International Journal of Computer Science and Information Technology
ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 4, Number 3, Year 2024
DOI: https://doi.org/10.62051/ijcsit.v4n3.43
Journal homepage: https://wepub.org/index.php/IJCSIT/index

Optimization and Practice of Convolutional Neural Networks in Image Classification Tasks

Lubin Liu

Shenzhen University, College of Electronic and Information Engineering, Shenzhen, Guangdong Province 518000, China

ABSTRACT

This study focuses on the optimization and practice of Convolutional Neural Networks (CNN) in image classification tasks. A deep neural network model was constructed by introducing the CBAM (Convolutional Block Attention Module) attention mechanism and residual structure. This model combines the stability of residual networks during training with the advantages of CBAM in feature extraction, aiming to improve image classification performance. The experiment used the CIFAR-10 dataset and achieved a testing accuracy of 86% through optimization methods such as data augmentation, batch normalization, reasonable control of the number of fully connected layers, and exponential decay learning rate strategy. The study also explored the effects of data augmentation, batch normalization layer utility, fully connected layer quantity control, residual structure effectiveness, and CBAM attention mechanism on model performance, providing valuable insights for optimizing CNN in image classification tasks.

KEYWORDS

Convolutional Neural Network (CNN); Image classification; CBAM attention mechanism; Residual structure; Data augmentation; Batch normalization; Test accuracy

1. INTRODUCTION

With the continuous development of artificial intelligence technology, computer vision, as one of its core fields, is gradually penetrating into various industries. Image classification, as a fundamental task in computer vision, aims to automatically classify images into predefined categories. Traditional image classification methods rely on manually designed features and machine learning algorithms, which perform poorly when dealing with complex images. With the rise of deep learning technology, Convolutional Neural Networks (CNNs) have gradually become the mainstream method for image classification tasks [1]. Convolutional neural networks can automatically learn high-level feature representations from raw images by simulating the hierarchical structure of the human visual system, thus achieving significant performance improvements in image classification tasks. However, in practical applications, the performance of convolutional neural networks is still affected by various factors, such as network structure, data preprocessing, regularization techniques, training strategies, etc [2]. Therefore, in-depth research on the optimization and practice of convolutional neural networks in image classification tasks is of great significance for improving their performance and promoting their popularization in practical applications.

2. MODEL ARCHITECTURE AND INNOVATION

2.1. Overview of Model Architecture

In this study, a deep neural network model was proposed that integrates channel attention and spatial attention mechanisms with CBAM (Convolutional Block Attention Module) and residual structure, aiming to achieve efficient and accurate feature extraction and classification. The model architecture consists of two residual blocks, one convolutional layer, two fully connected layers, and an embedded CBAM attention mechanism. This design combines the stability of residual networks in deep neural network training with the advantages of CBAM attention mechanism in feature extraction, aiming to improve the performance of the model in image classification tasks.

2.2. Residual Block Design

The core design of residual blocks lies in optimizing the training process of deep neural networks by introducing skip connections (also known as "shortcuts") [3]. Within the residual block, a padding strategy is employed to ensure that the input and output feature maps are of the same size, while utilizing two convolutional layers for feature extraction. Among them, one convolutional layer uses a 1x1 convolution kernel to ensure consistency in the number of channels. This design not only helps maintain a stable distribution of data, but also significantly accelerates the training speed of the model.

The advantage of residual blocks lies in their easy to learn nature, which is beneficial for optimizing the network. As the depth of the network increases, residual blocks can achieve higher accuracy, and the learned residuals are also more accurate. This design enables deep neural networks to better avoid the problem of gradient vanishing or exploding during training, thereby improving the training efficiency and final performance of the model.

2.2.1. CBAM attention mechanism

CBAM consists of Channel Attention (CA) and Spatial Attention (SA) mechanisms, which are used in series to achieve fine tuning of feature maps [4].

Channel Attention Mechanism (CA): Firstly, the feature map is compressed in two dimensions: global average pooling and global max pooling. Then, 1x1 convolution is used for dimensionality reduction and enhancement. Finally, the sigmoid activation function is used to obtain the weight of each channel. These weights are used to adjust the feature map strength of the corresponding channels, thereby enhancing the model's attention to important feature channels.

Spatial Attention Mechanism (SA): Firstly, the channels of the feature map are compressed, and two feature maps are obtained using global average pooling and global max pooling. Then, stack these two feature maps in the channel dimension and use 1x1 convolution to adjust the number of channels. Finally, the sigmoid activation function is used to obtain the weights for each spatial domain, which are used to adjust the strength of the feature maps corresponding to the spatial positions, enabling the model to more accurately focus on important feature positions.

