

**计算机视觉作业报告**

|  |  |
| --- | --- |
| 作业名称： | Learning CNN |
| 姓 名： | 胡单春 |
| 学 号： | 21921082 |
| 电子邮箱： | 3150102279@zju.edu.cn |
| 联系电话： | 15724998468 |
| 导 师： | 邵健 |

2019年 12 月 26 日

**Learning CNN**

1. 作业已实现的功能简述及运行简要说明

**已实现功能：**

1：使用pytorch实现最基本的卷积神经网络(CNN) LeNet-5以及一个物体分类的CNN。

2：用实现的LeNet-5在MNIST数据集上训练和测试。

3：用实现的物体分类CNN在CIFAR-10数据集上训练和测试。

运行简要说明：

1：请安装好python+pytorch环境。确保有requirement.tx中包含的python库。

2：如果要对LeNet-5在MNIST数据集上进行训练和测试，请运行python LeNet5.py。如果是对ResNet18在CIFAR10数据集上进行训练和测试，请运行python resnet.py

3: 如果想使用LeNet-5对图片进行分类，请运行python lenet\_predict.py -model\_path= LetNet5\_model.pth -img\_path=待检测图片路径；如果想使用ResNet18对图片进行分类，请运行python resnet\_predict.py -model\_path= ResNet18\_model.pth -img\_path=待检测图片路径

1. 作业的开发与运行环境

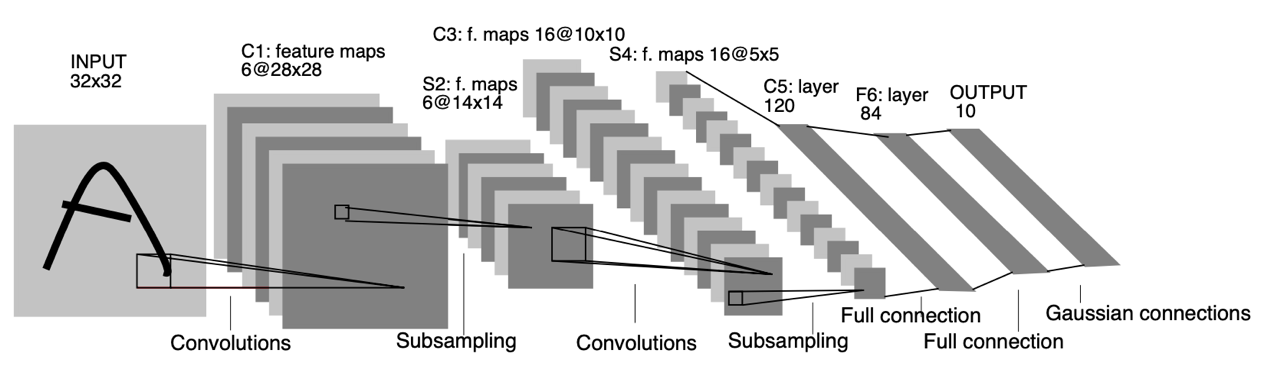
系统版本：macOS Catalina 10.15.2

python版本：3.7.5

python依赖环境：详见hw3文件夹下的requirements.txt

1. LeNet-5与ResNet50

LeNet-5：



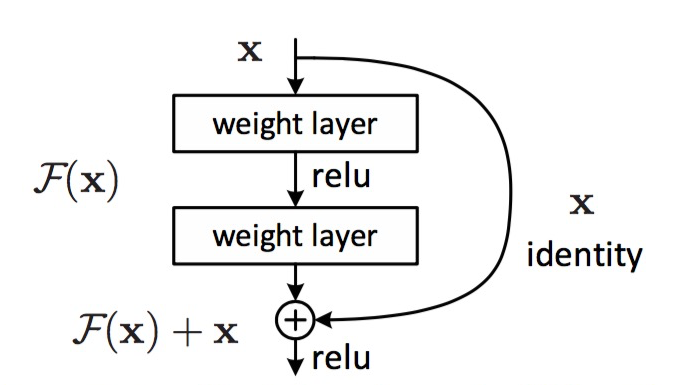
上图中的LeNet-5网络结构由Yann LeCun, Leon Bottou等人提出。

