第四课：如何查看函数

1. 工具箱：

```
dir():打开，看见
help():说明书
```

总结:

dir()函数，能让我们知道工具箱以及工具箱中的分隔区有什么东西。

help()函数，能让我们知道每个工具是如何使用的，工具的使用方法。

2. 魔术方法：__name__

魔方方法在Python中，所有以“__”双下划线包起来的方法，都统称为“**Magic Method**”，中文称『魔术方法』,例如类的初始化方法 init 如果想使用魔方方法必须要继承object

第五讲：python文件，控制台和jupyter

1. python文件：代码是以块为一个整体运行的话:Python文件的块是所有行的代码。

> 优:通用，传播方便，适用于大型项目
> 缺:需要从头运行

2. Python控制台：以任意行为块进行运行。

> 优:显示每个变量属性
> 缺:不利于代码阅读及修改

3. Jupyter: 以任意行为块进行运行的。

> 优:利于代码阅读及修改
> 缺:环境需要配置

第六讲：加载数据

```
Dataset
```

提供一种方式去获取数据及其label

> 1. **如何获取每一个数据及其label**
> 2. **告诉我们总共有多少的数据**

2. 查看具体方法

```
jupyter 中 help(**)
jupyter 中 ***??
```

```
Dataloader
```

为后面的网络提供不同的数据形式

第七讲 Dataset类代码实战

```
def __init__(self, root_dir, lab_dir):
```

1. 提供全局变量；
2. 为后面的方法提供初始变量。

```
def __getitem__(self, idx):
def __len__(self):
```

这些都是一个类的基本函数。

```
img, label = apple_datasets[0]
```

调用函数，使用函数，查看相关数据。

shift+enter: 直接进入下一行。

第八讲 TensorBoard的使用（1）

1. 基本步骤

> 1.创建基本文件夹；
> 2. 对文件夹运用add_scalar函数写入函数；
> 3. 再关闭writer.close( );

2. 在 terminal 下输入打开显示界面：

```
tensorboard --logdir=事件文件所在文件夹名 --port=端口名
```

3. 当出现重复命名时，可以创建子文件夹，也就是创建新的文件夹

```
writer = SummaryWriter("新文件夹")
```

4. 显示曲线

```
writer.add_scalar("文档名称", y, x)
```

第九讲 TensorBoard的使用（2）

1. 用tensorboard显示图片

> img_path= 图片路径
> img = Image.open(img_path)
> img_np = np.array(img)

2. 在tensorboard中显示

```
add_image(self, tag, img_tensor, global_step, dataformats='CHW')
```

第九讲 Transforms的使用（1）

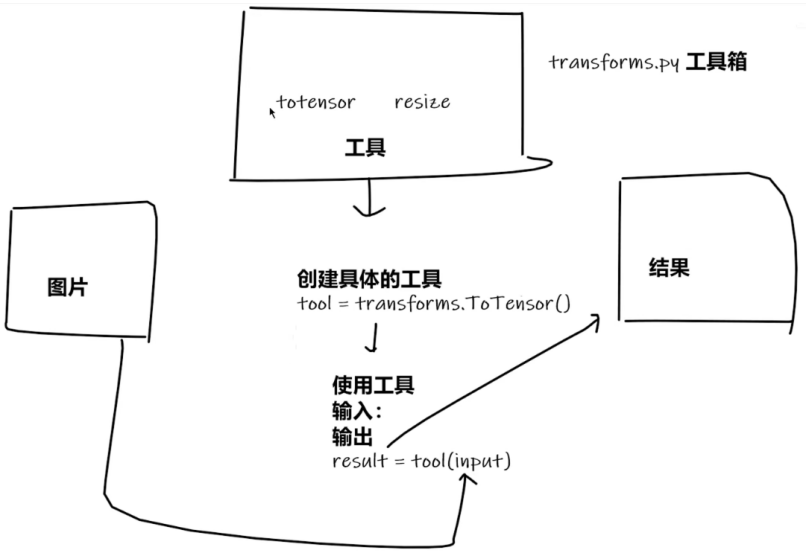
1.调出Structure:

