```python
import nbformat
from nbconvert import PythonExporter

# 将 notebook 文件转换为 Python 脚本
def notebook_to_script(notebook_path, output_script):
    with open(notebook_path, 'r', encoding='utf-8') as notebook_file:
        notebook_content = nbformat.read(notebook_file, as_version=4)

    python_exporter = PythonExporter()
    script_content, _ = python_exporter.from_notebook_node(notebook_content)

    with open(output_script, 'w', encoding='utf-8') as script_file:
        script_file.write(script_content)

notebook_to_script('/content/networks.ipynb', '/content/networks.py')
notebook_to_script('/content/utils.ipynb', '/content/utils.py')

# 导入生成的 Python 脚本
import networks
import utils
```

```python
import zipfile

# 定义 ZIP 文件的路径和解压目录
mnist_zip_path = '/content/mnist_dataset/images.zip'
mnist_extract_path = '/content/mnist_dataset/images'

# 解压 ZIP 文件
with zipfile.ZipFile(mnist_zip_path, 'r') as zip_ref:
    zip_ref.extractall(mnist_extract_path)

# 查看解压后的文件夹内容
import os
print(os.listdir(mnist_extract_path))
```

`['__MACOSX', 'images']`

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import pandas as pd
from PIL import Image
import os

# 导入转换后的 `networks` 模块
import networks
```

```python
# 列出 networks 模块中的所有属性
print(dir(networks))
```

`['AlexNet', 'AlexNetBN', 'BasicBlock', 'BasicBlock_AF', 'Bottleneck', 'Bottleneck_AF', 'ConvNet', 'F', 'LeNet', 'MLP', 'ResNet', 'ResNet101', 'ResNet152', 'ResNet18', 'ResNet18BN', 'ResNet18BN_AF', 'ResNet18_AF', 'ResNet34', 'ResNet50', 'ResNet_AF', 'SeinN', 'VGG', 'VGG11', 'VGG11BN', 'VGG13', 'VGG16', 'VGG19', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'cfg_vgg', 'nn', 'torch']`

```python
# 设置设备为 GPU 或 CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
# 设置 MNIST 数据集的预处理
mnist_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# 加载 MNIST 数据集
mnist_train = datasets.MNIST(root='./data', train=True, transform=mnist_transform, download=True)
mnist_test = datasets.MNIST(root='./data', train=False, transform=mnist_transform, download=True)

# 设置数据加载器
train_loader_mnist = DataLoader(mnist_train, batch_size=256, shuffle=True)
test_loader_mnist = DataLoader(mnist_test, batch_size=256, shuffle=False)
```

```python
# 自定义 MNIST 数据集类
class MNISTDataset(torch.utils.data.Dataset):
    def __init__(self, annotations_file, img_dir, transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = Image.open(img_path).convert("RGB")
        label_text = self.img_labels.iloc[idx, 1]

        # 将标签转换为整数（例如 '5F' -> 0, '5SX' -> 1）
        if label_text == '5F':
            label = 0
        elif label_text == '5SX':
            label = 1
        else:
            raise ValueError(f"Unexpected label value: {label_text}")

        if self.transform:
            image = self.transform(image)

        return image, label
```

```python
# MNIST 数据集路径
annotations_file = '/content/mnist_dataset/annotations.csv'
img_dir = '/content/mnist_dataset/images/images'

# 设置 MNIST 数据集的预处理
mnist_transform = transforms.Compose([
    transforms.Resize((28,28)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))     # 适用于 3 通道的 RGB 图像
])

# 加载 MNIST 数据集
mnist_dataset = MNISTDataset(annotations_file=annotations_file, img_dir=img_dir, transform=mnist_transform)
train_loader_mnist = DataLoader(mnist_dataset, batch_size=128, shuffle=True)
```

```python
# 训练模型函数

# 训练模型函数（添加异常捕获和进一步调试信息）
def train_model(model, data_loader, criterion, optimizer, epochs=20):
    model.train()     # 切换到训练模式
    for epoch in range(epochs):
        running_loss = 0.0
        for batch_idx, (inputs, labels) in enumerate(data_loader):
            try:
                # 打印 inputs 的类型和内容
                #print(f"Batch {batch_idx}: inputs 的类型是: {type(inputs)}")
                #print(f"Batch {batch_idx}: labels 的类型是: {type(labels)}")

                # 检查是否为 Tensor，如果不是则做类型转换
                if not isinstance(inputs, torch.Tensor):
                    raise ValueError(f"Batch {batch_idx}: inputs 不是 torch.Tensor")

                # 将 inputs 和 labels 移动到 GPU（或 CPU）
                inputs = inputs.to(device)
                labels = labels.to(device)

                # 前向传播
                outputs = model(inputs)
                loss = criterion(outputs, labels)

                # 反向传播和优化
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                running_loss += loss.item()

            except Exception as e:
                # 捕获任何异常并打印详细信息
                #print(f"在处理 Batch {batch_idx} 时发生错误: {e}")
                continue

        print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(data_loader)}")
```

```python
# 评估模型函数
def evaluate_model(model, test_loader):
    model.eval()     # 切换到评估模式
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_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()
    accuracy = 100 * correct / total
    print(f'Test Accuracy: {accuracy:.2f}%')
```

```python
# 初始化 LeNet 模型，传入形状参数
model_mnist = networks.LeNet(channel=3, num_classes=10).to(device)

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer_mnist = optim.SGD(model_mnist.parameters(), lr=0.01, momentum=0.9)

# 训练模型
train_model(model_mnist, train_loader_mnist, criterion, optimizer_mnist, epochs=20)

# 评估模型
evaluate_model(model_mnist, test_loader_mnist)
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

# 定义 MNIST 数据集的预处理步骤
mnist_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# 加载 MNIST 数据集
mnist_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=mnist_transform)
mnist_test_dataset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=mnist_transform)

# 数据加载器
train_loader_mnist = DataLoader(mnist_dataset, batch_size=256, shuffle=True)
test_loader_mnist = DataLoader(mnist_test_dataset, batch_size=256, shuffle=False)

# 使用 LeNet 替换 ResNet18
model_mnist = networks.LeNet(channel=1, num_classes=10).to(device)
model_mhist = networks.LeNet(channel=1, num_classes=2).to(device)

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer_mnist = optim.SGD(model_mnist.parameters(), lr=0.01, momentum=0.9)
optimizer_mhist = optim.SGD(model_mhist.parameters(), lr=0.01, momentum=0.9)

# 训练 MNIST 模型
train_model(model_mnist, train_loader_mnist, criterion, optimizer_mnist, epochs=20)
evaluate_model(model_mnist, test_loader_mnist)

# 训练 MHIST 模型
train_model(model_mhist, train_loader_mhist, criterion, optimizer_mhist, epochs=20)
evaluate_model(model_mhist, train_loader_mhist)
```

后续步骤： 说明错误