Chapter 10
Analyzing the Effect of Optimization Strategies in Deep Convolutional Neural Network

S. Akila Agnes and J. Anitha

Abstract Deep convolutional neural network (DCNN) is a powerful model for learning significant data at multiple levels of abstraction form an input image. However, training DCNN is often complicated because of parameter initialization, overfitting and convergence problems. Hence this work has been targeted to overcome the challenges of training DCNN with an optimized model. This chapter describes a deep learning framework for image classification with cifar-10 dataset. The model contains a set of convolutional layers with rectified linear unit activation function, max-pooling layers, and a fully-connected layer with softmax activation function. This model learns the features automatically and classifies the image without using the hand-crafted image based features. In this investigation, various optimizers have been applied in gradient descent technique for minimizing the loss function. Model with Adam optimizer constantly minimizes the objective function compared with other standard optimizers such as momentum, Rmsprop, and Adadelta. Dropout and batch normalization techniques are adapted to improve the model performance further by avoiding overfitting. Dropout function deactivates the insignificant node form the model after every epoch. The initialization of a large number of parameters in DCNN is regularized by batch normalization. Results obtained from the proposed model shows that batch normalization with dropout significantly improves the accuracy of the model with the tradeoff of computational complexity.

Keywords Batch normalization · Convolutional neural network
Dropout · Image classification · Optimization strategies

S. Akila Agnes · J. Anitha (✉)
Department of Computer Sciences Technology, Karunya Institute
of Technology and Sciences, Coimbatore, India
e-mail: anitha_j@karunya.edu

S. Akila Agnes
e-mail: akilaagnes@karunya.edu

J. Hemanth and V. E. Balas (eds.), *Nature Inspired Optimization Techniques for Image Processing Applications*, Intelligent Systems Reference Library 150,
https://doi.org/10.1007/978-3-319-96002-9_10

10.1 Introduction

Object classification plays a significant role in the area of computer vision. The goal of this process is to classify the objects into different categories, in the field of robotics to any intelligent systems. It is applied in various application domains such as medical imaging, vehicle tracking, industrial visual inspection, robot tracking, biometric systems and image remote sensing. The classification system examines the numerical properties of different image features and classifies them into different categories. It consists of two stages including training and testing. In training stage, the significant features of the input images are used to train the classification system against the target class. In testing stage, the classifier predicts the class for the input image.

The plethora of image classification methods has been proposed in the literature [1]. Various Machine Learning (ML) approaches such as Artificial Neural Network (ANN), Decision Tree Classifier, Support Vector Machine (SVM), and Expert System have been employed in the field of computer vision that label the input images to the desired category. The supervised learning algorithms enable the computer to learn on its own from the available dataset with labels and make predictions for the given data.

The efficiency of the machine learning system relies on the design of several handcrafted features extracted from the images. Despite various object classification algorithms and systems are introduced, there lacks a general and complete solution for recent challenges. New computational models such as Deep Learning (DL) models motivate the researchers to move towards the Artificial Intelligence. Deep learning has been evolved in 2006 with Deep Belief Networks (DBNs) [2] as a part of a machine learning algorithm that exploits many layers of non-linear information processing for pattern analysis and classification [3].

Supervised deep networks employ with labeled information and classify the input data in these labels. They exemplify the most common form of ML, deep or not [4]. This network is more flexible to build, more appropriate for end-to-end learning of complex systems [5] and more capable to train and test. It can be categorized into linear supervised deep method (e.g. Deep Neural Networks with linear activation functions) and non-linear supervised method (e.g. Deep Stacking Networks, Recurrent Neural Networks and Convolutional Neural Networks).

Conventional Neural Network (CNN) is the kind of DL which has been used in various applications of computer vision [6–13], especially for the classification of large sets of images. The performance of deep CNN is highly associated with the number of layers. It also has millions of parameters to tune with, which requires a large number of training samples. First Convolutional Neural Network is introduced by LeCun et al., in 1998 [8], has been the mainstream architecture in the neuronal network family for image classification tasks. Naturally, a CNN is specialized to learn useful local correlations and associate features in low level layers that support higher order learning. Further using Fully Connected (FC) layers in a general feed-forward neural network, CNN also depends on several convolutional and pooling layers before FC layers.

AlexNet [14] and VGG [15] networks have achieved better performance on image classification using deeper convolutional neural networks. Recently many researches have been moving in the field of deep networks. The advantage of deep CNN in images classification is that the entire model is trained end-to-end, from raw pixels to specific categories, which removes the requirement of handcrafted feature extraction.

The popular deep CNN architecture [14] composed of five convolutional layers and three fully-connected layers with a final soft-max classifier, and contains more than 60 million parameters. Some deeper networks, such as models with 16 and 19 hidden layers [15], 22 hidden layers [16] have attained better performance with more number of parameters. However, training deep CNN has several difficulties including vanishing gradients and overfitting [17]. This can be resolved by training a deeper CNN with well-designed architecture, initialization strategies, better optimizers and transfer learning.

