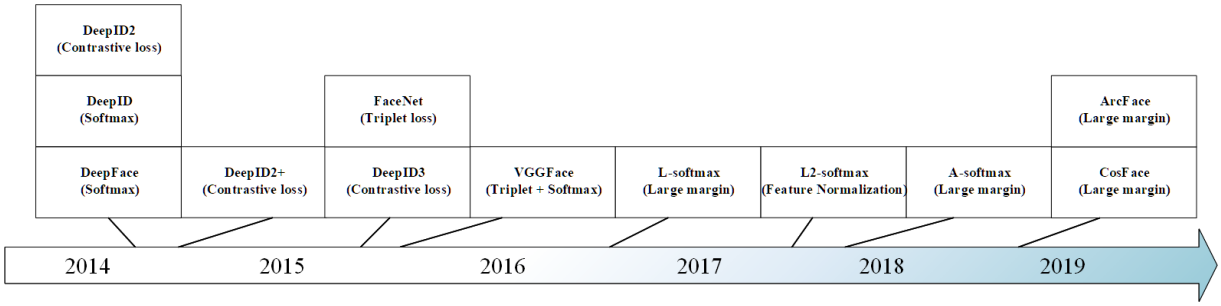



Fig. 20. Development history of deep face recognition.

*1) Human Action Recognition:* Human action recognition refers to the automatic recognition of human actions in videos by machines. As a kind of rich visual information, the action recognition method based on human body posture is used in many jobs. Cao *et al.* [154] utilized 3D CNN to extract features of joints instead of the whole body, and then these features are fed into a linear SVM for classification. Another approach [155] is to extend 2-D image convolutions to 3-D video convolutions to extract spatio–temporal patterns. Huang *et al.* [156] designed a 3D CNN structure to carry out sign language recognition that the 3-D convolutional layers automatically extract distinct spatial and temporal features from the video stream, which are input to the FC layer for classification. In addition, it is also possible to integrate the features extracted from 3D CNN and 2D CNN. Huang *et al.* [157] fused the 3-D convolutional pose stream with the 2-D convolutional appearance stream that providing more discriminative human action information.

*2) Object Recognition/Detection:* Wu *et al.* [158] proposed a generative 3D CNN of shape named 3-D ShapeNet, which can be applied to object detection of RGBD images. Maturana and Scherer [159] proposed VoxNet, which is a 3D CNN architecture that contains two 3-D convolutional layers, one pooling layer, and two FC layers. Song and Xiao [160] proposed a 3-D region proposal network to learn the geometric feature of objects and designed a joint object recognition network that fuses the output of VGGNet and the 3D CNN to learn 3-D bounding box and object classification jointly. Zhou and Tuzel [161] proposed a single end-to-end VoxelNet for point-cloud-based 3-D detection. VoxelNet contains feature learning layers, convolution middle layers, and RPN. Each convolution middle layer uses 3-D convolutions, batch normalization layers, and ReLU to aggregate voxelwise features. Pastor *et al.* [162] designed TactNet3D, harnessing tactile features to recognize objects.

In addition, high-dimensional images, such as X-rays and CT images, can be detected by 3D CNN. A lot of practitioners [163] are dedicated to these jobs.

## V. Prospects for CNN

With the rapid development of CNN, some methods are proposed to refine CNN, including model compression, security, NAS, and so on. In addition, CNNs have many obvious disadvantages, such as spatial information loss. New structures need to be introduced to handle these problems. Based on the points mentioned earlier, this section briefly introduces several promising trends of CNN.

### A. Model Compression

Over the past decade, CNNs have been designed with various modules, which helped CNN to reach high accuracy.

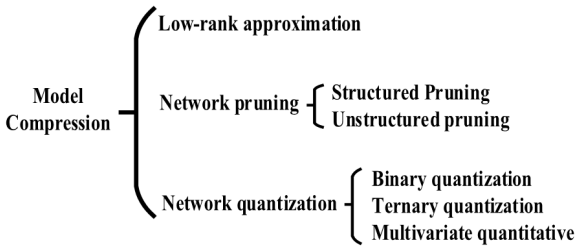



Fig. 21. Three directions of model compression.

However, high accuracy typically relies on extreme deep and wide architectures, making it challenging to deploy the models to embedded devices. Therefore, model compression is one of the possible ways to handle this problem, as shown in Fig. 21, including low-rank approximation, network pruning, and network quantization.

The low-rank approximation methods regard the weight matrix of the original network as the full-rank matrix and decompose it into a low-rank matrix to approximate the effect of the full-rank matrix. Jaderberg *et al.* [164] proposed a structure-independent method using cross channel redundancy or filter redundancy to reconstruct a low-rank filter. Sindhwani *et al.* [165] proposed a structural transformation network to learn a large class of structural parameter matrices characterized by low displacement rank through a unified framework.

Besides, according to networks' granularity, pruning can be divided into structured pruning and unstructured pruning. Han *et al.* [166] proposed deep pruning for CNN to prune networks by learning essential connections. However, the fine-grain pruning method increases the irregular sparsity of convolution kernels and computational cost. Therefore, many pruning methods focus on coarse granularity. Mao *et al.* [167] deemed that coarse pruning reaches a better compression ratio than fine-grained pruning. Through weighing the relationship between sparsity and precision, they provide some advice on how to ensure accuracy when structural pruning.