LeNet-5接收输入为32\*32的1通道的图片数据，首先经过一层6个核大小为5\*5的卷积层C1，因此C1层输出为6通道的28\*28((32-5+0)/1+1)的大小。然后S2层是采样层，在进行采样前，C1层的输出会经过一次非线性激活函数，原论文中是用的sigmoid函数，本次采用relu函数。S2层是池化层，大小为2\*2，步长为2，无padding。经过S2后的输出((28-2+0)/2+1=14)作为卷积层C3的输入。C3有16个特征提取器，卷积核大小为5\*5，步长为1。然后C3的输出((14-5+0/1)+1=10)同样作为relu函数的输入，之后再经过采样器S4，核大小2\*2，步长为2，无padding。S4的输出((10-2+0)/2+1=5)作为C5的输入。在原论文中C5是作为卷积层的，其卷积核大小为5\*5，步长为1，无填充，则生成的feature map大小为(5-5+0)/1+1=1\*1。可以简化为一层全连接层。F6是一个全连接层。最后OUTPUT层则输出维度为C（C为类别总数）的数据，代表图片在每一个类别上的概率。

ResNet50：

ResNet的提出是为了解决深度训练中存在的退化问题，即随着模型的深度加深，学习能力得到增强，在理论上，更深的模型不应该会产生比它浅的模型更高的错误率，然而在实际中，却存在着深度更大的模型错误率更高的情况。这就是“退化”问题，主要是因为模型变复杂时，随机梯度下降SGD的优化变得困难，导致模型达不到好的学习效果。

针对这个问题，ResNet的作者团队引入深度残差学习框架，提出了一个Residual结构，如图1。



**图1 残差学习：a building block**

通过恒等映射（identity mapping），构建新层。将原始所需要学的函数H(x)转化为F(x)+x，其中x为输入。作者假设F(x)的优化会比H(x)简单的多。

公式(3.1)

公式(3.2)

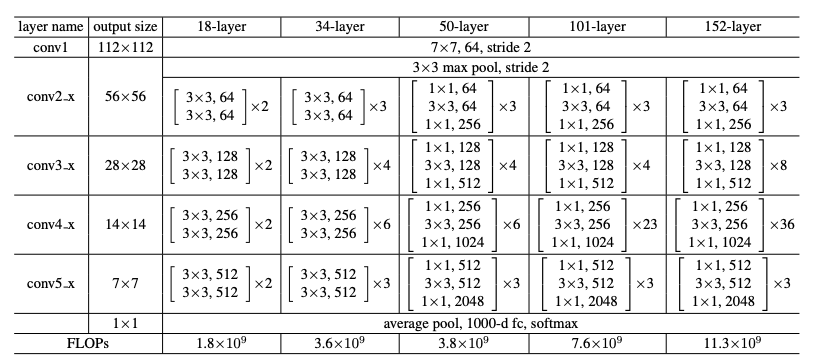
公式(3.1)表示图1中的block如何根据输入x得到输出y。

公式(3.2)表示图1中的F如何计算。在得到输出y后，再使用一个非线性函数。

在公式(3.1)中，由于F+x是通过element-wise加法获得，所以需要保证x和F的维度相等。如果在计算F时改变了输出/输出的通道数，那么需要对x进行线性变换，即通过公式(3.3)。

公式(3.3)

论文中的网络结构如图2。



**图2 网络结构参数**

由于原论文中ResNet18网络第一层网络参数是针对ImageNet的输入224\*224\*3的，而CIFAR10是32\*32\*3的，所以需要对参数进行修改。将第一层卷积层参数kernel\_size=7，stride=2，padding=3改为kernel\_size=3，stride=1，padding=1，第一个池化层参数kernel\_size=3，stride=2，padding=1改为kernel\_size=3，stride=1，padding=1。

