Computer Life
ISSN: 1819-4818 | Vol. 12, No. 3, 2024

Optimization and Practice of RED Network in Image Super Resolution Tasks

Lubin Liu

College of Electronics and Information Engineering, Shenzhen University, Shenzhen, China

Abstract: This study focuses on image super-resolution (ISR) technology, particularly exploring the application and optimization of RED networks in ISR tasks. Firstly, a RED network-based image super-resolution system was designed and implemented, verifying the effectiveness and reliability of the RED network in ISR tasks. Secondly, by adjusting the network structure and parameter configuration, the impact of different settings on model performance was analyzed in depth. The experimental results show that the RED network performs well in image super-resolution tasks, and appropriate parameter adjustments can significantly improve network performance. Finally, this study also discussed the optimization direction of RED network, providing reference for future research.

Keywords: Image super-resolution; RED network; Network optimization; Parameter adjustment; Performance analysis.

1. Introduction

In the vast field of computer vision, Image Super Resolution (ISR) technology is like a shining pearl, attracting the attention of many researchers with its unique charm and wide application prospects. The core goal of ISR technology is to restore high-resolution images from low resolution images. This technology not only has profound research value in academia, but also demonstrates enormous potential and value in various practical application scenarios such as medical imaging, video surveillance, and remote sensing images.

RED network (Residual Encoder Decoder Networks), as an effective model of deep learning in the field of image super-resolution, stands out among many ISR methods due to its unique network architecture and excellent performance. The RED network adopts a symmetrical encoder decoder structure, which achieves a gradual mapping from low resolution images to high-resolution images through the alternating use of convolutional and deconvolution layers. However, despite the significant performance advantages of the RED network, how to further optimize it in practical applications with limited computing resources to improve its processing speed and efficiency remains a key issue that urgently needs to be addressed.

2. Experimental purpose and requirements

The purpose of this experiment is to explore in depth the application and optimization of RED network in image super-resolution tasks. The specific experimental objectives are as follows:

Firstly, by designing and implementing an image super-resolution system based on the RED network, the effectiveness and reliability of the RED network in ISR tasks are verified. The achievement of this goal not only helps to deepen the understanding of the working principle of the RED network, but also provides a solid experimental foundation for subsequent optimization work. Secondly, analyze the impact of the architecture of the RED network on its performance. The unique symmetrical structure and skip connections mechanism of the RED network are important reasons for its superior performance. Through experiments, we will explore in detail how these structural characteristics affect the performance of the network and attempt to find the optimal architecture configuration.

In addition, the practical application of deep learning in the field of image enhancement is also one of the important goals of this experiment. As an important means of image enhancement, image super-resolution directly reflects the potential of deep learning technology in the field of image processing. The practical application of RED network will further verify the feasibility and effectiveness of deep learning in the field of image enhancement.

Finally, exploring the optimization and improvement directions of the RED network is another important task of this experiment. In response to the limitations of the RED network in terms of computing resources and processing speed, efforts will be made to improve its performance by simplifying the network structure and optimizing algorithm parameters. At the same time, we will also pay attention to the latest deep learning technologies and research results, in order to achieve further innovation and breakthroughs on the basis of the RED network.

3. Model Architecture and Innovation

The RED network is renowned for its unique symmetrical structure and jumper connection mechanism. In the RED network, the convolutional layer is responsible for extracting abstract features of the image, while the deconvolution layer is responsible for amplifying and restoring these features into high-resolution images. This structure enables the RED network to achieve efficient feature extraction and image reconstruction while preserving image details. However, the original RED network structure is relatively complex and has many layers, which to some extent limits its application in scenarios with limited computing resources. To address this issue, the original RED network was streamlined and optimized in this experiment (Figure 1).

Specifically, a lightweight RED network model was constructed by reducing the number of layers in the network from 30 to 10. This simplification not only reduces the computational complexity of the network, but also improves

its processing speed and efficiency. Meanwhile, in order to maintain the performance advantage of the streamlined network, the jumper connection mechanism in the RED network has been retained. Jumping connections achieve residual learning by adding the features of convolutional and deconvolution layers, which helps accelerate the learning speed of the network and improve performance. In the streamlined network, jumper connections still play an important role, allowing the network to maintain high reconstruction quality with fewer layers. In addition to streamlining the network structure and retaining jumper connections, other optimization methods have also been attempted to improve the performance of the RED network. For example, more efficient convolution algorithms and advanced optimizers were used to accelerate the training process of the network. At the same time, detailed adjustments and optimizations were made to the network parameters in order to further reduce computational complexity while maintaining performance.

