

# 机器学习编程课

蔚全爱

July 2025

## 1 经典机器学习算法 day1

### 1.1 python 基础知识

变量赋值及打印

```
a = 3
b = "abc"
print(a)
a = 4
print(a)
```

**Remark:** 在 Python 中，等号 ‘=’ 表示赋值而非数学意义的相等；字符串必须使用英文单引号或双引号括起来，否则会被当作变量名。

**Output:**

```
3
4
```

列表操作

```
c = ["ml", a, b, ["list"]]
print(c)
print(c[0], c[3][0])
print(c + ["aaa"])
print(c * 2)
c.append("new_element")
c.remove(["list"])
print(c)
```

**Remark:** 列表可以包含不同类型的元素，包括其他列表。通过 ‘+’ 连接或 ‘\*’ 重复使用 ‘append’ 方法添加元素，使用 ‘remove’ 方法删除元素。

**Output:**

```
['ml', 4, 'abc', ['list']]
ml list
['ml', 4, 'abc', ['list'], 'aaa']
['ml', 4, 'abc', ['list'], 'ml', 4, 'abc', ['list']]
['ml', 4, 'abc', ['list'], 'new_element']
['ml', 4, 'abc', 'new_element']
```

元组不可变性

```
Tuple = (1, 2, 3)
print(Tuple)
try:
    Tuple[0] = 5
except TypeError:
    print("元组无法被更改")
```

**Remark:** 元组一旦创建后，其内容不可更改。

**Output:**

```
(1, 2, 3)
元组无法被更改
```

字典的键值操作

```
Dict = {"name1": "ml", "name2": 1, "name3": [1, 2]}
print(Dict["name2"])
Dict["name4"] = "new_ele"
del Dict["name1"]
print(Dict)
```

**Remark:** 字典是键值对的集合，可以通过键访问对应的值，可动态添加或删除键值对。

**Output:**

```
1
{'name2': 1, 'name3': [1, 2], 'name4': 'new_ele'}
```

for 循环

```
for i in range(1, 10, 1):
    List = []
    for j in range(1, 10, 1):
        if i >= j:
            List.append(f"{i}*{j}={i * j}")
    print(List)
```

**Output:**

```
['1*1=1']
['2*1=2', '2*2=4']
['3*1=3', '3*2=6', '3*3=9']
['4*1=4', '4*2=8', '4*3=12', '4*4=16']
['5*1=5', '5*2=10', '5*3=15', '5*4=20', '5*5=25']
['6*1=6', '6*2=12', '6*3=18', '6*4=24', '6*5=30', '6*6=36']
['7*1=7', '7*2=14', '7*3=21', '7*4=28', '7*5=35', '7*6=42', '7*7=49']
['8*1=8', '8*2=16', '8*3=24', '8*4=32', '8*5=40', '8*6=48', '8*7=56', '8*8=64']
['9*1=9', '9*2=18', '9*3=27', '9*4=36', '9*5=45', '9*6=54', '9*7=63', '9*8=72', '9*9=81']
```

简单的 for 循环

```
eexample = range(10)
example1 = [i + 1 for i in example]
example2 = [i * 2 for i in example]
print(example1)
```

**Output:**

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

字典内容遍历

```
# 遍历字典内容
for i in Dict:
    print(i)
for i in Dict.keys():
    print(i)
for i in Dict.values():
    print(i)
for i in Dict.items():
    print(i)
for i in Dict:
    print(i, Dict[i])
for key, value in Dict.items():
    print(key, value)
```

**Remark:** 遍历字典时默认取键, 可通过 `.keys()`, `.values()`, `.items()` 访问不同元素结构。

**Output:**

```
name2
name3
name4
name2
name3
name4
1
[1, 2]
new_ele
('name2', 1)
('name3', [1, 2])
('name4', 'new_ele')
name2 1
name3 [1, 2]
name4 new_ele
```

if-else 语句:

```
if isinstance(a, int):
    print("a 是 int 类型")
elif isinstance(a, str):
```

```

        print("a 是 str 类型")
    else:
        print("a 不是 int 类型或 str 类型")

if 5 >= a >= 0:
    print(True)
else:
    print(False)