As the gradient is back propagated through the network only a few blocks that learn suitable representations and many blocks contribute very little information towards the final goal. This problem is called as diminishing feature. This is solved by a methodology called dropout that disables the corresponding residual blocks during training [18]. Dropout methodology has been first introduced by Srivastava et al. [19] and adopted in many successful architectures [14, 15]. Mostly this is applied to top layers that had a large number of parameters to prevent feature overfitting. Another methodology named batch normalization [20] has been introduced to reduce the internal covariate shift in neural network activations by normalizing them to have specific distribution. This can also works as a regularizer and the researchers experimentally show that a network with batch normalization achieves better accuracy than a network with dropout. Directly learning so many parameters from only thousands of training samples will result in serious overfitting even though the overfitting preventing technique is applied. Therefore, there is a challenge on how to make the deep CNN that fit small dataset while keeping the similar performance as on large-scale dataset.

As a popular benchmark in this field, the cifar-10 database [21] is frequently used to evaluate the performance of classification algorithms. Krizhevsky [22] has carried a classification task on the cifar-10 dataset using a multinomial regression model. This uses all the layers and a single hidden layer that resulted in an overall accuracy of 64.84%. Liu and Deng [23] has proposed a modified VGG-16 network and achieved 8.45% error rate on CIFAR-10 without severe overfitting.

This chapter presents a deep CNN (DCNN) architecture to classify the images in the cifar-10 dataset. The presented architecture overcomes the problems in gradient descents (such as vanishing gradients and overfitting) by integrating suitable layers, optimizers, drop out and batch normalization strategies. The architecture uses the Adam optimizer as an efficient optimizer for cifar-10 dataset classification. The suitable optimizer is selected based on the analysis of different optimization strategies, which aim to minimize the objective function. Further, the effect of dropout and batch normalization also evaluated in the presented architecture. The experimental results show that the presented architecture significantly decreases the loss function with improved validation accuracy.

The rest of the chapter is organized as follows: Sect. 10.2 describes the general CNN architecture and its specifications. Section 10.3 deals with the proposed deep CNN architecture for cifar-10 dataset classification. Results and discussions are reported in Sect. 10.4. Finally, Sect. 10.5 presents the conclusion.

10.2 CNN Architecture

Convolution neural network is a back propagation neural network that works on images. CNN architecture has set of convolutional layers followed by fully connected layers and a final softmax layer that makes predictions. CNN layers learn the parameters using backpropagation algorithm. Convolutional layer acquires the significant special representation from an image, which is essentially used for categorizing images. Generally, the performance of any classification technique depends on the features considered for grouping the data. Selecting interesting and discriminative features from images is the very tedious task. However these extracted features may not be appropriate for all classification problems.

Convolution neural network is able to learn these features automatically to make better predictions without human intervention. Almost every convolutional layer is followed by a non-linear activation function, which helps the network to learn discriminative representations of the image that improve the classification accuracy. Figure 10.1 shows the typical CNN architecture.

The layers involved in the architecture are:

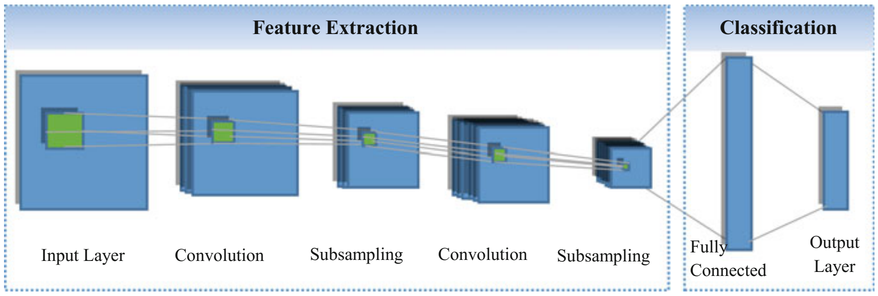


Fig. 10.1 The typical CNN architecture

10.2.1 Convolution Layer

Convolution layers are described by weights. This has multiple kernels per layer with fixed size, and each kernel is convolved over the entire image with a fixed stride that extracts a spatial or temporal features. The low-level features such as lines, edges, and corners are learned in the first convolution layer. More complex representations are learned in the consequent convolutional layers. As the network is deeper and deeper, the learned features contain higher-level information. The mathematical representation of the convolution operation is given in Eq. 1.

$$g(x,y) = h(x,y) * f(x,y) \tag{1}$$

where $f(x,y)$ is the convolution mask, $h(x,y)$ is the input image and $g(x,y)$ is the convoluted image.

In convolution operation, a filter slides over the input image to produce a feature map as shown in Fig. 10.2. Convolution operation captures different feature maps for the same input image with different filters. More features can be extracted by using more number of filters. In training, a CNN learns the values of these filters. The size of the feature map is determined by stride, padding and depth. Stride is the number of pixels that the filter jumps to slide over the input matrix. Larger stride will produce smaller feature maps. Affixing zeroes around the input matrix is called zero-padding or wide convolution. Padding allows the network to apply the filter to border elements of the input image matrix. Depth is the number of filters used in convolution operation.

10.2.2 Activation Layer

Activation layer uses activation functions that ignite a signal when a specific stimulus is presented. As compared to common activation functions such as tanh

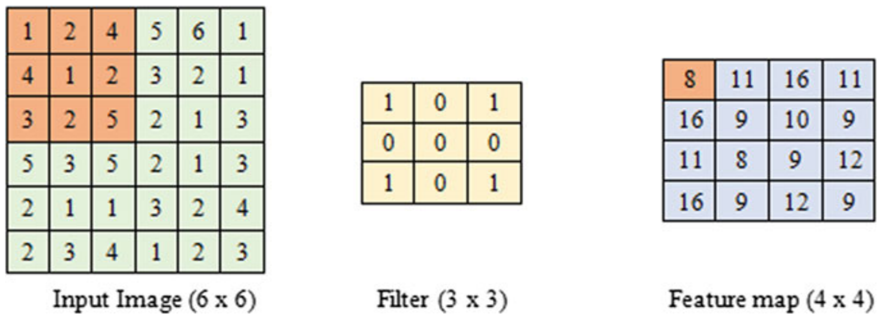


Fig. 10.2 Convolution of 5 × 5 image with 3 × 3 filter

and sigmoid, Rectified Linear Units (ReLU) is easy to compute and more robust to overfitting because of its sparse activation.