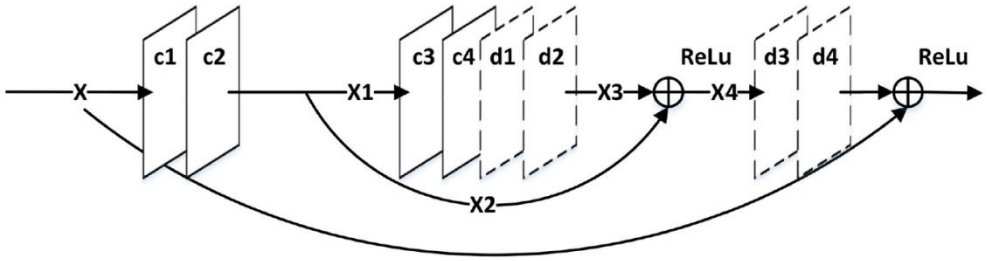

Fig.1 Schematic diagram of RED structure

4. Experimental core code

4.1. Data section

4.1.1. Dataset preparation process

In image super-resolution tasks, dataset preparation is a crucial step. In order to efficiently store and process large-scale image data, H5 format is adopted as the data storage method. The H5 format has been widely used in the field of big data processing due to its efficient data compression and fast read and write speeds. In the process of preparing the dataset, two functions, train and eval, were defined to process the images in the training and testing sets.

```
import h5py
import numpy as np
from PIL import Image as pil_image
import glob
def train(args):
h5_file=h5py.File(args.output_path,'w')
lr_patches=[]
hr_patches=[]
for image_path in
sorted(glob.glob(f'{args.images_dir}/*')
):
hr=pil_image.open(image_path).convert('R
GB')
hr_width=(hr.width//args.scale)*args.sca
le
hr_height=(hr.height//args.scale)*args.s
cale
hr=hr.resize((hr_width,hr_height),resamp
le=pil_image.BICUBIC)
lr=hr.resize((hr_width//args.scale,hr_he
ight//args.scale),resample=pil_image.BIC
UBIC)
lr=lr.resize((lr.width*args.scale,lr.height*args.s
cale),resample=pil_image.BICUBIC)
hr=np.array(hr).astype(np.float32)
lr=np.array(lr).astype(np.float32)
hr=convert_rgb_to_y(hr)
lr=convert_rgb_to_y(lr)
```

```
for i in range(0,lr.shape[0]-
args.patch_size+1,args.stride):
for j in range(0,lr.shape[1]-
args.patch_size+1,args.stride):
lr_patches.append(lr[i:i+args.patch_size
,j:j+args.patch_size])
hr_patches.append(hr[i:i+args.patch_size
,j:j+args.patch_size])
lr_patches=np.array(lr_patches)
hr_patches=np.array(hr_patches)
h5_file.create_dataset('lr',data=lr_patc
hes)
h5_file.create_dataset('hr',data=hr_patc
hes)
h5_file.close()
def eval(args):
h5_file=h5py.File(args.output_path,'w')
lr_group=h5_file.create_group('lr')
hr_group=h5_file.create_group('hr')
for i,image_path in
enumerate(sorted(glob.glob(f'{args.image
s_dir}/*'))):
hr=pil_image.open(image_path).convert('R
GB')
hr_width=(hr.width//args.scale)*args.sca
le
hr_height=(hr.height//args.scale)*args.scale
hr=hr.resize((hr_width,hr_height),resamp
le=pil_image.BICUBIC)
lr=hr.resize((hr_width//args.scale,hr_he
ight//args.scale),resample=pil_image.BIC
UBIC)
lr=lr.resize((lr.width*args.scale,lr.hei
ght*args.scale),resample=pil_image.BICUB
IC)
hr=np.array(hr).astype(np.float32)
lr=np.array(lr).astype(np.float32)
hr=convert_rgb_to_y(hr)
lr=convert_rgb_to_y(lr)
lr_group.create_dataset(str(i),data=lr)
hr_group.create_dataset(str(i),data=hr)
h5_file.close()
```

