浙江大学

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

一、 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

二、 作业的开发与运行环境

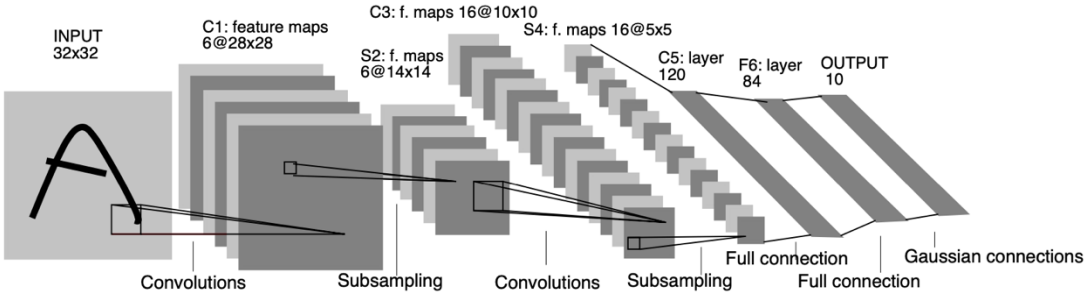
系统版本：macOS Catalina 10.15.2

python 版本：3.7.5

python 依赖环境：详见 hw3 文件夹下的 requirements.txt

三、 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。

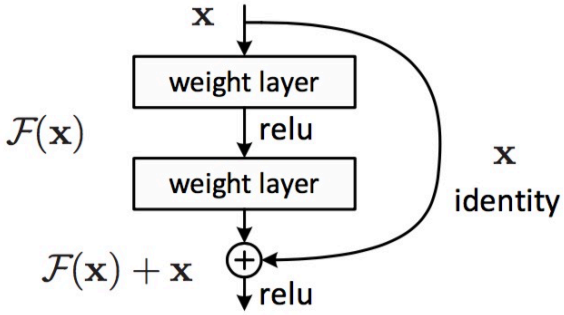


图 1 残差学习：a building block

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。

$$y = F(x,\{W_i\}) + x \qquad \text{公式(3.1)}$$

$$F = W_2\sigma(W_1x) \qquad \text{公式(3.2)}$$

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)。

$$y = F(x,\{W_i\}) + W_sx \qquad \text{公式(3.3)}$$

论文中的网络结构如图 2。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

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。

四、 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}%)]\tLoss: {:.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}%)]\tLoss:
{:.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('\nTrain 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('\nTrain Average Loss: {:.4f}, Accuracy: {}/{}
({:.2f})%\n'.format(
            test_loss, correct, len(test_loader.dataset),
100.*correct/len(test_loader.dataset)))
    else:
        print('\nValid 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('\nValid 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}%)]\tLoss: {:.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}%)]\tLoss:
{:.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("\nEarly Stop at Epoch {}, Accuracy: {}\n".format(best_epoch_id,
best_valid_accuray))
            with open('./resnet_train_log.txt', 'a+') as f:
                f.write("\nEarly Stop at Epoch {}, Accuracy:
{}\n".format(best_epoch_id, best_valid_accuray))
            break
        elif best_epoch_id == epoch:
            print("\nSave Model at Epoch {}, Accuracy: {}\n".format(best_epoch_id,
best_valid_accuray))
            with open('./resnet_train_log.txt', 'a+') as f:
                f.write("\nSave Model at Epoch {}, Accuracy:
{}\n".format(best_epoch_id, best_valid_accuray))
            torch.save(model, save_model_path)
        else:
            continue
    print("\nBest Accuracy: {} at Epoch {}\n".format(best_valid_accuray,
best_epoch_id))
    with open('./resnet_train_log.txt', 'a+') as f:
        f.write("\nBest 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('\nTrain 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('\nTrain Average Loss: {:.4f}, Accuracy: {}/{}
({:.2f})%\n'.format(
            test_loss, correct, len(test_loader.dataset),
100.*correct/len(test_loader.dataset)))
    else:
        print('\nValid 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('\nValid 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。

五、 实验结果与分析

1、 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)%。

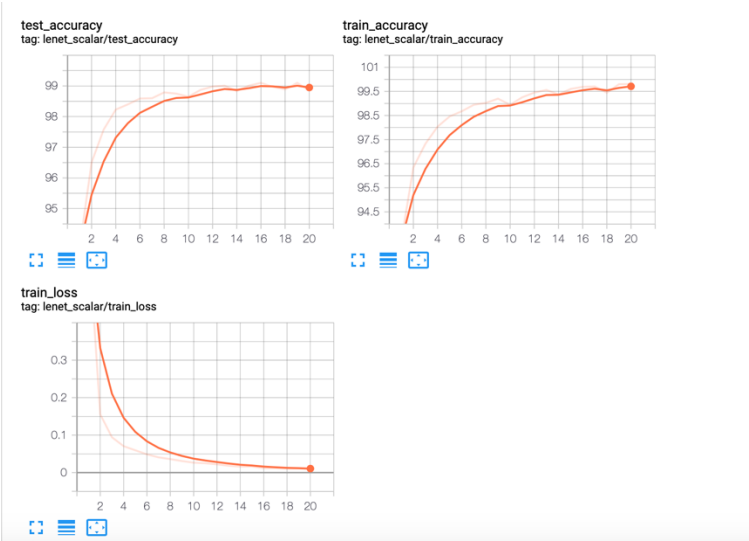


图 3 训练结果日志

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

2、 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。

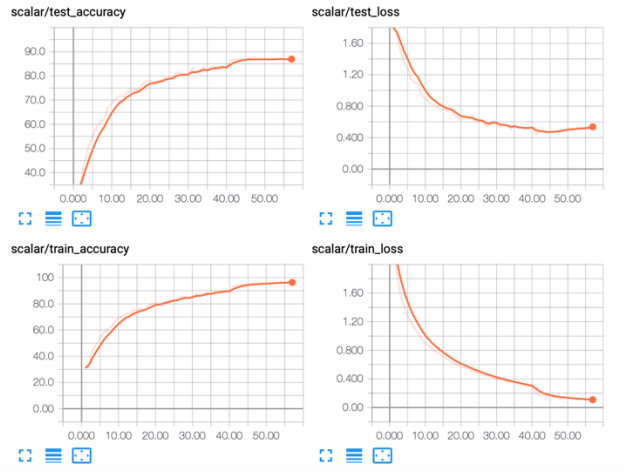


图 4 ResNet18 训练日志

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

六、 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。

七、 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.