```

**Remark:** 使用 `isinstance()` 可判断变量类型；Python 支持链式比较，如 `5 >= a >= 0`。

**Output:**

a 是 int 类型  
True

方法及调用

```

def method1(a):
    if isinstance(a, int):
        print("a 是 int 类型")
    elif isinstance(a, str):
        print("a 是 str 类型")
    else:
        print("a 不是 int 类型或 str 类型")

method1(a=1)

```

类实例化

```

class person:
    leg = 4
    eye = 2

a = person()
a.brain = "smart"

class MyClass:
    i = 12345
    def f(self, b):
        print(self.i)
        return 'hello world'

# 实例化类
x = MyClass()
x.i = 123
# 访问类的属性和方法
print("MyClass 类的属性 i 为: ", x.i)
print("MyClass 类的方法 f 输出为: ", x.f(b=2))

```

**Remark:** 类属性可直接访问；实例可动态添加属性（如 `brain`）；最后调用 `f(b=2)` 时只需要传入参数 `b`，不需要传入 `self`，因为 Python 会自动将实例作为第一个参数传递给方法。

```
class Circle:
    pi = 3.1415
    #特殊的函数，调用的时候会自动调用
    def __init__(self, r):
        self.r = r #注册的类的属性，只有实例化后才能访问

    def print_r(self):
        print(self.r)

    def cal(self, Type):
        pass

circle1 = Circle(r=1) #实例化为circle1
circle1.r = 2
circle1.pi = 3.14
print(Circle.pi, circle1.pi)
```

**Remark:** `__init__` 构造函数会在实例化时自动调用；类属性如 `pi` 可通过实例或类名访问，但可被实例覆盖；`circle1.pi = 3.14` 修改了实例的 `pi` 属性，但不会影响类属性。

**Output:**

3.1415 3.14

类的继承

```
class Ellipse(Circle):#继承自哪个类，在原类上增加不同的特征
    def __init__(self, r, r1):
        super(Ellipse, self).__init__(r)
        self.r1 = r1

    def cal(self, Type):
        if Type == "Circumference":
            return 2 * self.pi * self.r1 + 4 * (self.r - self.r1)
        elif Type == "Area":
            return 2 * self.pi * self.r1 * self.r
        else:
            raise NotImplementedError

ell = Ellipse(2, 3)
result = ell.cal("Area")
print(result)
ell.print_r()
```

**Remark:** `Ellipse` 类继承 `Circle`；`super()` 可调用父类构造函数和父类的 `self.f` 一致；子类可添加新属性（如 `r1`）

Output:

37.698  
2

## 1.2 基础算法

Kmeans 聚类算法的 sklearn 实现

```
import numpy as np
import sklearn.datasets as ds
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import homogeneity_score,
    completeness_score, v_measure_score, \
    adjusted_mutual_info_score, adjusted_rand_score,
    silhouette_score
import matplotlib as mpl
import matplotlib.pyplot as plt

# 生成数据
x, y = ds.make_blobs(n_samples=400, n_features=2, centers=4,
    random_state=2025)

# KMeans 聚类
model = KMeans(n_clusters=4)
model.fit(x)
y_pred = model.predict(x)

# 输出聚类评估指标
print('y_true = ', y)
print('y_pred = ', y_pred)
print('homogeneity_score = ', homogeneity_score(y, y_pred))
print('completeness_score = ', completeness_score(y, y_pred))
print('v_measure_score = ', v_measure_score(y, y_pred))
print('adjusted_mutual_info_score = ',
    adjusted_mutual_info_score(y, y_pred))
print('adjusted_rand_score = ', adjusted_rand_score(y, y_pred))
print('silhouette_score = ', silhouette_score(x, y_pred))

# 可视化结果
plt.figure(figsize=(8, 4))

plt.subplot(121)
plt.plot(x[:, 0], x[:, 1], 'r.', ms=3)

plt.subplot(122)
plt.scatter(x[:, 0], x[:, 1], c=y_pred, marker='.',
    cmap=mpl.colors.ListedColormap(list('rgbm')))

plt.tight_layout()
```

```
plt.show()
```

