AD18511 – DEEP LEARNING LABORATORY

DATE:

EX.NO: 12 **IMPLEMENTATION OF RESNET ARCHITECTURE.**

AIM:

To implement the ResNet architecture using Tensorflow.

DESCRIPTION:

- A novel architecture called **Residual Network** was launched by Microsoft Research experts in 2015 with the proposal of **ResNet.**
- The Residual Blocks idea was created by this design to address the issue of the vanishing/exploding gradient.
- The method called skip connection is applied in this network.It bypasses some levels in between to link-layer activations to subsequent layers.
- This creates a leftover block. These leftover blocks are stacked to create resnets.
- Pooling Layers: Between residual blocks, ResNet often includes pooling layers (e.g., max-pooling) to downsample feature maps and reduce spatial dimensions.
- Fully Connected Layer: Towards the end of the network, there is typically a fully connected layer or global average pooling layer to flatten the feature maps and produce class scores in the case of image classification tasks.
- Output Layer: The final layer produces the network's output, which could be class probabilities for classification tasks or regression values for regression tasks.
- Final Activation Function: The output layer is often followed by a softmax activation for classification tasks or a linear activation for regression tasks.

PROGRAM:

```
!pip install torch torchvision scikit-learn

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from sklearn.metrics import classification_report,confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

#define
class ResidualBlock(nn.Module):
    def_init_(self,in_channels,out_channels,stride=1):
        super(ResidualBlock,self)._init_()
        self.conv1=nn.Conv2d(in_channels,out_channels,kernel_size=3,stride=stride,padding=1,bias=False)
        self.bn1=nn.BatchNorm2d(out_channels)
        self.relu=nn.ReLU()
        self.conv2=nn.Conv2d(out_channels,out_channels,kernel_size=3,stride=1,padding=1,bias=False)
        self.bn2=nn.BatchNorm2d(out_channels)
        self.downsample=None
```

```
        if in_channels!=out_channels or stride!=1:
            self.downsample=nn.Sequential(
                nn.Conv2d(in_channels,out_channels,kernel_size=1,stride=stride,bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self,x):
        residual=x

        out=self.conv1(x)
        out=self.bn1(out)
        out=self.relu(out)

        out=self.conv2(out)
        out=self.bn2(out)

        if self.downsample is not None:
            residual=self.downsample(x)


        out+=residual
        out=self.relu(out)

        return out

#define the resnet architecture
class ResNet(nn.Module):
    def _init_(self,block,num_blocks,num_classes=10):
        super(ResNet,self)._init_()
        self.in_channels=16
        self.conv1=nn.Conv2d(3,16,kernel_size=3,stride=1,padding=1,bias=False)
        self.bn=nn.BatchNorm2d(16)
        self.relu=nn.ReLU()
        self.layer1=self.make_layer(block,16,num_blocks[0],stride=1)
        self.layer2=self.make_layer(block,32,num_blocks[1],stride=2)
        self.avg_pool=nn.AdaptiveAvgPool2d((1,1))
        self.fc=nn.Linear(32,num_classes)

    def make_layer(self,block,out_channels,num_blocks,stride):
        layers=[block(self.in_channels,out_channels,stride)]
        self.in_channels=out_channels
        for _ in range(1,num_blocks):
            layers.append(block(out_channels,out_channels,stride=1))
        return nn.Sequential(*layers)

    def forward(self,x):
        out=self.conv1(x)
        out=self.bn(out)
        out=self.relu(out)
        out=self.layer1(out)
        out=self.layer2(out)
        out=self.avg_pool(out)
        out=out.view(out.size(0),-1)
```