ReLU is the most common activation function used in convolution layer. Generally, activation function brings the non-linearity into DCNN. ReLU accelerates the convergence of the training procedure and leads to improved solutions. ReLU operation replaces all negative pixel values in the feature map by zero that is represented in Eq. 2.

$$relu(x) = \max(0, x) \quad (2)$$

where 'x' represents the input and $relu(x)$ represents the output function.

10.2.3 Pooling Layer

Pooling layer achieves a linear or non-linear downsampling. This layer reduces the computation complexity in terms of parameters reduction and alleviates overfitting. Pooling reduces the dimensionality of feature map but preserves the most important information. Various pooling methods are available for subsampling the feature map such as max, average, and sum. Max pooling operation takes the largest element from the rectified feature map within the window as shown in Fig. 10.3. As an alternative to taking the largest element, average or sum of all elements in that window can be taken.

10.2.4 Fully Connected Layer

All outputs of the preceding layer are attached to all inputs of the FC layer that predicts the image label. This layer uses activation functions such as softmax, sigmoid etc. for predicting the target class. Softmax function is used in the output layer for multi classification model, which return the probabilities of each class in that the target class has a higher probability. Softmax function provides a way of

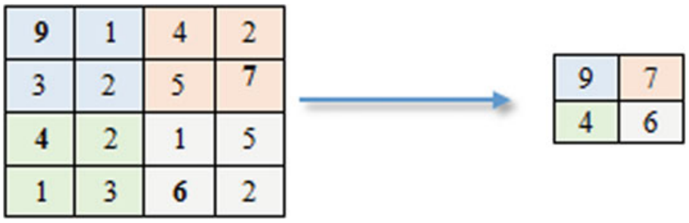


Fig. 10.3 Max pool with subsample 2 × 2

predicting discrete probability distribution over multiple classes and the sum of all the probabilities will be equal to one. Sigmoid function provides output in the range 0–1, which is mostly used for binary classification model.

10.3 Proposed DCNN Architecture

The DCNN architecture for classification of images in the cifar-10 dataset implemented in this work is shown in Fig. 10.4.

The DCNN model explored in this work consists of 6 consecutive convolutional layers and 3 fully connected layers. Each convolutional layer is followed by a subsampling layer. The input of the CNN model is a 32 × 32 × 3 image (i.e., the input has three channels of 32 × 32 pixels). The first convolutional stage consists of 48 kernels of size 3 × 3 with no subsampling. The second convolutional stage consists of 48 kernels of size 3 × 3 and a max pooling layer that subsamples the image by half. The third convolutional stage consists of 96 kernels of size 3 × 3 with no subsampling. The fourth convolutional stage consists of 96 kernels of size 3 × 3 and a max pooling layer that subsamples the image by half. The fifth convolutional stage consists of 192 kernels of size 3 × 3 with no subsampling. The sixth convolutional stage consists of 192 kernels of size 3 × 3 and a max pooling layer that subsamples the image by half.

Each kernel produces a 2-D image output (e.g., 48 of 32 × 32 images after the first convolutional layer), which is denoted as 48 @ 32 × 32 in Fig. 10.4. Kernels may contain different matrix values that are initialized randomly and updated during

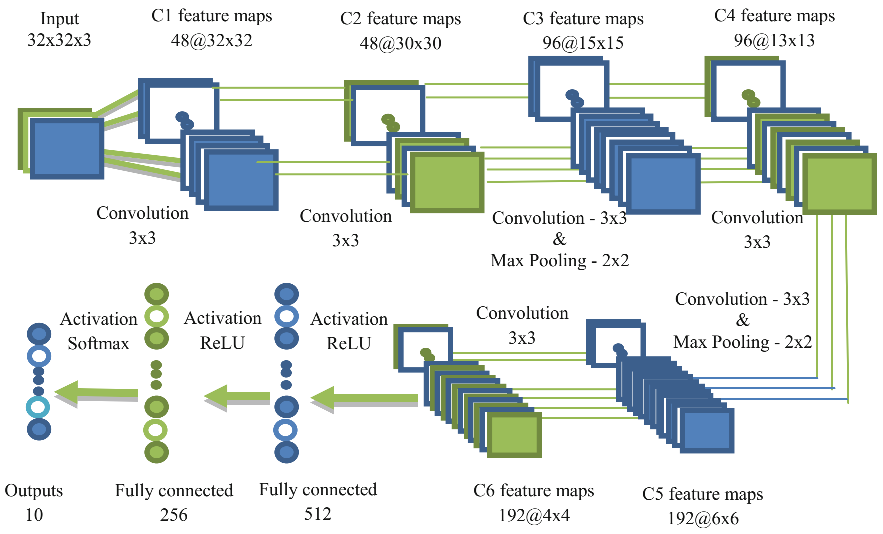


Fig. 10.4 Proposed DCNN architecture for cifar-10 dataset classification

training to optimize the classification accuracy. First, fully connected layer has 512 nodes and the second fully connected layer has 256 nodes, and the final stage has a softmax layer containing ten nodes. All convolutional and fully connected layers are equipped with the ReLU activation function. The last fully connected layer contains ten neurons which compute the classification probability for each class using softmax regression. To reduce overfitting, "dropout" and "batch normalization" is used after convolution layers. The effect of various optimizers in accelerating gradient descendent is analyzed in this work.