Output visualization:

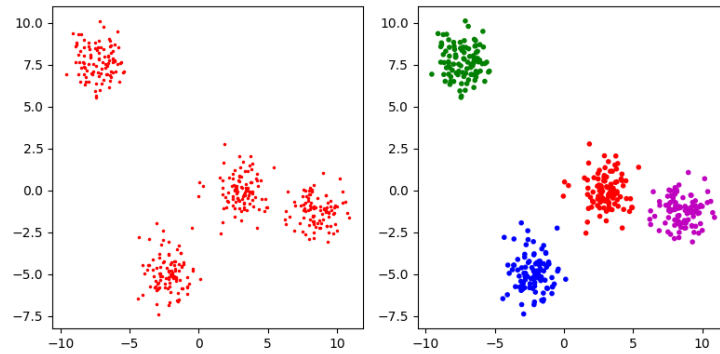


Figure 1: KMeans 聚类可视化结果：左图为原始数据，右图为聚类结果

Kmeans 聚类算法的手动实现：随机初始化聚类中心，分配每个样本到最近的聚类中心，更新聚类中心，直到收敛。

```
import numpy as np

class KMeans:
    def __init__(self, n_clusters=3, max_iter=100):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        # 随机初始化聚类中心
        self.centroids = X[np.random.choice(X.shape[0],
                                             self.n_clusters, replace=False)]

        for _ in range(self.max_iter):
            # 分配每个样本到最近的聚类中心
            distances = np.linalg.norm(X[:, np.newaxis] -
                                       self.centroids, axis=2)
            labels = np.argmin(distances, axis=1)
            # 更新聚类中心
            new_centroids = np.array([X[labels ==
                                       i].mean(axis=0) for i in
                                       range(self.n_clusters)])
            if np.all(new_centroids == self.centroids):
                break
            self.centroids = new_centroids
        return self

    def predict(self, X):
        distances = np.linalg.norm(X[:, np.newaxis] -
                                   self.centroids, axis=2)
```

```
return np.argmin(distances, axis=1)
```

完整的 KMeans 聚类算法实现

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler,
    PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt

if __name__ == "__main__":
    path = './iris.data' # 数据文件路径
    data = pd.read_csv(path, header=None)
    data[4] = pd.Categorical(data[4]).codes # 类别标签转为整数
    x, y = np.split(data.values, (4,), axis=1) #
        前4列是特征，最后1列是标签

    # 仅使用前两列特征
    x = x[:, :2]

    # 构建 Pipeline: 标准化 + 多项式特征 +
        逻辑回归，先后顺序执行
    lr = Pipeline([
        ('sc', StandardScaler()),
        ('poly', PolynomialFeatures(degree=10)),
        ('clf', LogisticRegression())
    ])

    lr.fit(x, y.ravel()) # 训练模型
    y_hat = lr.predict(x) # 预测类别
    y_hat_prob = lr.predict_proba(x) # 预测概率

    np.set_printoptions(suppress=True)
    print('y_hat = \n', y_hat)
    print('y_hat_prob = \n', y_hat_prob)
    print("准确率: %.2f%%" % (100 * np.mean(y_hat ==
        y.ravel()))))

    # 可视化决策边界
    N, M = 200, 200 # 横纵各采样多少个点
    x1_min, x1_max = x[:, 0].min(), x[:, 0].max() # 第0列的范围
    x2_min, x2_max = x[:, 1].min(), x[:, 1].max() # 第1列的范围

    t1 = np.linspace(x1_min, x1_max, N)
    t2 = np.linspace(x2_min, x2_max, M)
    x1, x2 = np.meshgrid(t1, t2) # 生成网格采样点
    x_test = np.stack((x1.flat, x2.flat), axis=1) # 测试点坐标集
```



```

# 设置中文字体、颜色
mpl.rcParams['font.sans-serif'] = ['simHei']
mpl.rcParams['axes.unicode_minus'] = False
cm_light = mpl.colors.ListedColormap(['#77E0A8', '#FF8080',
    '#A0A0FF']) # 背景色
cm_dark = mpl.colors.ListedColormap(['g', 'r', 'b'])
    # 点颜色