As a lightweight attention module, CBAM has low complexity and computational complexity, while demonstrating strong versatility. By introducing CBAM, the model can more effectively focus on important feature channels and spatial positions, thereby improving overall performance.

2.2.2. Design of pooling layer and convolutional layer

In this model, two pooling layers are used for downsampling, reducing the size of the feature map to decrease the model's parameter and computational complexity. This helps to accelerate the calculation speed and prevent overfitting. In addition, to increase the number of convolutional layers and expand the receptive field, a convolutional layer was added at the end of the model. This layer of convolution not only helps to extract richer feature information, but also reduces the number of

parameters connected to fully connected layers, thereby reducing the complexity and overfitting risk of the model [5].

2.2.3. Fully connected layer design

Finally, two fully connected layers were used for the classification task. These two fully connected layers are responsible for mapping the extracted features into the category space and outputting the final classification result. By optimizing the parameter configuration and activation function selection of the fully connected layer, the classification performance of the model can be further improved.

3. EXPERIMENTAL CORE CODE

3.1. Data section

3.1.1. Define dataset

In deep learning research, reasonable partitioning and preprocessing of datasets are crucial for model training and validation. In this study, the classic CIFAR-10 dataset was used and scientifically partitioned and preprocessed. The CIFAR-10 dataset is clearly divided into two parts: the training set and the validation set. The training set is used for the training process of the model, while the validation set is used to evaluate the performance of the model. To ensure the randomness and generalization ability of the data, DataLoader was used to encapsulate the training and validation sets, and reasonable batch sizes and random shuffling strategies were set.

```
self.train_dataset=cifar.CIFAR10(root=self.path,train=True,transform=self.transforms_train,download=True)
self.test_dataset=cifar.CIFAR10(root=self.path,train=False,transform=self.transforms_test,download=True)
self.train_loader=DataLoader(dataset=self.train_dataset,batch_size=self.batch_size,shuffle=True)
self.test_loader=DataLoader(dataset=self.test_dataset,batch_size=self.batch_size,shuffle=True)
```

3.1.2. Data enhancement

In deep learning tasks, data augmentation is an effective strategy to enhance the model's generalization ability. A series of complex data augmentation processes were implemented for the training set, aiming to prevent model overfitting by increasing data diversity. After preprocessing such as filling, cropping, and random flipping, the image is converted to Tensor type and standardized to accelerate model training and improve convergence. In contrast, the processing of validation sets is relatively simple, only including conversion to Tensor types and standardization.

```
self.transforms_train=transforms.Compose([
transforms.RandomCrop(32,padding=4),
transforms.RandomHorizontalFlip(),
transforms.ToTensor(),
transforms.Normalize((0.4914,0.4822,0.4465),(0.2023,0.1994,0.2010)),
])
self.transforms_test=transforms.Compose([
transforms.ToTensor(),
transforms.Normalize((0.4914,0.4822,0.4465),(0.2023,0.1994,0.2010)),])
```