4.1.2. Run the program to generate a dataset

The train and eval functions first read the original image and perform a series of preprocessing operations, including image scaling and color space conversion. Then, divide the processed image into multiple patches and store these patches in an H5 file. For the training set, all small blocks are integrated into one dataset for subsequent batch processing. For the test set, each small block of the image is stored separately for independent performance evaluation.

```
if__name__=='__main__':
parser=argparse.ArgumentParser()
parser.add_argument('--images-
dir',type=str,required=True,help='图像目录
路径')
parser.add_argument('--output-
path',type=str,required=True,help='输出
H5 文件路径')
parser.add_argument('--
eval',action='store_true',default=False,hel
p='是否处理测试集')
parser.add_argument('--patch-
size',type=int,default=40,help='切割图像块
的大小')
parser.add_argument('--
stride',type=int,default=14,help='切割图像
块的步幅')
parser.add_argument('--
scale',type=int,default=4,help='图像缩放比
例')
args=parser.parse_args()
if not args.eval:
train(args)
else:
eval(args)
```

4.2. Model section

The core of the RED network lies in the combination of convolutional and deconvolution layers, achieving precise mapping from low resolution to high-resolution images through multi-level feature extraction and image reconstruction. In the implementation of the RED network, image features are extracted by deepening convolutional layers layer by layer, followed by gradually enlarging the feature map and restoring image details through deconvolution layers. In addition, the RED network also introduces a jumper connection mechanism, which integrates low-level features with high-level features, effectively improving the network's reconstruction ability and training efficiency.

```
import torch
import torch.nn as nn
class SRCNN(nn.Module):
def__init__(self,num_channels=1):
super(SRCNN,self).__init__()
self.conv1=nn.Conv2d(num_channels,32,ker
nel_size=7)
self.conv2=nn.Conv2d(32,48,kernel_size=7
)
self.conv3=nn.Conv2d(48,64,kernel_size=5
)
self.conv4=nn.Conv2d(64,96,kernel_size=5
)
self.conv5=nn.Conv2d(96,128,kernel_size=
3)
self.deconv1=nn.ConvTranspose2d(128,96,k
ernel_size=3)
self.deconv2=nn.ConvTranspose2d(96,64,ke
rnel_size=5)
self.deconv3=nn.ConvTranspose2d(64,48,ke
rnel_size=5)
self.deconv4=nn.ConvTranspose2d(48,32,ke
rnel_size=7)
self.deconv5=nn.ConvTranspose2d(32,num_c
hannels,kernel_size=7)
self.relu=nn.ReLU(inplace=True)
def forward(self,x):
x0=x
x=self.relu(self.conv1(x))
x=self.relu(self.conv2(x))
x2=x
x=self.relu(self.conv3(x))
x=self.relu(self.conv4(x))
x4=x
x=self.relu(self.conv5(x))
x=self.relu(self.deconv1(x)+x4)
x=self.relu(self.deconv2(x))
x=self.relu(self.deconv3(x)+x2)
x=self.relu(self.deconv4(x))
x=self.relu(self.deconv5(x)+x0)
return x
```

4.3. Training Part

4.3.1. Training Preparation

In the training preparation stage, a series of initial parameters were first set through the command-line parameter parser, including the path of the training and testing datasets, output directory, image scaling ratio, learning rate, batch size, training rounds, number of worker threads, and random seeds. These parameters provide necessary configuration information for the training process. Subsequently, a loss function and optimizer were constructed, and the mean squared error loss function (MSELoss) was selected as the optimization objective. The Adam optimizer was used to update the network parameters. In order to further improve the training effect, a learning rate adjustment strategy was introduced, using an exponential decay learning rate scheduler (Exponential LR) to dynamically adjust the learning rate in order to achieve better performance during the training process.

```
import torch.optim as optim
from torch.utils.data import DataLoader
from tqdm import tqdm
from averagemeter import AverageMeter
#初始参数设定
parser=argparse.ArgumentParser()
parser.add_argument('--train-
file',type=str,default='dataset/train_da
ta.h5')
parser.add_argument('--eval-
file',type=str,default='dataset/test_dat
a.h5')
parser.add_argument('--outputs-
dir',type=str,default='Result')
parser.add_argument('--
scale',type=int,default=4)
parser.add_argument('--
lr',type=float,default=0.8*1e-4)
```

```
parser.add_argument('--batch-
size',type=int,default=16)
parser.add_argument('--num-
epochs',type=int,default=15)
parser.add_argument('--num-
workers',type=int,default=8)
parser.add_argument('--
seed',type=int,default=123)
args=parser.parse_args()
args.outputs_dir=os.path.join(args.outpu
ts_dir,'x{}'.format(args.scale))
if not os.path.exists(args.outputs_dir):
os.makedirs(args.outputs_dir)
#损失函数和优化器
criterion=nn.MSELoss()
optimizer=optim.Adam(model.parameters(),
lr=args.lr)
#学习率调整策略
step_schedule =
torch.optim.lr_scheduler.ExponentialLR(optimizer,
gamma=0.9)
```