# 预测新样本的类别
y_hat = lr.predict(x_test)
y_hat = y_hat.reshape(x1.shape) # 转为网格形状

# 绘图
plt.figure(facecolor='w')
plt.pcolormesh(x1, x2, y_hat, cmap=cm_light) # 预测区域图
plt.scatter(x[:, 0], x[:, 1], c=y.ravel(), edgecolors='k',
    s=50, cmap=cm_dark) # 样本点图
plt.xlabel('花萼长度', fontsize=14)
plt.ylabel('花萼宽度', fontsize=14)
plt.title('逻辑回归对Iris数据集的分类结果', fontsize=16)
plt.tight_layout()
plt.show()

```

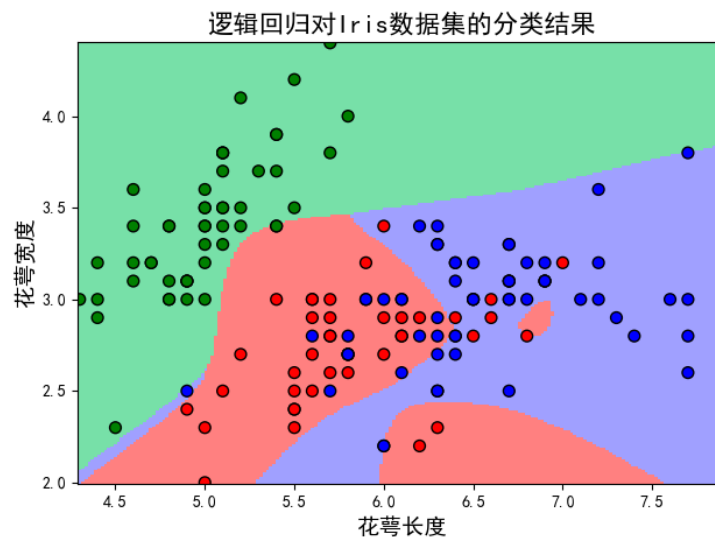


Figure 2: 逻辑回归在 Iris 数据集上的分类结果与决策边界

## 2 resnet 网络 day2

```

"""Train CIFAR10 with PyTorch."""
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
from torch.utils.data import Dataset
from PIL import Image
import torchvision
import torchvision.transforms as transforms

import os
import argparse

from models import *
from utils import progress_bar

class my_dataset(Dataset):
    def __init__(self, path, preprocess):
        self.preprocess = preprocess
        self.image_paths = []
        self.labels = []
        label_list = os.listdir(path)
        for label in label_list:
            image_folder = os.path.join(path, label)
            for file_names in os.listdir(image_folder):
                if file_names.endswith(("png", "jpg", "jpeg")):
                    self.image_paths.append(os.path.join(image_folder,
                                                            file_names))
                    self.labels.append(label_list.index(label))

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, item):
        image = Image.open(self.image_paths[item])
        image = self.preprocess(image)
        label = self.labels[item]
        return image, label

parser = argparse.ArgumentParser(description='PyTorch CIFAR10
    Training') ##允许自命令行的输入
## python main.py --lr 0.1 --resume
parser.add_argument('--lr', default=0.1, type=float,
    help='learning rate')
## 存储为bool类型

```

```

parser.add_argument('--resume', '-r', action='store_true',
                    help='resume from checkpoint')
args = parser.parse_args()
print(args)

device = 'cuda' if torch.cuda.is_available() else 'cpu'
best_acc = 0
start_epoch = 0
##数据预处理-加载数据集-加载加载器
# Data
print('==> Preparing data..')
##数据预处理
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),  ##转换成tensor加归一化
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023,
        0.1994, 0.2010)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023,
        0.1994, 0.2010)),
])

trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True,
    transform=transform_train)

##
    batchsize是每次加载多少个样本，shuffle是是否打乱顺序，一般是要打乱的，因为SGD打乱才能
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=128, shuffle=True, num_workers=0)
## windows只能设置为0，linux可以设置为4
testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True,
    transform=transform_test)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=100, shuffle=False, num_workers=0)

classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# Model
print('==> Building model..')
# net = VGG('VGG19')
net = ResNet18()
# net = PreActResNet18()

```