Network quantization is another way to reduce the computational cost, including binary quantization, ternary quantization, and multivariate quantization. Rastegari *et al.* [168] proposed binary-weight networks and XNOR-networks. The former makes the values of networks approximate to binary values, and the latter approximates convolutions by binary operations. Lin *et al.* [169] use a linear combination of multiple binary weight bases to approximate full-precision weights and multiple binary activations to reduce information loss, which suppresses the prediction accuracy degradation caused by previous binary CNN. Zhu *et al.* [170] proposed that


Fig. 22. Demonstration of fast adversarial example generation.

ternary quantization could alleviate the accuracy degradation. Besides, multivariate quantization is leveraged to represent weights by several values [171], [172].

As increasing networks use $1 \times 1$ convolution, a low-rank approximation is difficult to achieve model compression. Network pruning is a major practical way in model compression tasks. Binary quantization tremendously reduces the model size with the cost of losing accuracy. Hence, ternary quantization and multivariate quantization are harnessed to strike a proper balance between model size and accuracy.

### *B. Security of CNN*

There are many applications of CNN in daily life, including security identification system [173], [174], medical image identification [163], [175], [176], traffic sign recognition [177], and license plate recognition [178]. These applications are highly related to life and property security. Once models are disturbed or destroyed, the consequences will be severe. Therefore, the security of CNN should be attached with great importance. More precisely, researchers [179]–[182] proposed some methods to deceive CNN, resulting in a sharp drop in the accuracy. These methods can be classified into two categories: data poisoning and adversarial attacks.

Data poisoning indicates that poisoning the training data during the training phase. Poison refers to the insertion of noise into the training data. It is not easily distinguished at the image level and has no abnormalities found during the training process, whereas in test stages, the trained model has low accuracy. Furthermore, the noise can even be fine-tuned so that the model can identify certain targets incorrectly. Liao *et al.* [180] introduced that generated perturbation masks are injected into the training samples as a backdoor to deceive the model. Backdoor injection does not interfere with normal behaviors but stimulates the backdoor instance to misclassify specific targets. Liu *et al.* [183] proposed a method of fine-tuning and pruning that can effectively defend against the threat of backdoor injection. The combination of pruning and fine-tuning succeeds in suppressing, even eliminating the effects of the backdoors.

The adversarial attack is also one of the threats faced by deep neural networks. In Fig. 22, some noise is added to a normal image. Although naked eyes cannot distinguish the difference between two images, the CNN-based model cannot recognize them as the same. Goodfellow *et al.* [182] reckoned that the main factor for the vulnerability of neural networks is the linear characteristics, such as ReLU and Maxout. The cheap and analytical disturbance of these linear models could damage the neural network. Besides, they proposed a fast gradient notation to generate adversarial examples and found that many models misclassified these examples. Akhtar and Mian [184] listed three directions of defense against adversarial attacks, which are, respectively, improved on training examples, modified trained networks, and additional networks. First, for the training examples, adversarial examples can be utilized to enhance the robustness of models. Second, network architectures can be adjusted to ignore the noise. Last, additional networks can be used to help the backbone network against adversarial attacks.


Fig. 23. Procedure of NAS [185]. (a) NAS with agents. (b) NAS without agents.

### *C. Network Architecture Search*

NAS is another method to realize automatic machine learning of CNN. NAS constructs a search space through design choices, such as the number of kernels and skip connections. Besides, NAS finds a suitable optimizer to control the search process in the search space. As shown in Fig. 23, NAS could be divided into NAS with agents and without agents. Due to the high demands for NAS on computing resources, the integrated models consist of learned optimal convolutional layers in the small-scale data sets. Small-scale data sets are the agents that generate the overall model; therefore, this approach is the NAS with agents. The agentless NAS refers to learning the whole model directly on large-scale data sets.

Google Inc. [186] proposed a machine learning search algorithm that uses reinforcement learning to maximize the target network and implements an auto-built network on the CIFAR-10 data set. It achieved similar precision and speed to process networks with similar structures. Nevertheless, this approach is computationally expensive. Pham *et al.* [187] proposed efficient neural architecture search, which shares parameters among submodels and reduces resource requirements. Cai *et al.* [185] proposed ProxylessNAS, which is a path-level NAS method that has a model structure parameter layer at the end of the path and adds a binary gate before the output to reduce GPU utilization. It can directly learn architectures on the large-scale data set. In addition, there are many ways to reduce the search space of reinforcement learning. Tan *et al.* [188] designed the mobile neural architecture search to solve the CNN inferring delay problem. They introduced a decomposed hierarchical search space and performed the reinforcement learning structural search algorithm on this space. Ghiasi *et al.* [189] proposed NAS-FPN by applying NAS to feature pyramid structure search of object detection. They combined scalable search space with NAS to reduce search space. The scalable search space can cover all possible cross scale connections and generate multiscale feature representations.

### *D. Capsule Neural Network*

A lot of impressive applications of CNN are emerging in modern society, from simple cat–dog classifiers [190] to sophisticated autonomous vehicles [189], [191], [192]. However, is CNN a panacea? When does it not work?

CNN is not sensitive to slight changes of images, such as rotation, scaling, and symmetry, which has been demonstrated by Azulay and Weiss [193]. However, when trained models with Fig. 24(a), CNN cannot correctly recognize the similarity between Fig. 24(b), (c), or (d) and the former one, which is obvious to humans. This problem is caused by the architecture

TABLE V
SEVERAL HARDWARE IMPLEMENTATIONS FOR CNN

| Scheme | description |
|---|---|
| Convolution kernel oriented parallel pipeline convolver | Farabet et al. [194] make use of the inherent parallelism of CNN and multiple hardware multiply-accumulate units on FPGA. The whole system is implemented with a single FPGA and an external memory module, without extra parts. |
| Zynq SOC embedded scheme nn-X | It is a scalable and low-power coprocessor for real-time execution of CNN [195]. nn-X is implemented on programmable logic devices and comprises an array of configurable processing elements. |
| SIMD convolution engine | SIMD convolution engine [196] adopts many algorithm techniques, including partitioning large images, channel priority accumulation, considering both computation and memory bandwidth, etc., to realize SIMD computing architecture. In this scheme, the entire system is placed on a single FPGA chip, and DDR3 DRAM is used for external storage. |
| CPU server with FPGA acceleration board | This scheme [197] can adapt to the acceleration of cloud services, such as large-scale image recognition, online speech recognition, and has the characteristics of high scalability and high efficiency. |

of CNN. Therefore, in order to teach a CNN system to recognize different patterns of one object, a massive amount of data should be fed, making up for the flaw of CNN architectures with diverse data. However, labeled data is typically hard to obtain. Although some tricks like data augmentation can bring about some effects, the improvement is relatively limited.