1. 具体实现

LeNet-5：

**网络定义：**

|  |
| --- |
| **class** **LeNet5**(nn.Module):  **def** \_\_init\_\_(self):  super(LeNet5, self).\_\_init\_\_()  self.conv1 = nn.Conv2d(1, 6, 5, padding=2)  self.conv2 = nn.Conv2d(6, 16, 5)  self.fc1 = nn.Linear(16\*5\*5, 120)  self.fc2 = nn.Linear(120, 84)  self.fc3 = nn.Linear(84, 10)    **def** forward(self, x):  x = F.max\_pool2d(F.relu(self.conv1(x)), (2, 2))  x = F.max\_pool2d(F.relu(self.conv2(x)), (2, 2))  x = x.view(x.size()[0], -1)  x = F.relu(self.fc1(x))  x = F.relu(self.fc2(x))  x = self.fc3(x)    **return** x |

MNIST数据集中每张图片大小为28\*28\*1，通过2层卷积层和3层全连接层获得在每个类上的概率。

**数据集定义与加载：**

使用torchvision内置的datasets.MNIST获得MNIST数据集，并使用torch.utils.data.DataLoader加载数据集。使用transforms.Normalize()将数据标准化。

|  |
| --- |
| train\_loader = torch.utils.data.DataLoader(datasets.MNIST('data', train=True, download=True, transform=transforms.Compose([  transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,)),  ])),  batch\_size=batch\_size,  shuffle=True)  test\_loader = torch.utils.data.DataLoader(datasets.MNIST('data', train=False, transform=transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.1307,), (0.3081,))  ])),  batch\_size=batch\_size,  shuffle=False) |

**训练参数设置：**

定义每次训练的batch\_size、训练次数epoch、运行设备device、损失函数criterion和优化器optimizer。

|  |
| --- |
| batch\_size=512  epoch=20  device=torch.device("cuda" **if** torch.cuda.is\_available() **else** "cpu")  model = LeNet5().to(device)  criterion = nn.CrossEntropyLoss(reduction='sum')  optimizer = optim.Adam(model.parameters(), lr=1e-3, betas=(0.9, 0.99)) |

**训练和评估函数:**

train()函数用来训练模型，并对每次训练完的模型在训练集和测试集上使用eval()函数评估。其中部分训练log会通过tensorboard进行可视化，包括平均训练误差train\_loss、训练集精确度train\_accuracy与测试集精确度test\_accuracy。将训练日志保存在文件lenet5\_train\_log.txt中。

|  |
| --- |
| *# 训练过程*  **def** train(model, device, train\_loader, criterion, optimizer, epoch):  model.train()  running\_loss = 0.  **for** batch\_idx, (data, target) **in** enumerate(train\_loader):  data, target = data.to(device), target.to(device)  optimizer.zero\_grad()  output = model(data)  loss = criterion(output, target)  loss.backward()  running\_loss += loss.item()  optimizer.step()  **if** (batch\_idx+1)%30 == 0:  **print**("Train Epoch: {} [{}/{} ({:.2f}%)]**\t**Loss: {:.6f}".format(  epoch, batch\_idx\*len(data), len(train\_loader.dataset), 100.\*batch\_idx/len(train\_loader), loss.item()))  **with** open('./lenet5\_train\_log.txt', 'a+') **as** f:  f.write("Train Epoch: {} [{}/{} ({:.2f}%)]**\t**Loss: {:.6f}**\n**".format(  epoch, batch\_idx\*len(data), len(train\_loader.dataset), 100.\*batch\_idx/len(train\_loader), loss.item()))  **with** SummaryWriter('./lenet\_scalar') **as** writer:*#自动调用close()*  writer.add\_scalar('lenet\_scalar/train\_loss', running\_loss/len(train\_loader.dataset), epoch)  writer.add\_scalar('lenet\_scalar/train\_accuracy', eval(model, device, train\_loader, criterion), epoch)  writer.add\_scalar('lenet\_scalar/test\_accuracy', eval(model, device, test\_loader, criterion, is\_train=False), epoch)  *# In[9]:*  *# 评估函数*  **def** eval(model, device, test\_loader, criterion, is\_train=True):  model.eval()  test\_loss = 0  correct = 0  **with** torch.no\_grad():  **for** data, target **in** test\_loader:  data, target = data.to(device), target.to(device)  output = model(data)  test\_loss += criterion(output, target).item()  *# 获得最大概率下标*  pred = output.max(1, keepdim=True)[1]  correct += pred.eq(target.view\_as(pred)).sum().item()    test\_loss /= len(test\_loader.dataset)  **if** is\_train:  **print**('**\n**Train Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **with** open('./lenet5\_train\_log.txt', 'a+') **as** f:  f.write('**\n**Train Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **else**:  **print**('**\n**Valid Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **with** open('./lenet5\_train\_log.txt', 'a+') **as** f:  f.write('**\n**Valid Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **return** 100.\*correct/len(test\_loader.dataset) |