10.3.1 Dropout Layer

Dropout layer drops less contributed nodes in the forward pass by setting them to zero during training. Even some of the nodes are dropped out still the network can able to provide the correct classification for a given example, that makes sure that the network is not becoming too fitted to the training data and thus aids mitigate the overfitting problem. It is an optional layer in the architecture. The nodes to be dropped are randomly selected with a probability in each weight update cycle.

10.3.2 Batch Normalization

Normalization is simply a linear transformation applied to each activation. Batch normalization technique normalizes each input channel across a mini-batch as given in Eq. 3, which normalizes the activations of each channel with the mini-batch mean and mini-batch standard deviation i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}} \tag{3}$$

where $E[x]$ is mini-batch mean and $Var[x]$ is mini-batch standard deviation. Activations y_i are computed with the following transformation function for all input neurons, x_i.

$$y_i = w\hat{x}_i + b \tag{4}$$

where 'w' is weight and 'b' is bias.

Figure 10.5 illustrates the transformation of inputs (x_i) into activations (y_i) with batch normalization technique. Batch normalization acts like a regulator between input layer and transformation function, which normalize the inputs intended for distributing activation values uniformly all through the training process. A batch normalization layer is used between the convolutional layer and activation layer that

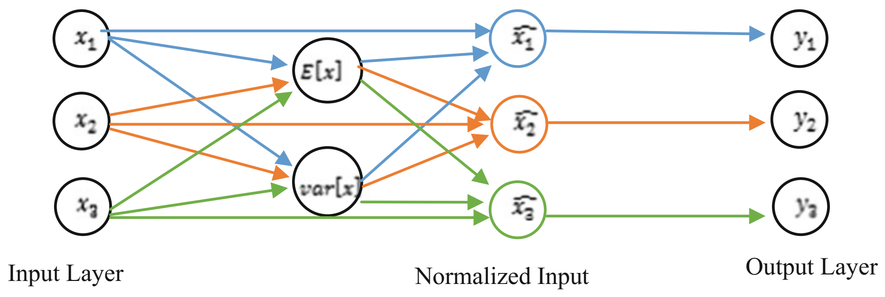


Fig. 10.5 Normalization of inputs with batch normalization

reduces the sensitivity to network initialization. Batch normalization significantly accelerates training speed by reducing vanishing gradient problems [24]. The presence of batch normalization has a benefit of optimizing the network training. Also, this has other benefits such as easy weight initialization, improvement in training speed, higher learning rate and regularization of values for the activation function.

10.3.3 Optimizing Gradient Descendant with Various Optimizer

The trainable parameters of CNN play a major role in efficiently and effectively training a model and produce accurate results. Optimization strategies have great influence on model's learning process and the prediction process. Optimization helps to minimize the error at training process and tune the model's internal learnable parameters such as weights (W) and the bias (b) values.

Gradient Descent is the most important technique used for training and optimizing Intelligent systems. Gradient descent works by iteratively performing updates based on the first derivative of a problem. For speedups, a technique called "momentum" is often used, which averages search steps over iterations. Gradient descent can be very effective, if the learning rate and momentum are well tuned. In order to achieve the objective, model learns appropriate model parameters in every iteration. Convergence of network depends on the internal structure of the model and optimizer [25]. The formula for updating the parameter in the model is given in Eq. 5.

$$\theta = \theta - \delta \cdot \nabla J(\theta) \tag{5}$$

where 'δ' represents the learning rate, '$\nabla J(\theta)$' represents the Gradient of Loss function $J(\theta)$ with respect to 'θ'.

Crossentropy loss function is the widely used cost function, which is used as an objective function to optimize the classification task. Crossentropy describes the loss between the predicted probability distributions and target probability distributions. Crossentropy is measured by Eq. 6.

$$H(p,q) = -\sum p_i \log q_i \tag{6}$$

where p_i is the target probability distribution and q_i is the predicted probability distribution of the current model.

Momentum: Momentum is a technique for quickening the Stochastic Gradient Descent (SGD) by changing the momentum largely towards the desired direction and minimally towards the fluctuating direction. When the objective function reaches local minima, the momentum is high. So the model is getting into local minima, is negligible. Moreover, this method performs larger updates frequently by which the model may miss the actual minima.
RmsProp: RmsProp is an optimizer that utilizes the magnitude of recent gradients to normalize the gradients. This method divides the current gradient by a moving average over the root mean squared gradients. RmsProp would boost the parameter multiple times and decrement it once by the current gradient. Also this has adaptable learning rate. This is a very robust optimizer which can deal stochastic objectives very nicely, making it applicable to mini-batch learning.
Adadelta: Adadelta is a method that uses the magnitude of recent gradients and steps to obtain an adaptive learning rate. This method stores an exponential moving average over the gradients and learning rate. The scale of learning rate for each individual parameter obtained by their ration.
Adam: Adaptive Moment Estimation (Adam) is another method, which determines the learning rates for each parameter. The scale of learning rate for each individual parameter obtained by their importance. Choosing a proper learning rate is a challenging task. Small learning rate leads to painfully slow convergence. Since adaptive algorithms dynamically adapt the learning rate and momentum, it supports network to converge quickly and discover the accurate parameter values. Whereas standard momentum techniques are deliberate in reaching the global minima. This method stores an exponentially moving average over the past squared gradients.