The pooling layer is widely used in CNN for many advantages, but it ignores the relationship between the whole and the partitions. To effectively organize network structures and solve the problem of spatial information loss of traditional CNN, Sabour *et al.* [198] proposed capsule neural networks (CapsNets) where neurons on different layers focus on different entities or attributes so that they add neurons to focus on the same category or attribute, similar to the structure of a capsule. When CapsNet is activated, the pathway between capsules forms a tree structure composed of sparsely activated neurons. Each output of a capsule is a vector, the length of which represents the probability of the existence of an object. Therefore, the output features include the specific pose information of objects, which means that CapsNet has the ability to recognize the orientation. In addition, unsupervised CapsNet was created by Kosiorek *et al.* [199], called stacked capsule autoencoder (SCAE). SCAE consists of four parts: part capsule autoencoder (PCAE), object capsule autoencoder (OCAE), and the decoders of PCAE and OCAE. PCAE is a CNN with a top-down attention mechanism. It can identify the pose and existence of capsules of different parts. OCAE is used to implement inference. SCAE can predict the activations of CapsNet directly based on the pose and the existence. Some experiments have proved that CapsNet is able to reach state-of-the-art results. Although it did not achieve satisfactory results on complicated large-scale data sets, such as CIFAR-100 or ImageNet, we can see that it is potential.

### *E. Hardware Implementation*

As neural networks become larger and more complex, a large number of computing resources are needed for training and evaluation. The growth of CPU performance is increasingly slow; and therefore, many other devices are leveraged to speed up the process. Taking advantage of the outbreak of deep learning, NVIDIA has launched GPU-based acceleration solutions, including new processor architectures (such as Turing and Ampere), efficient acceleration libraries (such as cuBLAS and cuDNN), and so on. GPU clusters have been widely used as processing platforms in deep learning systems. In addition, a field programmable gate array (FPGA) is another substitute solution. Due to its flexible architecture, it can be optimized for specific models, which is not applicable in fixed architectures, such as CPUs and GPUs. Table V shows four FPGA-based acceleration schemes.


Fig. 24. Same cat in different ways. (a) Original cat image. (b) Largely rotated cat image. (c) Largely scaled cat image. (d) Full-symmetry cat image.

Specifically, the first two schemes are convolutional kernel-based convolvers, with the following disadvantages: when the size of the convolution kernel changes (for example, the size of the first convolutional layer of AlexNet is $11 \times 11$, and the following kernels are gradually reduced to $5 \times 5$ and $3 \times 3$), it cannot give full play to the parallel computing performance of hardware, resulting in a large amount of resource waste. Furthermore, with the emergence of more convolution kernels (such as $1 \times 1$, $1 \times n$, and $n \times 1$ convolution kernels in NiN), the waste of resources becomes more and more severe and even degrades to serial computing structure. Nevertheless, SIMD architecture is not related to the size of convolution kernels; therefore, the runtime computing resource utilization is higher.

Moreover, in the first three schemes, all calculations and control functions are performed on FPGA boards. Scheme 1 and scheme 3 use softcore CPU to realize the control function (scheme 1 utilizes softcore CPU with 32-bit PowerPC architecture, and scheme 3 utilizes softcore CPU with 32-bit MicroBlaze architecture), while scheme 2 leverages SoC embedded hardcore ARM Cortex A9 CPU to realize the control function. The defects of the three schemes are obvious: whether the softcore CPU or the hardcore CPU, the frequency is relatively low ($<$1 GHz), which cannot be used to the scenarios requiring cloud service acceleration and can only be used for embedded devices. Instead, the fourth scheme is the cloud service FPGA acceleration scheme, which has the characteristics of high scalability and high efficiency. In 2014, Microsoft announced the Catapult project, which successfully demonstrated that using FPGA in the data center accelerated bing ranking by nearly two times. Based on the foundation, Microsoft has developed a high throughput CNN FPGA accelerator, which achieves excellent performance with low power consumption.

Through experiments, some researchers [194]–[197], [200]–[202] found that FPGA has a great advantage over CPU and GPU in power efficiency. It can be predicted that with the further improvement of FPGA integration and main frequency,

the acceleration ability of CNN may gradually catch up with GPU and become the booster of the next outbreak of deep learning.

## VI. Conclusion

Due to the advantages of CNNs, such as local connection, weight sharing, and downsampling dimensionality reduction, CNN has been widely deployed in both research and industry projects. This article provides a detailed survey on CNN, including its motivation and some inspiring convolutions, classic networks, related functions, applications, and prospects.

First, we discuss the motivations of CNN, its basic building blocks and present some inspiring convolutions, including deformable convolution, group convolution, and so on.

Second, some representative CNN models are briefly discussed. From them, we obtain some guidelines for devising novel networks from the perspective of accuracy and speed. More specifically, in terms of accuracy, deeper and wider neural structures are able to learn better representation than shallow ones. Besides, residual connections can be leveraged to build extremely deep neural networks, which can increase the ability to handle complex tasks. In terms of speed, dimension reduction and low-rank approximation are convenient tools.

