# 10/19/24, 10:47 AM ML-Final-Mini - Project-CNN-DNN Train.ipynb - Colab

importtorch importtorch.nnasnn importtorch.optimasoptim fromtorch.utils.dataimportDataLoader fromtorchvisionimportdatasets,transforms fromtorchvision.modelsimportresnet18,densenet121,squeezenet1\_0

#Transformationsforthedataaugmentation transform=transforms.Compose([transforms.Resize((224,224)), transforms.ToTensor(), transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))])

#Loadsdataset train\_dataset=datasets.CIFAR10(root='/content/All',train=True,download=True,transform=transform) test\_dataset=datasets.CIFAR10(root='/content/All',train=False,download=True,transform=transform) train\_loader=DataLoader(train\_dataset,batch\_size=64,shuffle=True) test\_loader=DataLoader(test\_dataset,batch\_size=64,shuffle=False)

#Loadthemodels resnet\_model=resnet18(pretrained=False,num\_classes=10) densenet\_model=densenet121(pretrained=False,num\_classes=10) squeezenet\_model=squeezenet1\_0(pretrained=False) squeezenet\_model.classifier[1]=nn.Conv2d(512,10,kernel\_size=(1,1),stride=(1,1))#Modifyfor10classes squeezenet\_model.num\_classes=10

#Selectsdeviceforencoding device=torch.device('cuda'iftorch.cuda.is\_available()else'cpu') resnet\_model,densenet\_model,squeezenet\_model=resnet\_model.to(device),densenet\_model.to(device),squeezenet\_model.to(device)

#Lossandoptimizer criterion=nn.CrossEntropyLoss() optimizer\_resnet=optim.Adam(resnet\_model.parameters(),lr=0.001) optimizer\_densenet=optim.Adam(densenet\_model.parameters(),lr=0.001) optimizer\_squeezenet=optim.Adam(squeezenet\_model.parameters(),lr=0.001)

#Trainingloop(wecanimplementitonanymodel) deftrain\_model(model,train\_loader,criterion,optimizer,device):

model.train() forepochinrange(10):#wearerunningfor10epochs running\_loss=0.0 fori,(inputs,labels)inenumerate(train\_loader): inputs,labels=inputs.to(device),labels.to(device)

#Zerogradients optimizer.zero\_grad()

#Forwardpass outputs=model(inputs) loss=criterion(outputs,labels)

#Backwardpasss loss.backward() optimizer.step()

running\_loss+=loss.item() ifi%100==99:#Printevery100batches print(f'Epoch[{epoch+1}],Step[{i+1}],Loss:{running\_loss/100:.4f}') running\_loss=0.0

#Testloop(wecanimplementitonanymodel) deftest\_model(model,test\_loader,device): model.eval()#Setstoevaluationmode correct=0 total=0 withtorch.no\_grad(): forinputs,labelsintest\_loader:

inputs,labels=inputs.to(device),labels.to(device) outputs=model(inputs)

\_,predicted=torch.max(outputs.data,1) total+=labels.size(0) correct+=(predicted==labels).sum().item() print(f'TestAccuracy:{100\*correct/total}%')

#TrainsandtestsResNetmodel print("TrainingResNet...") train\_model(resnet\_model,train\_loader,criterion,optimizer\_resnet,device) test\_model(resnet\_model,test\_loader,device)

#TrainsandtestsDenseNetmodel print("\nTrainingDenseNet...")

i d l(d d l i l d i i i i d d i )

# https://colab.research.google.com/drive/1yspMtg3FkVTDvqdoIH6lg4PPXsEIUURK#printMode=true 1/2

10/19/24, 10:47 AM ML-Final-Mini - Project-CNN-DNN Train.ipynb - Colab train\_model(densenet\_model,train\_loader,criterion,optimizer\_densenet,device) test\_model(densenet\_model,test\_loader,device)

#TrainsandtestsSqueezeNetmodel print("\nTrainingSqueezeNet...") train\_model(squeezenet\_model,train\_loader,criterion,optimizer\_squeezenet,device) test\_model(squeezenet\_model,test\_loader,device)

#Savesthemodels torch.save(resnet\_model.state\_dict(),'resnet\_model.pth') torch.save(densenet\_model.state\_dict(),'densenet\_model.pth') torch.save(squeezenet\_model.state\_dict(),'squeezenet\_model.pth')