**模型保存与使用：**

|  |
| --- |
| torch.save(model, './LetNet5\_model.pth')  **def** predict(model\_path, img\_path):  model = torch.load(model\_path)  device = torch.device('cuda' **if** torch.cuda.is\_available() **else** 'cpu')  model = model.to(device)  model.eval()  img = cv2.imread(img\_path)  img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  trans = transforms.Compose([  transforms.Resize((28,28)),  transforms.ToTensor(),  transforms.Normalize((0.1307,), (0.3081,))  ])  img = trans(img).to(device)  img = img.unsqueeze(0)  output = model(img)  prob = F.softmax(output, dim=1)  pred = prob.max(dim=1)[1].item()  **return** pred |

predict函数通过传入模型文件名和待预测图片文件名，输出预测结果。

ResNet18：

**block的实现：**

|  |
| --- |
| **def** conv3x3(inplanes, planes, stride=1):  **return** nn.Conv2d(inplanes, planes, kernel\_size=3, stride=stride, padding=1, bias=False)  *# In[2]:*  **class** **BasicBlock**(nn.Module):  expansion = 1  **def** \_\_init\_\_(self, inplanes, planes, stride=1, downsample=None):  super(BasicBlock, self).\_\_init\_\_()  self.conv1 = conv3x3(inplanes, planes, stride)  self.bn1 = nn.BatchNorm2d(planes)  self.relu = nn.ReLU(inplace=True)  self.conv2 = conv3x3(planes, planes)  self.bn2 = nn.BatchNorm2d(planes)  self.downsample = downsample  self.stride = stride  **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 |

其中self.conv1、self.conv2都是卷积核大小为3\*3的卷积层，而self.downsample则是希望学习的残差函数。

**ResNet网络定义：**

|  |
| --- |
| **class** **ResNet**(nn.Module):    **def** \_\_init\_\_(self, block, layers, num\_classes=10):  self.inplanes = 64  super(ResNet, self).\_\_init\_\_()  *# 原本kernel\_size=7,stride=2,padding=3，为了适应cifar-10的32\*32,(7,2,3) ->（3, 1, 1）*  self.conv1 = nn.Conv2d(3, 64, kernel\_size=3, stride=1, padding=1,  bias=False)  self.bn1 = nn.BatchNorm2d(64)  self.relu = nn.ReLU(inplace=True)  *# (3,2,1)->(3,1,1)*  self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=1, padding=1)  self.layer1 = self.\_make\_layer(block, 64, layers[0])  self.layer2 = self.\_make\_layer(block, 128, layers[1], stride=2)  self.layer3 = self.\_make\_layer(block, 256, layers[2], stride=2)  self.layer4 = self.\_make\_layer(block, 512, layers[3], stride=2)  *# self.avgpool = nn.AvgPool2d(7, stride=1)*  self.fc = nn.Linear(512 \* block.expansion, num\_classes)    **for** m **in** self.modules():  **if** isinstance(m, nn.Conv2d):  nn.init.kaiming\_normal\_(m.weight, mode='fan\_out', nonlinearity='relu')  **elif** isinstance(m, nn.BatchNorm2d):  nn.init.constant\_(m.weight, 1)  nn.init.constant\_(m.bias, 0)    **def** \_make\_layer(self, block, planes, blocks, stride=1):  downsample = None  **if** stride != 1 **or** self.inplanes != planes \* block.expansion:  downsample = nn.Sequential(  nn.Conv2d(self.inplanes, planes \* block.expansion,  kernel\_size=1, stride=stride, bias=False),  nn.BatchNorm2d(planes \* block.expansion),  )    layers = []  layers.append(block(self.inplanes, planes, stride, downsample))  self.inplanes = planes \* block.expansion  **for** i **in** range(1, blocks):  layers.append(block(self.inplanes, planes))    **return** nn.Sequential(\*layers)    **def** forward(self, x):  x = self.conv1(x)  x = self.bn1(x)  x = self.relu(x)  x = self.maxpool(x)    x = self.layer1(x)  x = self.layer2(x)  x = self.layer3(x)  x = self.layer4(x)    x = F.avg\_pool2d(x, 4)  x = x.view(x.size(0), -1)  x = self.fc(x)    **return** x  **def** ResNet18():  **return** ResNet(BasicBlock, [2, 2, 2, 2]) |