Third, we introduce activation functions, loss functions, optimizers, and hyperparameter selection for CNN. Through experimental analysis, several conclusions are reached. In addition, we offer some rules of thumb for the selection of these functions and hyperparameters.

Fourth, we discuss some typical applications of CNN. Different dimensional convolutions should be designed for various problems. Other than the most frequently used 2-D CNN used for image-related tasks, we present that 1-D and multidimensional CNN are harnessed in lots of scenarios as well.

Finally, even though convolutions possess many benefits and have been widely used, we reckon that they can be refined further in terms of model size, security, and NAS. Moreover, there are lots of problems that convolution is hard to handle, such as low generalization ability, lack of equivariance, and poor crowded-scene results, so that several promising directions are pointed. Finally, some hardware implementation schemes for CNN are discussed.

## References


[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[3] F. Rosenblatt, "Principles of neurodynamics. Perceptrons and the theory of brain mechanisms," Cornell Aeronaut. Lab., Buffalo, NY, USA, Tech. Rep., 1961.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[5] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 3, pp. 328–339, Mar. 1989.

[6] W. Zhang, "Shift-invariant pattern recognition neural network and its optical architecture," in *Proc. Annu. Conf. Jpn. Soc. Appl. Phys.*, 1988, p. 734, paper 6P-M-14.

[7] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[8] K. Aihara, T. Takabe, and M. Toyoda, "Chaotic neural networks," *Phys. Lett. A*, vol. 144, nos. 6–7, pp. 333–340, 1990.

[9] D. F. Specht, "A general regression neural network," *IEEE Trans. Neural Netw.*, vol. 2, no. 6, pp. 568–576, Nov. 1991.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[12] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2017, pp. 588–592.

[13] A. Dhillon and G. K. Verma, "Convolutional neural network: A review of models, methodologies and applications to object detection," *Prog. Artif. Intell.*, vol. 9, no. 2, pp. 85–112, Jun. 2020.

[14] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Comput.*, vol. 29, no. 9, pp. 2352–2449, Sep. 2017.

[15] Q. Liu *et al.*, "A review of image recognition with deep convolutional neural network," in *Proc. Int. Conf. Intell. Comput.* Cham, Switzerland: Springer, 2017, pp. 69–80.

[16] H. Ajmal, S. Rehman, U. Farooq, Q. U. Ain, F. Riaz, and A. Hassan, "Convolutional neural network based image segmentation: A review," *Proc. SPIE*, vol. 10649, Apr. 2018, Art. no. 106490N.

[17] T. Lindeberg, "Scale invariant feature transform," *Scholarpedia*, vol. 7, no. 5, p. 10491, May 2012.

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2005, pp. 886–893.

[19] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 2037–2041, Dec. 2006.

[20] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, Jan. 1962.

[21] D. M. Hawkins, "The problem of overfitting," *J. Chem. Inf. Comput. Sci.*, vol. 44, no. 1, pp. 1–12, Jan. 2004.

[22] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and Cooperation in Neural Nets*. Berlin, Germany: Springer, 1982, pp. 267–285.

[23] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2016, *arXiv:1511.07122*. [Online]. Available: https://arxiv.org/abs/1511.07122

[24] J. Dai *et al.*, "Deformable convolutional networks," 2017, *arXiv: 1703.06211*. [Online]. Available: https://arxiv.org/abs/1703.06211

[25] X. Zhu, H. Hu, S. Lin, and J. Dai, "Deformable ConvNets v2: More deformable, better results," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9308–9316.

[26] Z. Dong *et al.*, "CoDeNet: Efficient deployment of input-adaptive object detection on embedded FPGAs," 2020, *arXiv:2006.08357*. [Online]. Available: http://arxiv.org/abs/2006.08357

[27] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1492–1500.

[28] G. Huang, S. Liu, L. V. D. Maaten, and K. Q. Weinberger, "CondenseNet: An efficient DenseNet using learned group convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2752–2761.

[29] Z. Zhang *et al.*, "Differentiable learning-to-group channels via groupable convolutional neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3542–3551.

[30] T. S. Cohen and M. Welling, "Steerable CNNs," 2016, *arXiv:1612.08498*. [Online]. Available: http://arxiv.org/abs/1612.08498

[31] M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen, "3D steerable CNNs: Learning rotationally equivariant features in volumetric data," 2018, *arXiv:1807.02547*. [Online]. Available: http://arxiv.org/abs/1807.02547

[32] M. Weiler and G. Cesa, "General $E(2)$-equivariant steerable CNNs," 2019, *arXiv:1911.08251*. [Online]. Available: https://arxiv.org/abs/1911.08251

[33] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 2, Jul. 2005, pp. 729–734.

[34] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[35] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*. [Online]. Available: http://arxiv.org/abs/1312.6203

[36] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*. [Online]. Available: http://arxiv.org/abs/1609.02907

[37] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–8.

[38] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proc. World Wide Web Conf.*, 2018, pp. 499–508.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[40] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.

[43] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," in *Proc. AAAI Conf. Artif. Intell.*, 2017, vol. 31, no. 1, pp. 4278–4284.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[45] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015, *arXiv:1511.06434*. [Online]. Available: http://arxiv.org/abs/1511.06434

[46] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[47] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[48] A. Howard *et al.*, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[49] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.

[50] T.-J. Yang *et al.*, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 285–300.

[51] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7132–7141.

[52] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[53] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet v2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 116–131.