10.4 Results and Discussion

The proposed DCNN architecture has been trained and validated with images in the cifar-10 dataset. This dataset contains 60,000 images of size 32 × 32 with the following 10 categories such as airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Some of the sample images from this dataset are depicted in Fig. 10.6. In the dataset 50,000 images are used as the training set and 10,000 images are used as the validation dataset. The experimental DCNN architecture is

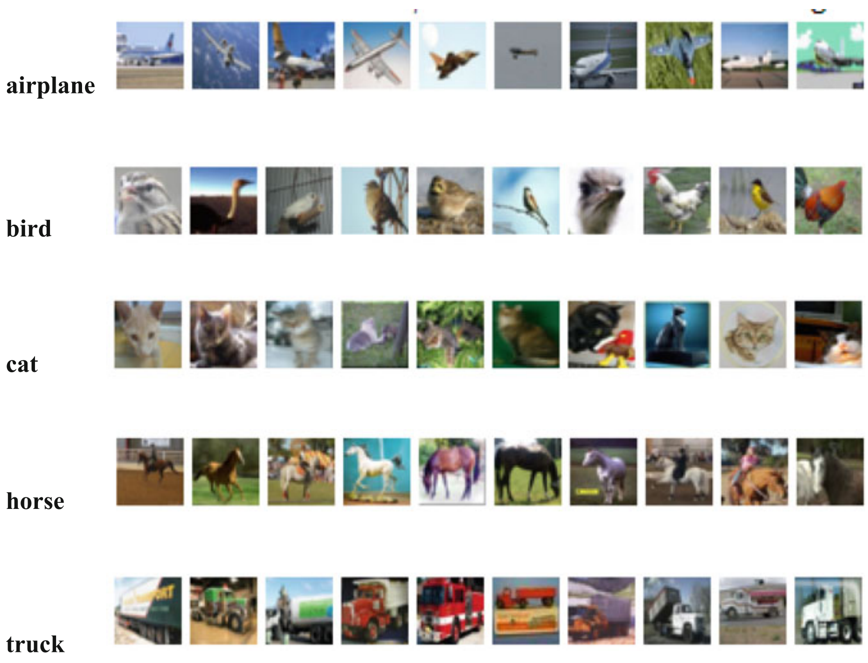


Fig. 10.6 Sample images from the cifar-10 dataset

developed in Keras, written in Python. This is an open source high level library, used to build neural network models.

The model uses accuracy as a metrics that can be evaluated during training and testing. The performance of the model is measured by validation score. The efficient model which is trained with a part of the dataset could able to predict the new one, that has never used for training. A loss function or objective function used in this experiment is crossentropy which is commonly used for image classification tasks. The classifier tries to minimize this crossentropy between the target and the estimated class probabilities.

This section presents the experimental results obtained during training and testing stages on the cifar-10 image dataset. In order to speed up the experiments, the training of the network is stopped, if the validation accuracy is not improved for 5 consecutive epochs. The upper bound for the number of training epochs considered in this experiment is 25 epochs.

Figure 10.7 shows a training loss over time for DCNN with various optimizers. İt is observed that the crossentropy loss over time is much higher throughout the training process for Rmsprop, momentum and Adadelta optimizers. Whereas the stepwise behavior of the entropy loss in Adam optimizer is significantly reduced over time.

The validation accuracy for DCNN with various optimizers for first 10 epochs is presented in Table 10.1.

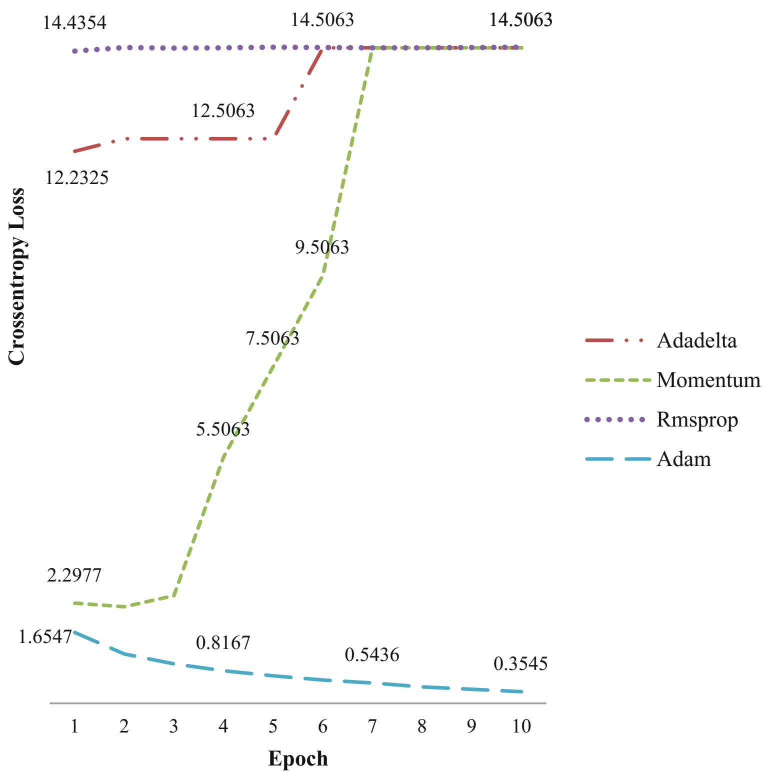


Fig. 10.7 Training loss over time for DCNN with various optimizers

Table 10.1 Performance comparison of various optimizers in DCNN in terms of validation accuracy (%)