**数据集定义与加载：**

使用torchvision内置的datasets.CIFAR10获得CIFAR10数据集，并使用torch.utils.data.DataLoader加载数据集。分别定义tranform预处理训练集数据与测试集数据。

|  |
| --- |
| train\_transform = transforms.Compose([  transforms.RandomCrop(32, padding=4),  transforms.RandomHorizontalFlip(0.5),  transforms.ToTensor(),  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),  ])  test\_transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),  ])  train\_set = datasets.CIFAR10(root='./data', train=True, download=True, transform=train\_transform)  test\_set = datasets.CIFAR10(root='./data', train=False, download=True, transform=test\_transform)  train\_loader = torch.utils.data.DataLoader(train\_set, batch\_size=batch\_size, shuffle=True)  test\_loader = torch.utils.data.DataLoader(test\_set, batch\_size=batch\_size, shuffle=False) |

**训练参数设置：**

定义每次训练的batch\_size、训练次数epoch、运行设备device、损失函数criterion和优化器optimizer。

|  |
| --- |
| batch\_size=256  epoch=200  device=torch.device("cuda" **if** torch.cuda.is\_available() **else** "cpu")  model = ResNet50().to(device)  save\_model\_path = './ResNet50\_model.pth'  criterion = nn.CrossEntropyLoss(reduction='sum')  optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight\_decay=5e-4)  scheduler = lr\_scheduler.MultiStepLR(optimizer, milestones=[40, 90, 150], gamma=0.2) |

随着epoch的增大，为了降低学习率，得到更好的收敛效果，定义了一个scheduler，在第40次、第90次、第150次的时候将学习率变为原来的0.2。

**训练和评估函数:**

train()函数用来训练模型，并对每次训练完的模型在训练集和测试集上使用eval()函数评估。其中部分训练log会通过tensorboard进行可视化，包括平均训练误差train\_loss、训练集精确度train\_accuracy与测试集精确度test\_accuracy。将训练日志保存在renset\_train\_log.txt中。