[54] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "GhostNet: More features from cheap operations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1580–1589.

[55] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Trans. Neural Netw.*, vol. 20, no. 3, pp. 498–511, Mar. 2009.

[56] V. Nair and G. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. ICML*, Haifa, Israel, 2010, pp. 807–814.

[57] M. T. Hagan, H. B. Demuth, and M. H. Beale, "Design neural network," Mech. Ind. Publishing Press, 2002, pp. 9–11.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[59] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on CIFAR-10," *Unpublished Manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[60] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," 2015, *arXiv:1511.07289*. [Online]. Available: http://arxiv.org/abs/1511.07289

[61] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*. [Online]. Available: http://arxiv.org/abs/1710.05941

[62] D. Misra, "Mish: A self regularized non-monotonic activation function," 2019, *arXiv:1908.08681*. [Online]. Available: http://arxiv.org/abs/1908.08681

[63] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: http://arxiv.org/abs/1708.07747

[64] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[66] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 2, Jun. 2006, pp. 1735–1742.

[67] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2005, pp. 539–546.

[68] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1891–1898.

[69] Y. Sun, X. Wang, and X. Tang, "Deeply learned face representations are sparse, selective, and robust," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 2892–2900.

[70] Y. Sun, D. Liang, X. Wang, and X. Tang, "DeepID3: Face recognition with very deep neural networks," 2015, *arXiv:1502.00873*. [Online]. Available: http://arxiv.org/abs/1502.00873

[71] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 815–823.

[72] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proc. Brit. Mach. Vis. Conf.*, vol. 1, no. 3, 2015, p. 6.

[73] B. Amos *et al.*, "OpenFace: A general-purpose face recognition library with mobile applications," School Comput. Sci., CMU, Pittsburgh, PA, USA, Tech. Rep. CM-CS-16-118, 2016.

[74] D. Cheng, Y. Gong, S. Zhou, J. Wang, and N. Zheng, "Person re-identification by multi-channel parts-based CNN with improved triplet loss function," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1335–1344.

[75] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," 2017, *arXiv:1703.07737*. [Online]. Available: http://arxiv.org/abs/1703.07737

[76] R. Kuma, E. Weill, F. Aghdasi, and P. Sriram, "Vehicle re-identification: An efficient baseline using triplet embedding," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–9.

[77] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 499–515.

[78] J. Yao, Y. Yu, Y. Deng, and C. Sun, "A feature learning approach for image retrieval," in *Proc. Int. Conf. Neural Inf. Process.*, 2017, pp. 405–412.

[79] H. Jin, X. Wang, S. Liao, and S. Z. Li, "Deep person re-identification with improved embedding and efficient training," in *Proc. IEEE Int. Joint Conf. Biometrics (IJCB)*, Oct. 2017, pp. 261–267.

[80] G. Wisniewksi, H. Bredin, G. Gelly, and C. Barras, "Combining speaker turn embedding and incremental structure prediction for low-latency speaker diarization," in *Proc. 18th Annu. Conf. Int. Speech Commun. Assoc. (ISCA)*, 2017, pp. 1–6.

[81] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks," in *Proc. ICML*, 2016, vol. 2, no. 3, p. 7.

[82] L. Tan, K. Zhang, K. Wang, X. Zeng, X. Peng, and Y. Qiao, "Group emotion recognition with individual facial emotion CNNs and global image based CNNs," in *Proc. 19th ACM Int. Conf. Multimodal Interact.*, Nov. 2017, pp. 549–552.

[83] Y. Liu, L. He, and J. Liu, "Large margin softmax loss for speaker verification," 2019, *arXiv:1904.03479*. [Online]. Available: http://arxiv.org/abs/1904.03479

[84] D. Saad, *On-Line Learning in Neural Networks*, no. 17. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[85] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[86] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," 2014, *arXiv:1406.4773*. [Online]. Available: http://arxiv.org/abs/1406.4773

[87] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, Jan. 1999.

[88] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$," in *Proc. Doklady Ussr*, vol. 269, 1983, pp. 543–547.

[89] W. Su, L. Chen, M. Wu, M. Zhou, Z. Liu, and W. Cao, "Nesterov accelerated gradient descent-based convolution neural network with dropout for facial expression recognition," in *Proc. 11th Asian Control Conf. (ASCC)*, Dec. 2017, pp. 1063–1068.

[90] A. L. Maas *et al.*, "Building DNN acoustic models for large vocabulary speech recognition," *Comput. Speech Lang.*, vol. 41, pp. 195–213, Jan. 2017.

[91] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand gesture recognition with 3D convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2015, pp. 1–7.

[92] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 7, pp. 1–39, 2011.

[93] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*. [Online]. Available: http://arxiv.org/abs/1212.5701

[94] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," 2015, *arXiv:1506.07503*. [Online]. Available: http://arxiv.org/abs/1506.07503

[95] T. Sercu, C. Puhrsch, B. Kingsbury, and Y. LeCun, "Very deep multilingual convolutional neural networks for LVCSR," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 4955–4959.

[96] A. Rakhlin, "Convolutional neural networks for sentence classification," GitHub, San Francisco, CA, USA, 2016. [Online]. Available: https://github.com/alexander-rakhlin/CNN-for-Sentence-Classification-in-Keras

[97] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," Dept. Comput. Sci., Toronto Univ., Toronto, ON, Canada, Tech. Rep., Jul. 2012.

[98] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[99] S. Sharma, R. Kiros, and R. Salakhutdinov, "Action recognition using visual attention," 2015, *arXiv:1511.04119*. [Online]. Available: http://arxiv.org/abs/1511.04119

[100] F. Korzeniowski and G. Widmer, "A fully convolutional deep auditory model for musical chord recognition," in *Proc. IEEE 26th Int. Workshop Mach. Learn. Signal Process. (MLSP)*, Sep. 2016, pp. 1–6.