3.2. Model Section

3.2.1. Subject of network

This article is based on a deep learning model using convolutional neural networks, named CNNNet. This model integrates attention mechanism and residual structure, aiming to improve the performance of image classification tasks. The main body of the network consists of multiple convolutional layers, batch normalization layers, pooling layers, and fully connected layers. Specifically, the model introduces the CBAM attention module in the initial stage to capture key feature information in the image. Subsequently, by stacking multiple convolutional blocks and connecting them with residuals, the model can gradually extract high-order features of the image.

```
class CNNNet(nn.Module):
def__init__(self):
super(CNNNet,self).__init__()
self.cbam=CBAM(3)
self.conv1=nn.Conv2d(in_channels=3,out_channels=16,kernel_size=3,stride=1,padding=1,bias=False)
self.bn1=nn.BatchNorm2d(16)
self.conv2=nn.Conv2d(in_channels=16,out_channels=36,kernel_size=3,stride=1,padding=1,bias=False)
self.bn2=nn.BatchNorm2d(36)
self.res1=nn.Conv2d(in_channels=3,out_channels=36,kernel_size=1,stride=1,bias=False)
self.resbn1=nn.BatchNorm2d(36)
self.pool1=nn.MaxPool2d(kernel_size=2,stride=2)
self.conv3=nn.Conv2d(in_channels=36,out_channels=72,kernel_size=3,stride=1,padding=1,bias=False)
self.bn3=nn.BatchNorm2d(72)
self.conv4=nn.Conv2d(in_channels=72,out_channels=156,kernel_size=3,stride=1,padding=1,bias=False)
self.bn4=nn.BatchNorm2d(156)
self.res2=nn.Conv2d(in_channels=36,out_channels=156,kernel_size=1,stride=1,bias=False)
self.resbn2=nn.BatchNorm2d(156)
self.pool2=nn.MaxPool2d(kernel_size=2,stride=2)
self.conv5=nn.Conv2d(in_channels=156,out_channels=312,kernel_size=5,stride=1,bias=False)
self.bn5=nn.BatchNorm2d(312)
self.fc1=nn.Linear(312*4*4,156)
self.fc2=nn.Linear(156,10)
def forward(self,x):
x=self.cbam(x)
x1=self.resbn1(self.res1(x))
x=F.relu(self.bn1(self.conv1(x)))
x=F.relu(self.bn2(self.conv2(x))+x1)
x=self.pool1(x)
x2=self.resbn2(self.res2(x))
x=F.relu(self.bn3(self.conv3(x)))
x=F.relu(self.bn4(self.conv4(x))+x2)
x=self.pool2(x)
```

```
x=F.relu(self.bn5(self.conv5(x)))
x=x.view(-1,312*4*4)
x=F.relu(self.fc1(x))
x=F.relu(self.fc2(x))
return x
```

3.2.2. CBAM structure

The CBAM module consists of a channel attention module and a spatial attention module. The channel attention module learns the weights of each channel through global average pooling and global maximum pooling operations, combined with a multi-layer perceptron structure. The spatial attention module focuses on the spatial position information of the feature map, calculates the average pooling and maximum pooling of the channel dimension feature map, and processes it through convolutional layers to generate a spatial attention weight map.

```
class ChannelAttention(nn.Module):
def__init__(self,in_planes,ratio=3):
super(ChannelAttention,self).__init__()
self.avg_pool=nn.AdaptiveAvgPool2d(1)
self.max_pool=nn.AdaptiveMaxPool2d(1)
self.fc1=nn.Conv2d(in_planes,in_planes//ratio,1,bias=False)
self.relu1=nn.ReLU()
self.fc2=nn.Conv2d(in_planes//ratio,in_planes,1,bias=False)
self.sigmoid=nn.Sigmoid()
def forward(self,x):
avg_out=self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
max_out=self.fc2(self.relu1(self.fc1(self.max_pool(x))))
out=avg_out+max_out
return self.sigmoid(out)
class SpatialAttention(nn.Module):
def__init__(self,kernel_size=7):
super(SpatialAttention,self).__init__()
assert kernel_size in(3,7),'kernel size must be 3 or 7'
padding=3 if kernel_size==7 else 1
self.conv1=nn.Conv2d(2,3,kernel_size,padding=padding,bias=False)
self.sigmoid=nn.Sigmoid()
def forward(self,x):
avg_out=torch.mean(x,dim=1,keepdim=True)
max_out,_=torch.max(x,dim=1,keepdim=True)
x=torch.cat([avg_out,max_out],dim=1)
x=self.conv1(x)
return self.sigmoid(x)
class CBAM(nn.Module):
def__init__(self,planes):
super(CBAM,self).__init__()
```

```
self.ca=ChannelAttention(planes)
self.sa=SpatialAttention()
def forward(self,x):
x=self.ca(x)*x
x=self.sa(x)*x
return x
```

3.3. Training Part

3.3.1. Loss function

In classification tasks, the cross entropy loss function is used to measure the difference between the model's predicted results and the actual labels.

```
self.criterion=nn.CrossEntropyLoss()
```

3.3.2. Optimizer

Using stochastic gradient descent (SGD) optimizer to optimize model parameters and introducing momentum mechanism to accelerate convergence.

```
self.optimizer=torch.optim.SGD(network.parameters(),lr=0.001,momentum=0.9)
```

3.3.3. Learning strategy

Adopting an exponential decay strategy to dynamically adjust the learning rate to adapt to different stages of training.

```
self.step_schedule=torch.optim.lr_scheduler.ExponentialLR(self.optimizer,gamma=0.96)
```