|  |
| --- |
| *# 训练过程*  **def** train(model, device, train\_loader, optimizer, scheduler, criterion, num\_epoch):  model.train()  count = 0  best\_valid\_accuray = 0.  best\_epoch\_id = 0  **for** epoch **in** range(1, num\_epoch+1):  running\_loss = 0.  **for** batch\_idx, (data, target) **in** enumerate(train\_loader):  count += 1  data, target = data.to(device), target.to(device)  optimizer.zero\_grad()  output = model(data)  loss = criterion(output, target)  loss.backward()  optimizer.step()  running\_loss += loss.item()  **if** (batch\_idx+1)%30 == 0:  **print**("Train Epoch: {} [{}/{} ({:.2f}%)]**\t**Loss: {:.6f}".format(  epoch, batch\_idx\*len(data), len(train\_loader.dataset), 100.\*batch\_idx/len(train\_loader), loss.item()))  **with** open('./resnet\_train\_log.txt', 'a+') **as** f:  f.write("Train Epoch: {} [{}/{} ({:.2f}%)]**\t**Loss: {:.6f}**\n**".format(  epoch, batch\_idx\*len(data), len(train\_loader.dataset), 100.\*batch\_idx/len(train\_loader), loss.item()))    scheduler.step()    **with** SummaryWriter('./resnet\_scalar') **as** writer:*#自动调用close()*  writer.add\_scalar('resnet\_scalar/train\_loss', running\_loss/len(train\_loader.dataset), epoch)  writer.add\_scalar('resnet\_scalar/train\_accuracy', eval(model, device, train\_loader)[0], epoch)  test\_accuracy, test\_loss = eval(model, device, test\_loader, is\_train=False)  **if** test\_accuracy>best\_valid\_accuray:  best\_valid\_accuray = test\_accuracy  best\_epoch\_id = epoch  writer.add\_scalar('resnet\_scalar/test\_accuracy', test\_accuracy, epoch)  writer.add\_scalar('resnet\_scalar/test\_loss', test\_loss, epoch)  **if** early\_stop(best\_epoch\_id, epoch):  **print**("**\n**Early Stop at Epoch {}, Accuracy: {}**\n**".format(best\_epoch\_id, best\_valid\_accuray))  **with** open('./resnet\_train\_log.txt', 'a+') **as** f:  f.write("**\n**Early Stop at Epoch {}, Accuracy: {}**\n**".format(best\_epoch\_id, best\_valid\_accuray))  **break**  **elif** best\_epoch\_id == epoch:  **print**("**\n**Save Model at Epoch {}, Accuracy: {}**\n**".format(best\_epoch\_id, best\_valid\_accuray))  **with** open('./resnet\_train\_log.txt', 'a+') **as** f:  f.write("**\n**Save Model at Epoch {}, Accuracy: {}**\n**".format(best\_epoch\_id, best\_valid\_accuray))  torch.save(model, save\_model\_path)  **else**:  **continue**  **print**("**\n**Best Accuracy: {} at Epoch {}**\n**".format(best\_valid\_accuray, best\_epoch\_id))  **with** open('./resnet\_train\_log.txt', 'a+') **as** f:  f.write("**\n**Best Accuracy: {} at Epoch {}**\n**".format(best\_valid\_accuray, best\_epoch\_id))  *# In[36]:*  **def** early\_stop(best\_epoch\_id, epoch, patience=10):  **if** epoch - best\_epoch\_id > patience:  **return** True  **return** False  *# 评估函数*  **def** eval(model, device, test\_loader, is\_train=True):  model.eval()  test\_loss = 0  correct = 0  **with** torch.no\_grad():  **for** data, target **in** test\_loader:  data, target = data.to(device), target.to(device)  output = model(data)  test\_loss += criterion(output, target).item()  *# 获得最大概率下标*  pred = output.max(1, keepdim=True)[1]  correct += pred.eq(target.view\_as(pred)).sum().item()    test\_loss /= len(test\_loader.dataset)  **if** is\_train:  **print**('**\n**Train Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **with** open('./resnet\_train\_log.txt', 'a+') **as** f:  f.write('**\n**Train Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **else**:  **print**('**\n**Valid Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **with** open('./resnet\_train\_log.txt', 'a+') **as** f:  f.write('**\n**Valid Average Loss: {:.4f}, Accuracy: {}/{} ({:.2f})%**\n**'.format(  test\_loss, correct, len(test\_loader.dataset), 100.\*correct/len(test\_loader.dataset)))  **return** 100.\*correct/len(test\_loader.dataset), test\_loss |

为了防止过拟合，通过early\_stop函数实现早停，保存最佳模型。

**模型使用：**

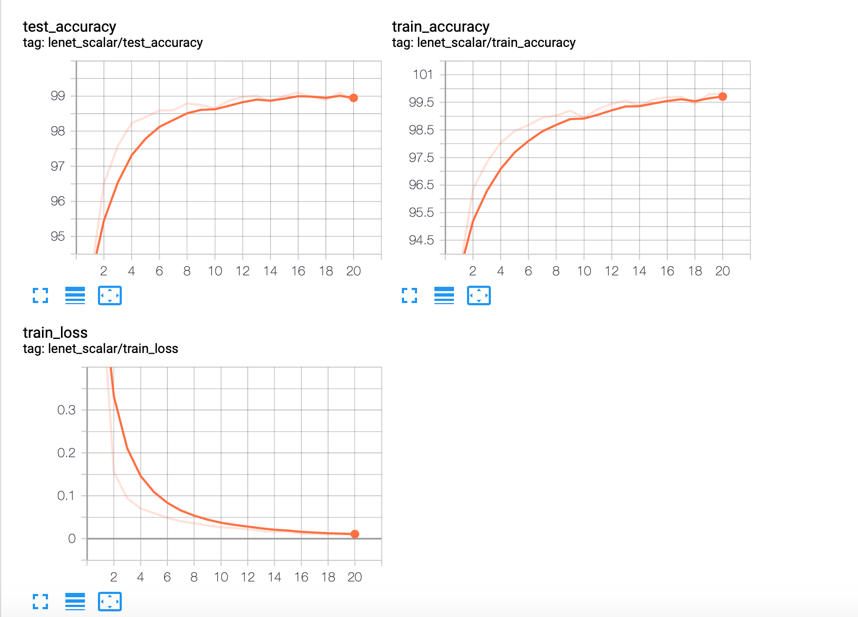
|  |
| --- |
| classes = ('plane','car','bird','cat','deer','dog','forg','horse','ship','truck')  **def** predict(model\_path, img\_path):  model = torch.load(model\_path)  transform=transforms.Compose([  transforms.Resize((32,32)),  transforms.ToTensor(),  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))  ])  device = torch.device('cuda' **if** torch.cuda.is\_available() **else** 'cpu')  model = model.to(device)  model.eval()  img = Image.open(img\_path)  img = transform(img).to(device)  img = img.unsqueeze(0)  output = model(img)  prob = F.softmax(output, dim=1)  pred = prob.max(dim=1)[1].item()  **return** classes[pred] |