```

# net = GoogLeNet()
# net = DenseNet121()
# net = ResNeXt29_2x64d()
# net = MobileNet()
# net = MobileNetV2()
# net = DPN92()
# net = ShuffleNetG2()
# net = SENet18()
# net = ShuffleNetV2(1)
# net = EfficientNetB0()
# net = RegNetX_200MF()
# net = SimpleDLA()
net = net.to(device)
if device == 'cuda':
    # net = torch.nn.DataParallel(net)
    cudnn.benchmark = True

if args.resume:
    # Load checkpoint.
    print('==> Resuming from checkpoint..')
    assert os.path.isdir('checkpoint'), 'Error: no checkpoint
        directory found!'
    checkpoint = torch.load('./checkpoint/ckpt.pth')
    net.load_state_dict(checkpoint['net'])
    # weight = net.state_dict()
    # torch.save(weight, "/your/path")
    # weight = torch.load("/your/path")
    # net.load_state_dict(weight)
    best_acc = checkpoint['acc']
    start_epoch = checkpoint['epoch']

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=args.lr,
                       momentum=0.9, weight_decay=5e-4) #
                       loss=L+\lambda||w||^2
scheduler =
    torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
        T_max=200)

# Training
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs, targets) in enumerate(trainloader):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()

```

```

        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        # for param in net.parameters():
        #     print(param.data, param.grad)
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

    # tqdm
    progress_bar(batch_idx, len(trainloader), 'Loss: %.3f
    | Acc: %.3f%% (%d/%d)'
                % (train_loss / (batch_idx + 1), 100. *
                   correct / total, correct, total))

def test(epoch):
    global best_acc
    net.eval()
    # for param in net.parameters():
    #     param.requires_grad = False
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in
            enumerate(testloader):
                inputs, targets = inputs.to(device),
                    targets.to(device)
                outputs = net(inputs)
                loss = criterion(outputs, targets)

                test_loss += loss.item()
                _, predicted = outputs.max(1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()

        progress_bar(batch_idx, len(testloader), 'Loss:
        %.3f | Acc: %.3f%% (%d/%d)'
                    % (test_loss / (batch_idx + 1), 100.
                       * correct / total, correct, total))

    # Save checkpoint.
    acc = 100. * correct / total
    if acc > best_acc:
        print('Saving..')
        state = {

```

```

        'net': net.state_dict(),
        'acc': acc,
        'epoch': epoch,
    }
    if not os.path.isdir('checkpoint'):
        os.mkdir('checkpoint')
    torch.save(state, './checkpoint/ckpt.pth')
    best_acc = acc

for epoch in range(start_epoch, start_epoch + 200):
    train(epoch)
    test(epoch)
    scheduler.step()

```

```

# 自定义 Dataset，用于加载本地图片数据
class own_dataset(Dataset):
    #init,len,getitem
    def __init__(self, root, preprocess):
        """
        param root: 数据集根目录，包含多个类别文件夹
        param preprocess: 图像预处理函数
        """
        super(own_dataset, self).__init__()
        self.preprocess = preprocess
        self.image_paths = []
        self.labels = []

        label_list = os.listdir(root)#只会列出来
        for label in label_list:
            image_folder = os.path.join(root, label)##
                root路径加上label
            for file in os.listdir(image_folder):
                if file.endswith(("png", "jpg", "gif")):
                    self.image_paths.append(os.path.join(image_folder,
                                                            file))
                    self.labels.append(label_list.index(label))

    def __len__(self):
        return len(self.image_paths)

    ##
    getitem方法：通过索引获取图像和标签，此时的索引是一个随机数
    ## 所以必须要有len方法，从0到len-1随机采
    def __getitem__(self, item):
        image = Image.open(self.image_paths[item])## PIL.Image
        image = self.preprocess(image)##Tensor
        label = self.labels[item]
        return image, label

```

```

def print_len(self):
    print(len(self.image_paths))

# 构建数据集对象
trainset = own_dataset(root="./data",
    preprocess=transform_train)

```

**Remark:** 本类继承自 Dataset，用于加载本地文件夹结构的数据集。init ,len ,getitem 方法是必须的，目录结构应为：

```

./data/class1/image1.png
./data/class2/image2.jpg
...

```

ResNet 实现

```

"""ResNet in PyTorch.