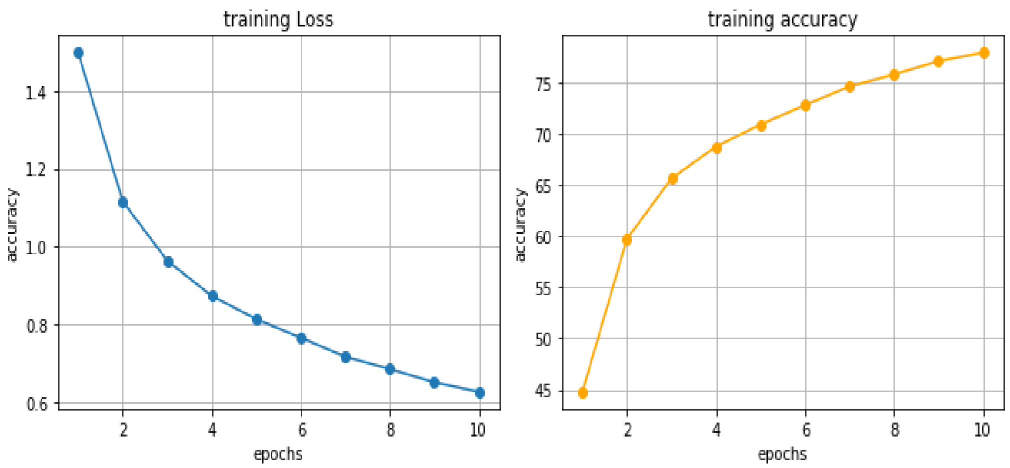
```
        out=self.fc(out)
        return out

def ResNet18():
    return ResNet(ResidualBlock,[2,2])


def train_model(model,trainloader,criterion,optimizer,num_epochs=10):
    model.train()
    train_losses=[]
    train_accuracies=[]

    for epoch in range(num_epochs):
        running_loss=0.0
        correct=0
        total=0

        for data in trainloader:
            inputs,labels=data
            optimizer.zero_grad()
            outputs=model(inputs)
            loss=criterion(outputs,labels)
            loss.backward()
            optimizer.step()


            running_loss+=loss.item()
            _,predicted=torch.max(outputs.data,1)
            total+=labels.size(0)
            correct+=(predicted==labels).sum().item()


        train_loss=running_loss/len(trainloader)
        train_accuracy=100*correct/total
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)


        print(f'Epoch{epoch+1}/{num_epochs},Loss:{train_loss:.4f},Accuracy:{train_accuracy:.2f}%')
    return model,train_losses,train_accuracies

transform=transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))])
trainset=torchvision.datasets.CIFAR10(root='./data',train=True,download=True,transform=transform)
trainloader=torch.utils.data.DataLoader(trainset,batch_size=64,shuffle=True)


resnet=ResNet18()

criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(resnet.parameters(),lr=0.01,momentum=0.9)

resnet,train_losses,train_accuarcies=train_model(resnet,trainloader,criterion,optimizer,num_epochs=10)
```

OUTPUT:

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:18<00:00, 9283771.40it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
Epoch1/10,Loss:1.4989,Accuracy:44.78%
Epoch2/10,Loss:1.1158,Accuracy:59.72%
Epoch3/10,Loss:0.9633,Accuracy:65.63%
Epoch4/10,Loss:0.8727,Accuracy:68.76%
Epoch5/10,Loss:0.8130,Accuracy:70.90%
Epoch6/10,Loss:0.7650,Accuracy:72.80%
Epoch7/10,Loss:0.7162,Accuracy:74.63%
Epoch8/10,Loss:0.6845,Accuracy:75.81%
Epoch9/10,Loss:0.6504,Accuracy:77.12%
Epoch10/10,Loss:0.6258,Accuracy:77.94%
```

```
num_epochs=10
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(range(1,num_epochs+1),train_losses,marker='o')
plt.title('training Loss')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.grid()


plt.subplot(1,2,2)
plt.plot(range(1,num_epochs+1),train_accuarcies,marker='o',color='orange')
plt.title('training accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.grid()

plt.tight_layout()
plt.show()

trainset=torchvision.datasets.CIFAR10(root='./data',train=False,download=True,transform=transform)
testloader=torch.utils.data.DataLoader(trainset,batch_size=64,shuffle=False)

resnet.eval()
all_preds=[]
all_labels=[]

with torch.no_grad():
    for data in testloader:
        inputs,labels=data
        outputs=resnet(inputs)
        _,predicted=torch.max(outputs,1)
        all_preds.extend(predicted.tolist())
        all_labels.extend(labels.tolist())


print(classification_report(all_labels,all_preds))
```

OUTPUT:



Files already

downloade

d and

verified

	precision	recall	f1-score	support
0	0.79	0.72	0.75	1000
1	0.97	0.73	0.83	1000
2	0.77	0.41	0.53	1000
3	0.53	0.46	0.49	1000
4	0.77	0.62	0.69	1000
5	0.45	0.85	0.59	1000
6	0.69	0.86	0.77	1000
7	0.87	0.66	0.75	1000
8	0.86	0.85	0.85	1000
9	0.76	0.93	0.83	1000
accuracy			0.71	10000
macro avg	0.74	0.71	0.71	10000
weighted avg	0.74	0.71	0.71	10000

RESULT:

The ResNet architecture is implemented and the model is trained and tested successfully.