4.3.2. Data preprocessing

In the training part of image super-resolution tasks, data preprocessing is a crucial step. In order to efficiently and orderly load and process data, the DataLoader component in the PyTorch framework was adopted. For the training set, an instance of the TrainData class was created and passed to DataLoader, with parameters set such as batch size, whether to shuffle data, number of worker threads, whether to use lock page memory, and whether to discard the last incomplete batch. Similarly, for the test set, an instance of the EvalDataset class was created and the corresponding DataLoader was configured. These preprocessing steps ensure that the data can be loaded into the model in the predetermined manner, providing a solid foundation for subsequent training and evaluation.

```
train_dataset=TrainDataset(args.train_fi
le)
train_dataloader=DataLoader(dataset=trai
n_dataset,
batch_size=args.batch_size,
shuffle=True,
num_workers=args.num_workers,
pin_memory=True,
drop_last=True)
eval_dataset=EvalDataset(args.eval_file)
eval_dataloader=DataLoader(dataset=eval_
dataset,batch_size=1)
```

4.3.3. Training process

The training process is the core part of image super-resolution tasks, involving iterative optimization and performance evaluation of the model. Firstly, set the initial values for the optimal weight, optimal round, and highest peak signal-to-noise ratio (PSNR). Then, through multiple training iterations, the model parameters are continuously adjusted to minimize the loss function. In each round, the average loss is recorded and a learning rate scheduling strategy is adopted to dynamically adjust the learning rate. In addition, performance evaluations were conducted on the model at the end of each round, and the average PSNR on the validation set was calculated. Finally, save the model weights for each round and update the best weight, best round, and highest PSNR for subsequent analysis and comparison.

```
best_weights=copy.deepcopy(model.state_d
ict())
best_epoch=0
best_psnr=0.0
for epoch in range(args.num_epochs):
model.train()
epoch_losses=AverageMeter()
with tqdm(total=(len(train_dataset)-
len(train_dataset)%args.batch_size))as
t:
t.set_description('epoch:{}/{}'.format(e
poch,args.num_epochs-1))
for data in train_dataloader:
inputs,labels=data
inputs=inputs.to(device)
labels=labels.to(device)
preds=model(inputs)
loss=criterion(preds,labels)
epoch_losses.update(loss.item(),len(inpu
ts))
optimizer.zero_grad()
loss.backward()
optimizer.step()
t.set_postfix(loss='{:.6f}'.format(epoch
_losses.avg))
t.update(len(inputs))
torch.save(model.state_dict(),os.path.jo
in(args.outputs_dir,'epoch_{}.pth'.forma
t(epoch)))
step_schedule.step()
model.eval()
epoch_psnr=AverageMeter()
for data in eval_dataloader:
inputs,labels=data
inputs=inputs.to(device)
labels=labels.to(device)
with torch.no_grad():
preds=model(inputs).clamp(0.0,1.0)
epoch_psnr.update(calc_psnr(preds,labels
),len(inputs))
print('eval
psnr:{:.2f}'.format(epoch_psnr.avg))
if epoch_psnr.avg>best_psnr:
best_epoch=epoch
best_psnr=epoch_psnr.avg
best_weights=copy.deepcopy(model.state_d
ict())
print('best
epoch:{},psnr:{:.2f}'.format(best_epoch,
best_psnr))
torch.save(best_weights,os.path.join(arg
s.outputs_dir,'best.pth'))
```