Optimizers in DCNN				
Epochs	Adadelta	Momentum	Rmsprop	Adam
1	10	19.38	10	49.87
2	10	27.4	10	61.3
3	10	9.98	10	67.87
4	10	10	10	70.54
5	10	10	10	73.46
6	10	10	10	74.55
7	10	10	10	76.19
8	10	10	10	77.74
9	10	10	10	75.46
10	10	10	10	77.74

The back propagation network uses the batch gradient technique, which is a first-order optimization technique with favorable convergence properties. The performance of models with various optimizers in terms of validation accuracy is reported in Table 10.1. It is observed that Rmsprop, momentum, and Adadelta optimizers are not able to achieve validation accuracies whereas Adam optimizer achieves higher validation accuracy. From these observations, it is concluded that the Adam optimizer outperforms other optimizers in terms of entropy loss and accuracy due to its adaptive learning nature on the training set. The results show that the effect of optimizers also significantly changes the accuracy of the model, in addition to the structure of the architecture.

The model summary with a number of parameters used in the proposed DCNN architecture is shown in Table 10.2. It shows the number of parameters initialized at every layer in the DCNN architecture. Trainable parameters are initialized with minimum random numbers to avoid dead neurons, but not too small to avoid zero gradients. Uniform distribution is generally preferred for parameter initialization. Totally 1,172,410 parameters are tuned by DCNN training to classify the images in the cifar-10 dataset. The parameters in the model can be further re-tuned by introducing dropout and batch normalization.

Table 10.2 Model summary of the proposed DCNN architecture

Layer (type)	Output shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 48)	1344
activation_1 (Activation)	(None, 32, 32, 48)	0
conv2d_2 (Conv2D)	(None, 30, 30, 48)	20,784
activation_2 (Activation)	(None, 30, 30, 48)	0
max_pooling2d_1 (MaxPooling2)	(None, 15, 15, 48)	0
conv2d_3 (Conv2D)	(None, 15, 15, 96)	41,568
activation_3 (Activation)	(None, 15, 15, 96)	0
conv2d_4 (Conv2D)	(None, 13, 13, 96)	83,040
activation_4 (Activation)	(None, 13, 13, 96)	0
max_pooling2d_2 (MaxPooling2)	(None, 6, 6, 96)	0
conv2d_5 (Conv2D)	(None, 6, 6, 192)	166,080
activation_5 (Activation)	(None, 6, 6, 192)	0
conv2d_6 (Conv2D)	(None, 4, 4, 192)	331,968
activation_6 (Activation)	(None, 4, 4, 192)	0
max_pooling2d_3 (MaxPooling2)	(None, 2, 2, 192)	0
flatten_1 (Flatten)	(None, 768)	0
dense_1 (Dense)	(None, 512)	393,728
activation_7 (Activation)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131,328
activation_8 (Activation)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570
Total params: 1,172,410		
Trainable params: 1,172,410		
Non-trainable params: 0		

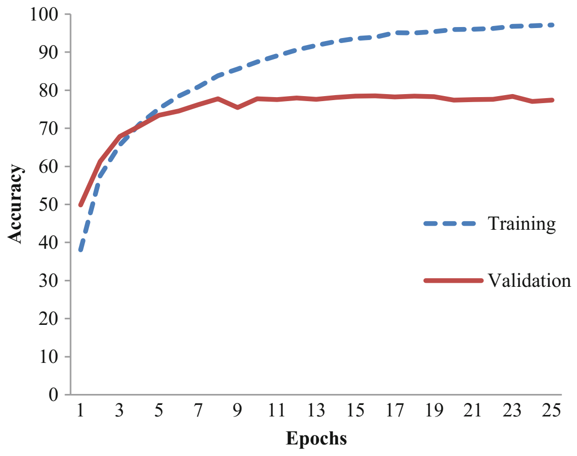


Fig. 10.8 Training and validation accuracy curve for DCNN with Adam optimizer

The training and validation accuracy for the proposed model with Adam optimizer is shown in Fig. 10.8. It is observed from the figure that in later epochs, there is no significant improvement in the validation accuracy as compared to the training accuracy. The best training accuracy and validation accuracy achieved in this model is 97.13 and 78.47% respectively. The validation accuracy is lower than the training accuracy due to overfitting of the model. This happens when the model learns the training data very detail and creates a negative impact on the performance of the model on new data. This issue can be solved by introducing dropout after the convolution layer.