[101] M. J. A. M. van Putten, S. Olbrich, and M. Arns, "Predicting sex from brain rhythms with deep learning," *Sci. Rep.*, vol. 8, no. 1, pp. 1–7, Dec. 2018.

[102] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 261–270.

[103] T. Dozat, "Incorporating Nesterov momentum into Adam," in *Proc. ICLR*, 2016, pp. 1–4.

[104] D. Q. Nguyen and K. Verspoor, "Convolutional neural networks for chemical-disease relation extraction are improved with character-based word embeddings," 2018, *arXiv:1805.10586*. [Online]. Available: http://arxiv.org/abs/1805.10586

[105] S. Maetschke, B. Antony, H. Ishikawa, G. Wollstein, J. Schuman, and R. Garnavi, "A feature agnostic approach for glaucoma detection in OCT volumes," *PLoS ONE*, vol. 14, no. 7, Jul. 2019, Art. no. e0219126.

[106] A. Schindler, T. Lidy, and A. Rauber, "Multi-temporal resolution convolutional neural networks for acoustic scene classification," 2018, *arXiv:1811.04419*. [Online]. Available: http://arxiv.org/abs/1811.04419

[107] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," 2019, *arXiv:1904.09237*. [Online]. Available: http://arxiv.org/abs/1904.09237

[108] M. Jahanifar, N. Z. Tajeddin, N. A. Koohbanani, A. Gooya, and N. Rajpoot, "Segmentation of skin lesions and their attributes using multi-scale convolutional neural networks and domain specific augmentations," 2018, *arXiv:1809.10243*. [Online]. Available: http://arxiv.org/abs/1809.10243

[109] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake news detection on social media using geometric deep learning," 2019, *arXiv:1902.06673*. [Online]. Available: http://arxiv.org/abs/1902.06673

[110] S. Liu *et al.*, "Decompose to manipulate: Manipulable object synthesis in 3D medical images with structured image decomposition," 2018, *arXiv:1812.01737*. [Online]. Available: http://arxiv.org/abs/1812.01737

[111] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," 2012, *arXiv:1206.2944*. [Online]. Available: http://arxiv.org/abs/1206.2944

[112] J. Rapin and O. Teytaud. (2018). *Nevergrad—A Gradient-Free Optimization Platform Version 0.2*. [Online]. Available: https://GitHub.com/FacebookResearch/Nevergrad

[113] Y.-R. Liu, Y.-Q. Hu, H. Qian, Y. Yu, and C. Qian, "ZOOpt: Toolbox for derivative-free optimization," 2017, *arXiv:1801.00329*. [Online]. Available: http://arxiv.org/abs/1801.00329

[114] U. Erdenebayar, H. Kim, J.-U. Park, D. Kang, and K.-J. Lee, "Automatic prediction of atrial fibrillation based on convolutional neural network using a short-term normal electrocardiogram signal," *J. Korean Med. Sci.*, vol. 34, no. 7, p. e64, 2019.

[115] S. Harbola and V. Coors, "One dimensional convolutional neural network architectures for wind prediction," *Energy Convers. Manage.*, vol. 195, pp. 70–75, Sep. 2019.

[116] D. Han, J. Chen, and J. Sun, "A parallel spatiotemporal deep learning network for highway traffic flow forecasting," *Int. J. Distrib. Sensor Netw.*, vol. 15, no. 2, pp. 1–12, 2019.

[117] Q. Zhang, D. Zhou, and X. Zeng, "HeartID: A multiresolution convolutional neural network for ECG-based biometric human identification in smart health applications," *IEEE Access*, vol. 5, pp. 11805–11816, 2017.

[118] O. Abdeljaber, O. Avci, S. Kiranyaz, M. Gabbouj, and D. J. Inman, "Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks," *J. Sound Vib.*, vol. 388, pp. 154–170, Feb. 2017.

[119] O. Abdeljaber, S. Sassi, O. Avci, S. Kiranyaz, A. A. Ibrahim, and M. Gabbouj, "Fault detection and severity identification of ball bearings by online condition monitoring," *IEEE Trans. Ind. Electron.*, vol. 66, no. 10, pp. 8136–8147, Oct. 2019.

[120] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.

[121] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," 2017, *arXiv:1707.01629*. [Online]. Available: http://arxiv.org/abs/1707.01629

[122] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "DenseNet: Implementing efficient ConvNet descriptor pyramids," 2014, *arXiv:1404.1869*. [Online]. Available: http://arxiv.org/abs/1404.1869

[123] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen, "Medical image classification with convolutional neural network," in *Proc. 13th Int. Conf. Control Automat. Robot. Vis. (ICARCV)*, Dec. 2014, pp. 844–848.

[124] Y. Jiang, L. Chen, H. Zhang, and X. Xiao, "Breast cancer histopathological image classification using convolutional neural networks with small SE-ResNet module," *PLoS ONE*, vol. 14, no. 3, Mar. 2019, Art. no. e0214587.

[125] D. R. Bruno and F. S. Osorio, "Image classification system based on deep learning applied to the recognition of traffic signs for intelligent robotic vehicle navigation purposes," in *Proc. Latin Amer. Robot. Symp. (LARS) Brazilian Symp. Robot. (SBR)*, Nov. 2017, pp. 1–6.

[126] R. Madan, D. Agrawal, S. Kowshik, H. Maheshwari, S. Agarwal, and D. Chakravarty, "Traffic sign classification using hybrid hog-surf features and convolutional neural networks," in *Proc. ICPRAM*, 2019, pp. 613–620.