predict函数通过传入模型文件名和待预测图片文件名，输出预测结果。

1. 实验结果与分析
2. 用MNIST训练集训练LeNet-5，并在测试集上验证效果

**训练与验证：**在hw3目录下依次运行rm -rf lenet\_scalar/与python LeNet5.py

**查看训练结果日志：**运行tensorboard –logdir=lenet\_scalar，根据指示打开浏览器对应网址，即见图3。最后训练结果为训练集上Accuracy (99.81)%，测试集上Accuracy (98.86)%。



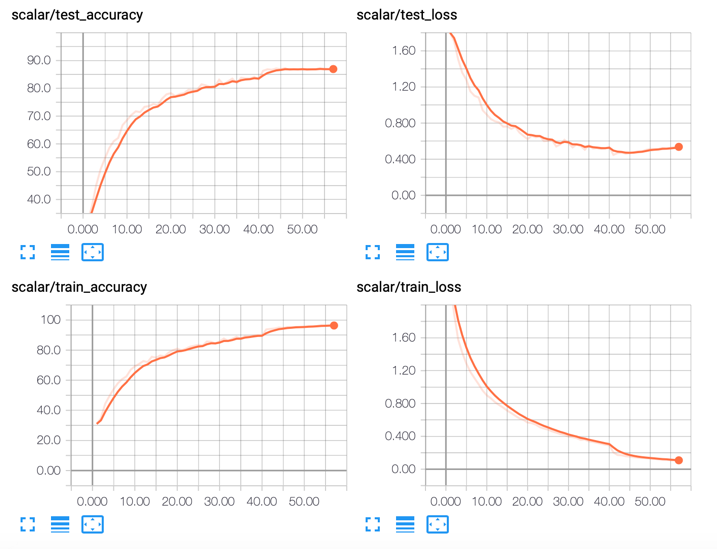
**图3 训练结果日志**

**使用模型预测：**运行python lenet\_predict.py -model\_path= LeNet5\_model.pth -img\_path=待检测图片路径

1. 用CIFAR10训练集上训练ResNet18，并在验证集上验证效果

**训练与验证：**在hw3目录下依次运行rm -rf resnet\_scalar/与python resnet.py

**查看训练结果日志：**在hw3目录下运行tensorboard –logdir=resnet\_scalar，根据指示打开浏览器对应网址，即见图4。最后训练结果为训练集上Accuracy (95.32)%，测试集上Accuracy (87.26)%，在epoch=46时早停。



**图4 ResNet18训练日志**

**使用模型预测：**运行python resnet\_predict.py -model\_path= ResNet18\_model.pth -img\_path=待检测图片路径

1. 结论与心得体会

在这次作业中熟悉了pytorch如何构建CNN模型，并熟悉了数据集使用与加载，同时对如何训练与验证模型效果有了一定的了解。学会使用tensorboard对训练日志进行可视化，便于分析模型。

1. 参考文献

[1] Kaiming He et al, Deep Residual Learning for Image Recognition, CVPR 2016.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.