For Pre-activation ResNet, see 'preact_resnet.py'.

Reference:
[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
    Deep Residual Learning for Image Recognition.
    arXiv:1512.03385

"""

import torch
import torch.nn as nn
import torch.nn.functional as F

# nn.Module是所有神经网络模块的基类
# 所有自定义的网络都需要继承nn.Module
# nn.model需要两个方法，初始化方法__init__和前向传播方法forward
# 调用时直接model()不需要model.forward()
# 一般模型都是块的形式，堆叠起来的
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride,
            padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        ## 同一个类的两次实例，参数不共享
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
            stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(

```

```

        nn.Conv2d(in_planes, self.expansion * planes,
                    kernel_size=1, stride=stride,
                    bias=False),
        nn.BatchNorm2d(self.expansion * planes)
    )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes,
                                kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                                stride=stride, padding=1,
                                bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion *
                                planes, kernel_size=1,
                                bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion * planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion * planes,
                            kernel_size=1, stride=stride,
                            bias=False),
                nn.BatchNorm2d(self.expansion * planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):

```



```

def __init__(self, block, num_blocks, num_classes=10):
    super(ResNet, self).__init__()
    self.in_planes = 64

    self.conv1 = nn.Conv2d(3, 64, kernel_size=3,
                           stride=1, padding=1, bias=False)
    self.bn1 = nn.BatchNorm2d(64)
    self.layer1 = self._make_layer(block, 64,
                                   num_blocks[0], stride=1)
    self.layer2 = self._make_layer(block, 128,
                                   num_blocks[1], stride=2)
    self.layer3 = self._make_layer(block, 256,
                                   num_blocks[2], stride=2)
    self.layer4 = self._make_layer(block, 512,
                                   num_blocks[3], stride=2)
    self.linear = nn.Linear(512 * block.expansion,
                           num_classes)

def _make_layer(self, block, planes, num_blocks, stride):
    ##需要保证输入的通道数和输出的通道数一致，或者通过shortcut来调整
    strides = [stride] + [1] * (num_blocks - 1)
    layers = []
    for stride in strides:
        layers.append(block(self.in_planes, planes,
                           stride))
        self.in_planes = planes * block.expansion
    # layer是一个列表，没有办法识别
    # 需要用nn.Sequential将其转换为一个模块
    return nn.Sequential(*layers)

def forward(self, x):
    out = F.relu(self.bn1(self.conv1(x)))
    out = self.layer1(out)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    ##需要自己尝试一下，不加效果更好？
    out = F.avg_pool2d(out, 4)
    out = out.view(out.size(0), -1)
    out = self.linear(out)
    return out

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

def ResNet34():
    return ResNet(BasicBlock, [3, 4, 6, 3])

```

```

def ResNet50():
    return ResNet(Bottleneck, [3, 4, 6, 3])

##Bottleneck比BasicBlock多了一个1x1的卷积层，减少了参数量
def ResNet101():
    return ResNet(Bottleneck, [3, 4, 23, 3])

def ResNet152():
    return ResNet(Bottleneck, [3, 8, 36, 3])

def test():
    net = ResNet18()
    y = net(torch.randn(1, 3, 32, 32))
    print(y.size())

# test()

```

**Remark:** 实现 ResNet 网络结构，主要注释解释如下：

- `nn.Module` 是所有神经网络模块的基类，自定义网络必须继承它；
- 网络必须实现两个方法：`__init__()` 进行模块初始化；`forward()` 实现前向传播；
- 实际调用模型时使用 `model(x)`，无需显式调用 `model.forward(x)`；
- 如果输入输出维度不一致（例如通道或步长不同），则使用 `shortcut` 分支自动调整维度；
- 一般模型都是块的形式，堆叠起来的