3.3.4. Training process

The training process includes steps such as forward propagation, loss calculation, backpropagation, and parameter update. Gradually optimize model performance through iterative training cycles.

```
def train(self,epochs=3):
print('----------Train Start----------')
for epoch in range(epochs):
losses=0.0
for i,data in enumerate(self.train_loader,0):
train_data,train_label=data
train_data,train_label=train_data.to(device),train_label.to(device)
self.optimizer.zero_grad()
predict=self.network(train_data)
loss=self.criterion(predict,train_label)
loss.backward()
self.optimizer.step()
losses+=loss.item()
if i%1000==999:
print(f'[epoch{epoch+1}/{epochs},{(i+1)/len(self.train_loader)*100:.2f}%]loss:{losses/1000:.2f}')
losses=0.0
self.evaluate(self.test_loader)
self.step_schedule.step()
```

```
print('----------Finished Training----------')
torch.save(self.network.state_dict(),'./checkpoint/model.pth')
```

3.3.5. Testing section

Evaluate the performance of the model on the validation set and measure its generalization ability by calculating accuracy.

```
def evaluate(self,data_loader):
correct=0
total=0
with torch.no_grad():
for data in data_loader:
inputs,labels=data
inputs,labels=inputs.to(device),labels.to(device)
outputs=self.network(inputs)
_,predicted=torch.max(outputs.data,1)
total+=labels.size(0)
correct+=(predicted==labels).sum().item()
print(f'Accuracy of the network on the 10000 test images:{100*correct//total}%')
```

4. EXPERIMENTAL PROCEDURE

4.1. EDR

During the experiment, key indicators such as training loss, validation loss, and validation accuracy were recorded in detail for each training cycle. These data help analyze the training effectiveness and generalization ability of the model.

4.2. Experimental Test Results

After multiple experiments and adjustments, the final model achieved a testing accuracy of 86% on the CIFAR-10 dataset. This indicates that the proposed model architecture and optimization strategy have good performance in image classification tasks.

5. EXPERIMENTAL SUMMARY

This experiment successfully constructed and trained a Convolutional Neural Network (CNN), which achieved a testing accuracy of 86% through a series of adjustments and optimizations. This experiment delves into the impact of network structure optimization, data preprocessing, regularization techniques, and training strategies on model performance, and draws the following conclusions:

The importance of data augmentation: The correct data augmentation strategy is crucial for significantly improving model accuracy. In this experiment, random flipping and cropping of images have been proven to be extremely effective enhancement methods.

The utility of batch normalization (BN) layer: BN layer performs well in accelerating model convergence. However, its effectiveness is highly dependent on the choice of batch size. When the batch size is too small, the BN layer may not be able to accurately simulate the statistical

characteristics of the entire dataset, thereby affecting its performance. Therefore, when using BN layers, the batch size should be appropriately increased to ensure their effectiveness.

Control of the number of fully connected layers: Too many fully connected layers may slow down the learning speed of the network and affect the overall performance of the model. This indicates that when building a network, it is necessary to reasonably control the number of fully connected layers to balance the complexity and learning ability of the model.

The effectiveness of residual structure: Experimental results show that residual structure can stably improve the performance of the model. By introducing residual connections, the model can more effectively convey gradient information, alleviate the problem of gradient vanishing in deep networks, and thus improve the training efficiency and final performance of the model.

The performance of CBAM attention mechanism: Although CBAM attention mechanism can enhance the model's attention to important features in some cases, its effect is not significant in this experimental task.

REFERENCES

[1] Zhu Junyu Technical Analysis of Image Recognition Based on Convolutional Neural Networks [J]. Changjiang Information and Communication, 2023, 36 (08): 66-68

[2] Li Wenjing, Bai Jing, Peng Bin, Yang Zhanyuan Overview of Graph Convolutional Neural Networks and Their Applications in Image Recognition [J]. Computer Engineering and Applications, 2023, 59 (22): 15-35

[3] Dou Hui, Zhang Lingming, Han Feng, Shen Fufu, Zhao Jian A review of interpretability research on convolutional neural networks [J]. Journal of Software, 2024, 35 (01): 159-184

[4] Zhang Ke, Feng Xiaohan, Guo Yurong, Su Yukun, Zhao Kai, Zhao Zhenbing, Ma Zhanyu, Ding Qiaolin A Review of Deep Convolutional Neural Network Models for Image Classification [J]. Chinese Journal of Image and Graphics, 2021, 26 (10): 2305-2325

[5] Wei Xiushen Analytical Deep Learning [M]. Electronic Industry Press: 201811.200