[127] M. Zhang, W. Li, and Q. Du, "Diverse region-based CNN for hyperspectral image classification," *IEEE Trans. Image Process.*, vol. 27, no. 6, pp. 2623–2634, Jun. 2018.

[128] A. Sharma, X. Liu, X. Yang, and D. Shi, "A patch-based convolutional neural network for remote sensing image classification," *Neural Netw.*, vol. 95, pp. 19–28, Nov. 2017.

[129] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[130] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7263–7271.

[131] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*. [Online]. Available: http://arxiv.org/abs/1804.02767

[132] W. Liu *et al.*, "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.

[133] H. Law and J. Deng, "CornerNet: Detecting objects as paired keypoints," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 734–750.

[134] H. Law, Y. Teng, O. Russakovsky, and J. Deng, "CornerNet-lite: Efficient keypoint based object detection," 2019, *arXiv:1904.08900*. [Online]. Available: http://arxiv.org/abs/1904.08900

[135] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[136] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[137] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," 2015, *arXiv:1506.01497*. [Online]. Available: http://arxiv.org/abs/1506.01497

[138] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2117–2125.

[139] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[140] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Cham, Switzerland: Springer, 2015, pp. 234–241.

[141] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," 2016, *arXiv:1606.02147*. [Online]. Available: http://arxiv.org/abs/1606.02147

[142] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2881–2890.

[143] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018.

[144] A. Pal, S. Jaiswal, S. Ghosh, N. Das, and M. Nasipuri, "SegFast: A faster SqueezeNet based semantic image segmentation technique using depth-wise separable convolutions," in *Proc. 11th Indian Conf. Comput. Vis., Graph. Image Process.*, Dec. 2018, pp. 1–7.

[145] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2961–2969.

[146] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-time instance segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9157–9166.

[147] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.

[148] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, "Panoptic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9404–9413.

[149] H. Liu *et al.*, "An end-to-end network for panoptic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 6172–6181.

[150] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1701–1708.

[151] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "SphereFace: Deep hypersphere embedding for face recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 212–220.

[152] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive angular margin loss for deep face recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4690–4699.

[153] H. Wang *et al.*, "CosFace: Large margin cosine loss for deep face recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 5265–5274.

[154] C. Cao, Y. Zhang, C. Zhang, and H. Lu, "Action recognition with joints-pooled 3D deep convolutional descriptors," in *Proc. IJCAI*, vol. 1, 2016, p. 3.

[155] A. Stergiou and R. Poppe, "Spatio-temporal FAST 3D convolutions for human action recognition," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 183–190.

[156] J. Huang, W. Zhou, H. Li, and W. Li, "Attention-based 3D-CNNs for large-vocabulary sign language recognition," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 9, pp. 2822–2832, Sep. 2019.

[157] Y. Huang, S.-H. Lai, and S.-H. Tai, "Human action recognition based on temporal pose CNN and multi-dimensional fusion," in *Proc. Eur. Conf. Comput. Vis. (ECCV) Workshops*, 2018, pp. 1–15.

[158] Z. Wu *et al.*, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1912–1920.

[159] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2015, pp. 922–928.

[160] S. Song and J. Xiao, "Deep sliding shapes for amodal 3D object detection in RGB-D images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 808–816.

[161] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4490–4499.

[162] F. Pastor, J. M. Gandarias, A. J. García-Cerezo, and J. M. Gómez-de-Gabriel, "Using 3D convolutional neural networks for tactile object recognition with robotic palpation," *Sensors*, vol. 19, no. 24, p. 5356, Dec. 2019.

[163] K. Jnawali, M. R. Arbabshirani, N. Rao, and A. A. Patel, "Deep 3D convolution neural network for CT brain hemorrhage classification," *Proc. SPIE*, vol. 10575, Feb. 2018, Art. no. 105751C.

[164] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*. [Online]. Available: http://arxiv.org/abs/1405.3866

[165] V. Sindhwani, T. N. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," 2015, *arXiv:1510.01722*. [Online]. Available: http://arxiv.org/abs/1510.01722

[166] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[167] H. Mao *et al.*, "Exploring the regularity of sparse structure in convolutional neural networks," 2017, *arXiv:1705.08922*. [Online]. Available: http://arxiv.org/abs/1705.08922

[168] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[169] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," 2017, *arXiv:1711.11294*. [Online]. Available: http://arxiv.org/abs/1711.11294

[170] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*. [Online]. Available: http://arxiv.org/abs/1612.01064

[171] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," 2016, *arXiv:1612.01543*. [Online]. Available: http://arxiv.org/abs/1612.01543

[172] P. Micikevicius *et al.*, "Mixed precision training," 2017, *arXiv:1710.03740*. [Online]. Available: http://arxiv.org/abs/1710.03740

[173] M. Sajjad *et al.*, "CNN-based anti-spoofing two-tier multi-factor authentication system," *Pattern Recognit. Lett.*, vol. 126, pp. 123–131, Sep. 2019.

[174] K. S. Itqan, A. R. Syafeeza, F. G. Gong, N. Mustafa, Y. C. Wong, and M. M. Ibrahim, "User identification system based on finger-vein patterns using convolutional neural network," *ARPN J. Eng. Appl. Sci.*, vol. 11, no. 5, pp. 3316–3319, 2016.

[175] S. Hamidian, B. Sahiner, N. Petrick, and A. Pezeshk, "3D convolutional neural network for automatic detection of lung nodules in chest CT," *Proc. SPIE*, vol. 10134, Mar. 2017, Art. no. 1013409.