Regularization is a very important technique to prevent over fitting in machine learning problems. In this model, the regularization technique called dropout is applied to avoid over fitting. Dropout does not rely on modifying the loss function but the network itself. Figure 10.9 shows the performance of the network model by introducing the dropout of 0.5 after the dense layer. It is noticed that the validation

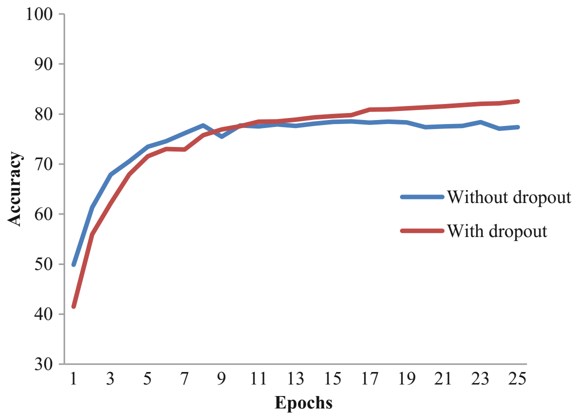


Fig. 10.9 Performance comparison of validation accuracy without dropout and with dropout

accuracy suddenly start to go up and oscillate on high values until the next learning rate drop.

The key idea of dropout is randomly droping the parts of neural network during training and thus preventing the over learning of features. It is observed from Fig. 10.9 that after 10 epochs, there is an improvement in the validation accuracy with dropout compared to a network without dropout. Dropout decreases the loss from 1.1423 to 0.6112 and improves validation accuracy from 77.4 to 81.32% on cifar-10 dataset. Also, time taken for training the neural network is minimized.

The model summary of the proposed DCNN architecture with batch normalization is shown in Table 10.3. To improve the efficiency of the DCNN model,

Table 10.3 Model summary of the proposed DCNN architecture with batch normalization

Layer (type)	Output shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 48)	1344
batch_normalization_1 (Batch)	(None, 32, 32, 48)	192
activation_1 (Activation)	(None, 32, 32, 48)	0
conv2d_2 (Conv2D)	(None, 30, 30, 48)	20,784
batch_normalization_2 (Batch)	(None, 30, 30, 48)	192
activation_2 (Activation)	(None, 30, 30, 48)	0
max_pooling2d_1 (MaxPooling2)	(None, 15, 15, 48)	0
conv2d_3 (Conv2D)	(None, 15, 15, 96)	41,568
batch_normalization_3 (Batch)	(None, 15, 15, 96)	384
activation_3 (Activation)	(None, 15, 15, 96)	0
conv2d_4 (Conv2D)	(None, 13, 13, 96)	83,040
batch_normalization_4 (Batch)	(None, 13, 13, 96)	384
activation_4 (Activation)	(None, 13, 13, 96)	0
max_pooling2d_2 (MaxPooling2)	(None, 6, 6, 96)	0
conv2d_5 (Conv2D)	(None, 6, 6, 192)	166,080
batch_normalization_5 (Batch)	(None, 6, 6, 192)	768
activation_5 (Activation)	(None, 6, 6, 192)	0
conv2d_6 (Conv2D)	(None, 4, 4, 192)	331,968
batch_normalization_6 (Batch)	(None, 4, 4, 192)	768
activation_6 (Activation)	(None, 4, 4, 192)	0
max_pooling2d_3 (MaxPooling2)	(None, 2, 2, 192)	0
flatten_1 (Flatten)	(None, 768)	0
dense_1 (Dense)	(None, 512)	393,728
activation_7 (Activation)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131,328
activation_8 (Activation)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570
Total params: 1,175,098		
Trainable params: 1,173,754		
Non-trainable params: 1344		

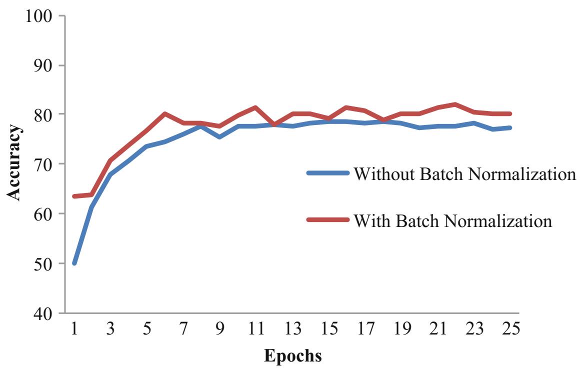
batch normalization layer is added after every convolution layer. In this experiment, a momentum of 0.99 is used in the batch normalization layer for moving the mean and variance. The presence of this layer improves the overall accuracy and learning rate. This layer performs a transformation at each batch by normalizing the previous layer's activations that in turn maintains activation mean towards to 0 and standard deviation towards to 1. The model re-tuned with batch normalization yields the number of trainable parameters as 1,175,098.

The effect of the batch normalization in the performance of the model in terms of validation accuracy is shown in Fig. 10.10. It shows that batch normalization really has positive effects on neural networks but it delays the convergence of network. By observing the loss over time, the regularizing effect of batch normalization becomes very prominent. The batch normalized network learns consistently. Overall, batch normalized models achieve higher validation and test accuracies on all datasets. Due to these results, the use of batch normalization is generally advised since it prevents model divergence and may increase convergence speeds through higher learning rates.

The performance of batch normalization with and without dropout is shown in Fig. 10.11. The batch normalization and dropout can be used at the same time for improving the accuracy in the validation dataset. The batch normalized model consistently achieves higher validation accuracy. Whereas it adds computational complexity that can be handled by keeping higher learning rate. It is recommended to keep the batch normalization between convolution and activation layers for getting best results. Also, the dropout layer introduced after dense layer reduces the over fitting issues. Figure 10.10 shows the accuracy improvement from 79.99 to 83.23% in first 25 epochs with the inclusion of dropout and batch normalization in the deep CNN for cifar-10 dataset.