> 1.在setting 中，输入keymap,再输入structure,最后修改快捷键；
> 2.或者在view下面的toolwindow下查询。

2. 输入一个函数，不知道参数该如何用，可以按

```
ctrl+P
```

3. transform的基本步骤：



第十一讲 Transforms的使用（2）

1. 打开图像的3种方法

> 1. PIL库中(PILimage格式)：Image.open()
> 2. cv2库中(narray格式)： cv2.imread()
> 3. transforms库中(Tensor格式)： transforms.ToTensor()

2. SummaryWriter输入，没有库的时候按住：

```
# 1. 选择没有安装的库
Alt+Enter
# 2. 选择需要的函数的属性：
ctrl+P
# 3. 换到下一行
shift + enter
# 4. 查看没有输入完的函数
Ctrl + 空格
# 补全快捷键
 ctrl+shift+space
```

第十二讲 常见的Transforms（1和2）

1. __call__函数：是把类当做函数使用的时候的定义

2. PyCharm小技巧设置忽略大小写，进行提示匹配：

```
setting 搜索 case，选择第一个：code Completion
Match case 取消打勾
```

3. Compose()用法

> 1. Compose()中的参数需要是一个列表Python中，列表的表示形式为[数据1，数据2，...]
> 2. 在Compose中，数据需要是transforms类型
> 所以得到，Compose([transforms参数1, transforms参数2,..])
> 3. transforms参数1的输出数据类型要跟transforms参数2输入数据类型一致。

4. Resize的用法（方法1：按照像素进行缩放）

```
tran_resize = transforms.Resize((512, 512))
img_resize = tran_resize(img)
img_resize = trans_totensor(img_resize)
writer.add_image('reszie', img_resize, 0)
```

5. Resize的用法（方法2：按照比例进行缩放）

```
tran_resize_2 = transforms.Resize(512)
tran_compose = transforms.Compose([tran_resize_2, trans_totensor])
```

```
img_resize_2 = tran_compose(img)
writer.add_image('resize', img_resize_2, 1)
```

6. 总结

```
关注输入和输出类型
多看官方文档
关注方法需要什么参数
```

```
不知道返回值的时候
print
print(type())
debug
```

第十三讲 torchvision的数据集使用

1. 一段打上注释

```
ctrl+/
```

2. dataset的使用：

```
torchvision.datasets.数据集
```

第十五讲 DataLoader的使用

1. 在SummaryWriter中显示DataLoader数据：

```
test_data = torchvision.datasets.CIFAR10('./datasets', train=False, transform=torchvision.transforms.ToTensor())
test_dataLoader = DataLoader(test_data, batch_size=64, num_workers=0, drop_last=False, shuffle=True)

# 批量显示数据
writer = SummaryWriter("logs")
for epoch in range(3):
 step = 0
 for data in test_dataLoader:
 imgs, labels = data
  writer.add_images('Epoch:{}'.format(epoch), imgs, step)
  step = step+1

writer.close()
```

第十六讲 神经网络的基本骨架-nn.Module的使用

1. .py文件名的抬头设置：

> 1. setting -> fille and code templates -> python scripy
> 2. 输入：

```
# -*- coding: utf-8 -*-
# Date: ${DATE}
# Time: ${TIME}
# Author: 黄麒睿
```

2. 利用已有的网络框架：

> 1. 建立好类以后；
> 2. 在code -> Override Methods...(Ctrl+O)
> 3. 快捷键为：(Alt+insert) -> (Ctrl+O)

3. 进行debug的时候，需要在自己代码内进行操作(快捷键是F7)：

```
Step Into My Code()
```

第十七讲 卷积操作

1. 升维，用reshape() 可以，用unsqueeze() 也可以
2. con2d中stride的值可以是2维也可以是1维。
3.