4.4. Testing section

In the testing part of the image super-resolution task, the validation phase aims to evaluate the performance of the trained model on unseen data. Specifically, set the model to evaluation mode and use the validation dataset for inference. During the inference process, gradient calculation was turned off to accelerate the calculation process and ensure that the range of output values is within a reasonable range. Subsequently, the peak signal-to-noise ratio (PSNR) between the model prediction results and the true labels was calculated, and the Average Meter class was used to accumulate and average the PSNR. Finally, the average PSNR on the

validation set is output to quantify the performance of the model in image super-resolution tasks.

```
model.eval()
epoch_psnr=AverageMeter()
for data in eval_dataloader:
inputs,labels=data
inputs=inputs.to(device)
labels=labels.to(device)
with torch.no_grad():
preds=model(inputs).clamp(0.0, 1.0)

epoch_psnr.update(calc_psnr(preds,labels
),len(inputs))
print('eval
psnr:{:.2f}'.format(epoch_psnr.avg))
```

5. Experimental procedure

The experimental data records are shown in Table 1, which displays the PSNR values under different network structures and parameter settings. The experimental results show that different network structures and parameter configurations can lead to significant differences in PSNR.

Tab.1 Experimental Data Record

Test No	Net.Struc.	Change	Batch_size	Rounds	Optimizer	Learning rate	PSNR
1	SRCNN	_	16	60	adam	0.0001	30.17
2	SRCNN	Add BN layers to each layer	16	60	adam	0.0001	29.79
...	...	...	...	...	...	...	...
28	RED(10)	Batch size changed to 16, changed to 64, 48, 64, 96, 128	16	50	adam	0.0008	30.49

From the experimental data recorded in Table 1, it can be found that network structure and parameter settings have a significant impact on the performance of image super-resolution tasks. In the SRCNN network, after adding a batch normalization (bn) layer, although other parameters remain unchanged, the PSNR value decreases, indicating that the bn layer does not always have a positive impact on performance. In the RED (10) network, higher PSNR values were obtained by adjusting batch size and channel number, indicating that appropriate parameter adjustments can effectively improve network performance. Overall, these experimental data provide valuable optimization basis and contribute to a deeper understanding of the performance of image super-resolution tasks.

6. Experimental Summary

In this experiment, the RED network model was designed and trained to successfully achieve super-resolution reconstruction of images. The experimental results show that the RED network performs well in image super-resolution tasks and can effectively improve image resolution. At the same time, the experiment also found that factors such as the direct correlation between input and output, limitations of BN layers, limitations on the number of fully connected layers, and optimization of receptive fields have a significant impact on model performance. Future work will further explore the optimization and improvement directions of RED networks to enhance their performance in image super-resolution tasks.

References

[1] Peng Yanfei, Li Yongxin, Meng Xin, Cui Yun Image super-resolution reconstruction using pyramid variance pooling network [J]. Liquid Crystal and Display, 2024, 39 (10): 1380-1390

[2] Bi Xiuping, Chen Shi, Zhang Lefei Blueprint Separable Convolutional Transformer Network for Lightweight Image Super Resolution [J]. Chinese Journal of Image and Graphics, 2024, 29 (04): 875-889

[3] Shu Zhong, Zheng Boer Research on Super Resolution Distortion Control Image Reconstruction Based on Convolutional Neural Networks [J]. Packaging Engineering, 2024, 45 (07): 222-233

[4] Zhou Ying, Pei Shenghu, Chen Haiyong, Xu Shibo Image super-resolution network based on multi-scale adaptive attention [J]. Optical Precision Engineering, 2024, 32 (06): 843-856

[5] Cheng Deqiang, Yuan Hang, Qian Jiansheng, Kou Qiqi, Jiang He Image super-resolution algorithm based on deep feature differentiation network [J]. Chinese Journal of Electronics and Information Technology, 2024, 46 (03): 1033-1042

[6] Zhao Xiaoqiang, Wang Ze, Song Zhaoyang, Jiang Hongmei Image super-resolution reconstruction based on dynamic attention network [J]. Journal of Zhejiang University (Engineering Edition), 2023, 57 (08): 1487-1494