Further, the performance of the classification system can be improved by changing the structure of architecture and tuning its parameters. Besides the number of layers and the layer density of the architecture, all tunable factors such as the

Fig. 10.10 Performance comparison of validation accuracy without batch normalization and with batch normalization



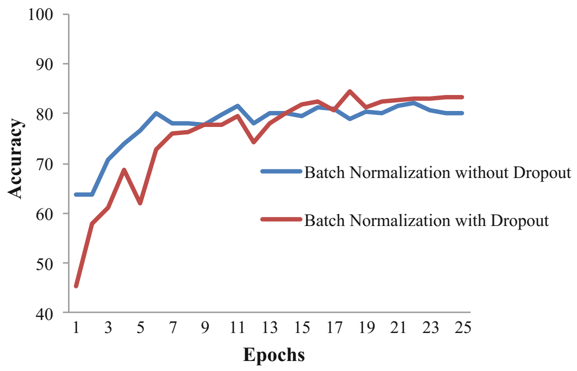


Fig. 10.11 Performance comparison of validation accuracy without dropout and with dropout in batch normalization

filter size, pooling method, number of epochs and layer patterns can improve the accuracy further. The architecture of CNN goes deeper and deeper, the network needs to learn tens of thousands to millions of parameters. A large amount of training data is required to train these parameters properly. Overfitting problem can be caused by poor quality training data and it can be avoided by training the network with noise free training data. Also overfitting due to the small dataset can be reduced with data augmentation, which can increase the training data by performing various transformations.

10.5 Conclusion

A new deep convolution neural network model is proposed in this chapter for image classification in cifar-10 dataset. The proposed model is analyzed with various optimization strategies, the inclusion of dropout and batch normalization. The Adam optimizer reduces the entropy loss over time as compared to other optimizers such as momentum, Adadelta and Rmsprop in this model. The Adam optimizer achieves a maximum validation accuracy of 78% in the first 25 epochs to classify the images in cifar-10 dataset. İntroducing dropout after dense layer prevents the model from learning too detail with the training data and achieves an accuracy of 81.32%. The CNN model with dropout and batch normalization ensures an improved performance in the validation phase with an accuracy of 83.42%. Since CNN model with batch normalization and dropout avoids model deviation, it is recommended to use with higher learning rates. Experimental results show that the proposed model exhibits significant improvement in the performance on classifying the images in the cifar-10 dataset.

References

1. Ionescu, R.T., Popescu, M.: State-of-the-art approaches for image classification. In: Knowledge Transfer between Computer Vision and Text Mining, pp. 41–52, Springer (2016)
2. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. **18**(7), 1527–1554 (2006)
3. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
4. Deng, L., Yu, D.: Deep learning: methods and applications. Found. Trends Signal Process. **7** (3–4), 197–387 (2014)
5. Da, C., Zhang, H., Sang, Y.: Brain CT image classification with deep neural networks. In: Handa, H., Ishibuchi, H., Ong, Y.S., Tan, K. (eds.) Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, vol. 1, pp. 653–662. Springer (2015)
6. Paoletti, M.E., Haut, J.M., Plaza, J., Plaza, A.: A new deep convolutional neural network for fast hyperspectral image classification. ISPRS J. Photogramm. Remote Sens. (2017)
7. Sudars, K.: Face recognition Face2vec based on deep learning: small database case. Autom. Control Comput. Sci. **51**(1), 50–54 (2017)
8. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
9. Mugahed, A., et. al.: An automatic computer-aided diagnosis system for breast cancer in digital mammograms via deep belief network. J. Med. Biol. Eng., 1–14 (2017)
10. Wang, Y., et. al.: Automatic tumor segmentation with deep convolutional neural networks for radiotherapy applications. Neural Process. Lett., 1–12 (2018)
11. Zhang, Y.D., et. al.: Image based fruit category classification by 13-layer deep convolutional neural network and data augmentation. Multimed. Tools Appl., 1–20 (2017)
12. Chi, J., Walia, E., Babyn, P., Wang, J., Groot, G., Eramian, M.: Thyroid nodule classification in ultrasound images by fine-tuning deep convolutional neural network. J. Digit. Imaging **30** (4), 477–486 (2017)
13. Wang, X., Zhang, W., Wu, X., Xiao, L., Qian, Y., Fang, Z.: Real-time vehicle type classification with deep convolutional neural networks. J. Real-Time Image Process., 1–10 (2017)
14. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems, vol. 1, pp. 1097–1105 (2012)
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint. arXiv:1409–1556 (2014)
16. Szegedy, C., et. al.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
17. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv:1605.07146 (2017)
18. Huang, G., et. al.: Deep networks with stochastic depth. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer Vision—ECCV 2016. Lecture Notes in Computer Science, vol. 9908. Springer (2016)
19. Srivastava, N., et. al.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learning Res. **15**, 1929–1958 (2014)
20. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning, vol. 37, pp. 448–456 (2015)
21. https://www.cs.toronto.edu/kriz/cifar.html
22. Krizhevsky, A.: Learning multiple layers of features from tiny images, Technical report. University of Toronto (2009)

23. Liu, S., Deng, W.: Very deep convolutional neural network based image classification using small training sample size. In: 3rd IAPR Asian Conference on Pattern Recognition (2015)
24. Bengio, Y., Simard, P.: Frasconi. P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
25. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv:1609.04747 (2017)