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

第十七/十八讲 最大池化/非线性激活/线性层

1. Con2D和MaxPool2D的stride(1维和2维),dialation和ceil_mode。
2. 正则化是防止过拟合提高泛化能力

第十九讲 搭建小实战和Sequential介绍

1. 通过自建数据检查网络搭建是否正确的检测方法
2. tensorboard的可视化

```
writer.add_graph()
```

第二十讲 损失函数和反向传播

1. 损失函数的意义

> 1.计算实际输出和目标之间的差距
> 2.为我们更新输出提供一定的依据(反向传播)

2. 查看相关的损失函数需要：各类损失函数的输入要求和意义

第二十一讲 优化器/现有网络模型的使用及修改

1. 查看官方文档，优化器使用的基本方法

```
optimizer.zero_grad()
output = HQR_optim(imgs)
Result_Loss = loss(output, label)
Result_Loss.backward()
optimizer.step()
```

2. 网络模型下载：分为：下载已训练好的参数和下载没有训练好的参数。
3. 在已有模型中添加一层

```
VGG16_false.classifier.add_module('HQR', Linear(1000, 10))
```

4. 对现有的模型进行改进

```
VGG16_false.classifier[6] = Linear(4096, 10)
```

第二十二讲 模型的保存与读取

1. 保存和加载方法1

```
# 保存方法1(结构+参数)
torch.save(vgg16_False, '../vgg16_method1.pth')

# 打开方法1
method1 = torch.load('../vgg16_method1.pth')
print(method1)
```

2. 保存和加载方式2

```
# 保存方法2(参数)
torch.save(vgg16_False.state_dict(), '../vgg16_method2.pth')

# 打开方法2
method2 = torchvision.models.vgg16(pretrained=False)
method2.load_state_dict(torch.load('../vgg16_method2.pth'))
print(method2)
```

3. 陷阱

```
# 自建模型，并用方法1保存
# 模型
HQR_optim = HQR_save()
torch.save(HQR_optim, '../hqr_optim.pth')

# 打开方式1的陷阱(因为没有网络结构)
method3 = torch.load('../hqr_optim.pth')
print(method3)
# 此时会报错

 # 解决方法(再抬头把原网络所对应的文件夹放入进去)
 from model_save import *
```

第二十三讲 完整的模型训练套路（1）

1. 基本步骤：

- [ ] 导入和加载数据
- [ ] 加载模型、优化器和tensorboard
- [ ] 创建训练中需要的参数

- [ ] 训练和测试(with torch.no_grad())
- [ ] 保存训练数据
- [ ] 放置再GPU上跑，并记录时间

2. scratches and Consoles 进行测试

```
import torch
a = torch.tensor(5)
print(a)
print(a.item())
```

3. 测试的时候，梯度需要为0：

```
with torch.no_grad():
```

第二十四讲 完整的模型训练套路（2）

1. argmax的使用(对应于分类问题的准确率)

```
output = torch.tensor([[0.1, 0.2],
 [0.05, 0.4]])
# 0表示纵向；1表示横向
print(output.argmax(0))
# argmax与标签进行比较
(output.argmax(0)==target).sum()
```

2. 模型名称.train() 与 模型名称.eval()

- [ ] 模型名称.train()表示模型开始训练，一般对Dropout和BatchNorm等有作用；
- [ ] 模型名称.eval()表示模型开始验证，一般对Dropout和BatchNorm等有作用；

第二十五讲 利用GPU训练

1. GPU只对以下可以利用 .cuda

> 1. 网络模型；
> 2. 数据（输入，输出）
> 3. 损失函数

2. 使用GPU方法：

```
# 方法1
```

```
if torch.cuda.is_available():
 test_imgs = test_imgs.cuda()

# 方法2
device = torch.device("cuda:0")
imgs = imgs.to(device)

# 方法2中，device的优化写法
 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

3. 查看GPU的使用情况

```
nvidia-smi
```

4. 在网络上运行：Google colab

第二十六讲 完整的模型验证(测试)套路

利用已经训练好的模型，给他提供输入

1. 基本步骤：

- [ ] 读出图片，并转换到规定格式

```
# 由四通道转换为三通道
image = image.convert('RGB')
transfomer = torchvision.transforms.Compose([
 torchvision.transforms.Resize((32, 32)),
 torchvision.transforms.ToTensor()
])
```

- [ ] 加载已训练模型，注意模型是在GPU还是再CPU上训练

```
model = torch.load('../save_model/CIFAR10.pth',
map_location=torch.device('cpu'))
```

- [ ] 输出结果，如果是分类问题，输出最大的

```
print(target.argmax(1))
```

第二十七讲 看看开源项目