[176] H. Ke, D. Chen, X. Li, Y. Tang, T. Shah, and R. Ranjan, "Towards brain big data classification: Epileptic EEG identification with a lightweight VGGNet on global MIC," *IEEE Access*, vol. 6, pp. 14722–14733, 2018.

[177] A. Shustanov and P. Yakimov, "CNN design for real-time traffic sign recognition," *Procedia Eng.*, vol. 201, pp. 718–725, Jan. 2017.

[178] J. Špaňhel, J. Sochor, R. Juránek, A. Herout, L. Maršík, and P. Zemčík, "Holistic recognition of low quality license plates by CNN using track annotated data," in *Proc. 14th IEEE Int. Conf. Adv. Video Signal Based Surveill. (AVSS)*, Aug. 2017, pp. 1–6.

[179] T. Xie and Y. Li, "A gradient-based algorithm to deceive deep neural networks," in *Proc. Int. Conf. Neural Inf. Process.* Cham, Switzerland: Springer, 2019, pp. 57–65.

[180] C. Liao, H. Zhong, A. Squicciarini, S. Zhu, and D. Miller, "Backdoor embedding in convolutional neural network models via invisible perturbation," 2018, *arXiv:1808.10307*. [Online]. Available: http://arxiv.org/abs/1808.10307

[181] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 634–643.

[182] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*. [Online]. Available: http://arxiv.org/abs/1412.6572

[183] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2018, pp. 273–294.

[184] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14410–14430, 2018.
[185] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*. [Online]. Available: http://arxiv.org/abs/1812.00332
[186] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*. [Online]. Available: http://arxiv.org/abs/1611.01578
[187] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.
[188] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2820–2828.
[189] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "NAS-FPN: Learning scalable feature pyramid architecture for object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7036–7045.
[190] T. Jajodia and P. Garg, "Image classification–cat and dog images," *Image*, vol. 6, no. 23, pp. 570–572, 2019.
[191] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Model predictive control with a CNN cost model," 2017, *arXiv:1707.05303*. [Online]. Available: http://arxiv.org/abs/1707.05303
[192] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, "Object classification using CNN-based fusion of vision and LIDAR in autonomous vehicle environment," *IEEE Trans. Ind. Informat.*, vol. 14, no. 9, pp. 4224–4231, Sep. 2018.
[193] A. Azulay and Y. Weiss, "Why do deep convolutional networks generalize so poorly to small image transformations?" 2018, *arXiv:1805.12177*. [Online]. Available: http://arxiv.org/abs/1805.12177
[194] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 32–37.
[195] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2014, pp. 682–687.
[196] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2015, pp. 161–170.
[197] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," Microsoft Res., Redmond, WA, USA, White Paper, Feb. 2015, pp. 1–4, vol. 2, no. 11.
[198] S. Sabour, N. Frosst, and G. E Hinton, "Dynamic routing between capsules," 2017, *arXiv:1710.09829*. [Online]. Available: http://arxiv.org/abs/1710.09829
[199] A. R. Kosiorek, S. Sabour, Y. W. Teh, and G. E. Hinton, "Stacked capsule autoencoders," 2019, *arXiv:1906.06818*. [Online]. Available: http://arxiv.org/abs/1906.06818
[200] W. Zhao *et al.*, "F-CNN: An FPGA-based framework for training convolutional neural networks," in *Proc. IEEE 27th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2016, pp. 107–114.
[201] K. Guo *et al.*, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.
[202] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2016, pp. 326–331.

**Zewen Li** received the B.S. degree in computer science from Hohai University, Nanjing, China, in 2020.

He is enjoying a gap year between undergraduate and graduate school. His current research interests include computer vision, reinforcement learning, and distributed machine learning.

**Fan Liu** (Member, IEEE) received the B.S. and Ph.D. degrees from the Nanjing University of Science and Technology, Nanjing, China, in 2009 and 2015, respectively.

In 2008, he was a Researcher with Ajou University, Suwon, South Korea. In 2014, he was with Microsoft Research Asia, Beijing, China. He is currently a Professor with Hohai University, Nanjing. His research interests include computer vision, pattern recognition, and machine learning.

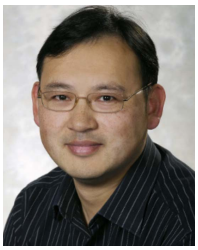
Dr. Liu also serves as a Reviewer for the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the *ACM Transactions on Intelligent Systems and Technology*, *Information Sciences*, *Neurocomputing*, and *Pattern Analysis and Applications*.

**Wenjie Yang** is currently pursuing the bachelor's degree in computer science with Hohai University, Nanjing, China.

His current research interests include object detection, face recognition, and reinforcement learning.

**Shouheng Peng** is currently pursuing the bachelor's degree with Hohai University, Nanjing, China.

His current research interests include computer vision, deep learning, and image processing.



**Jun Zhou** (Senior Member, IEEE) received the B.S. degree in computer science and the B.E. degree in international business from the Nanjing University of Science and Technology, Nanjing, China, in 1996 and 1998, respectively, the M.S. degree in computer science from Concordia University, Montreal, QC, Canada, in 2002, and the Ph.D. degree in computing science from the University of Alberta, Edmonton, AB, Canada, in 2006.

He was a Research Fellow with the Research School of Computer Science, The Australian National University, Canberra, ACT, Australia, and a Researcher with the Canberra Research Laboratory, National Information and Communications Technology Australia, Canberra. In 2012, he joined the School of Information and Communication Technology, Griffith University, Nathan, QLD, Australia, where he is currently an Associate Professor. His research interests include pattern recognition, computer vision, and spectral imaging and their applications in remote sensing and environmental informatics.