PyTorch下的可视化工具（网络结构/训练过程可视化）

CV开发者都爱看的 极市平台 2023-04-02 22:00:57 发表于广东 手机阅读

作者 | 锦恢@知乎（已授权）

来源 | https://zhuanlan.zhihu.com/p/220403674

编辑 | 极市平台

极市导读

本文大致想说一下pytorch下的**网络结构可视化**和**训练过程可视化**。>>加入极市CV技术交流群，走在计算机视觉的最前沿

一、网络结构的可视化

我们训练神经网络时，除了随着step或者epoch观察损失函数的走势，从而建立对目前网络优化的基本认知外，也可以通过一些额外的可视化库来可视化我们的神经网络结构图。这将更加地高效地向读者展现目前的网络结构。

为了可视化神经网络，我们先建立一个简单的卷积层神经网络：

```
import torch
import torch.nn as nn

class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
```

```
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 16, 3, 1, 1),
            nn.ReLU(),
            nn.AvgPool2d(2, 2)
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.fc = nn.Sequential(
            nn.Linear(32 * 7 * 7, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU()
        )

        self.out = nn.Linear(64, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        output = self.out(x)
        return output
```

输出网络结构：

```
MyConvNet = ConvNet()
print(MyConvNet)
```

输出结果：

```
ConvNet(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (conv2): Sequential(
```

```
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=1568, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU()
  )
  (out): Linear(in_features=64, out_features=10, bias=True)
)
```

有了基本的神经网络后，我们分别通过 `HiddenLayer` 和 `PyTorchViz` 库来可视化上述的卷积层神经网络。

> 需要说明的是，这两个库都是基于Graphviz开发的，因此倘若你的电脑上没有安装并且没有添加环境变量，请自行安装Graphviz工具，安装教程

1.1 通过HiddenLayer可视化网络

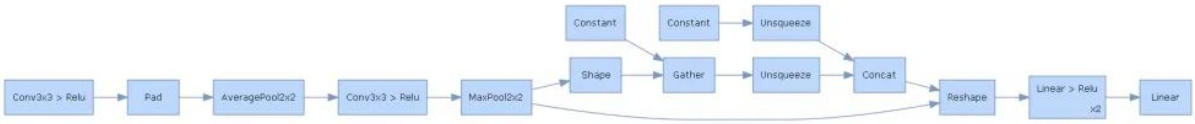
首先当然是安装库啦，打开cmd，输入：

```
pip install hiddenlayer
```

绘制的基本程序如下：

```
import hiddenlayer as h
vis_graph = h.build_graph(MyConvNet, torch.zeros([1 ,1, 28, 28]))   # 获取绘制图像的对象
vis_graph.theme = h.graph.THEMES["blue"].copy()     # 指定主题颜色
vis_graph.save("./demo1.png")   # 保存图像的路径
```

效果如下：



1.2 通过PyTorchViz可视化网络

先安装库：

```
pip install torchviz
```

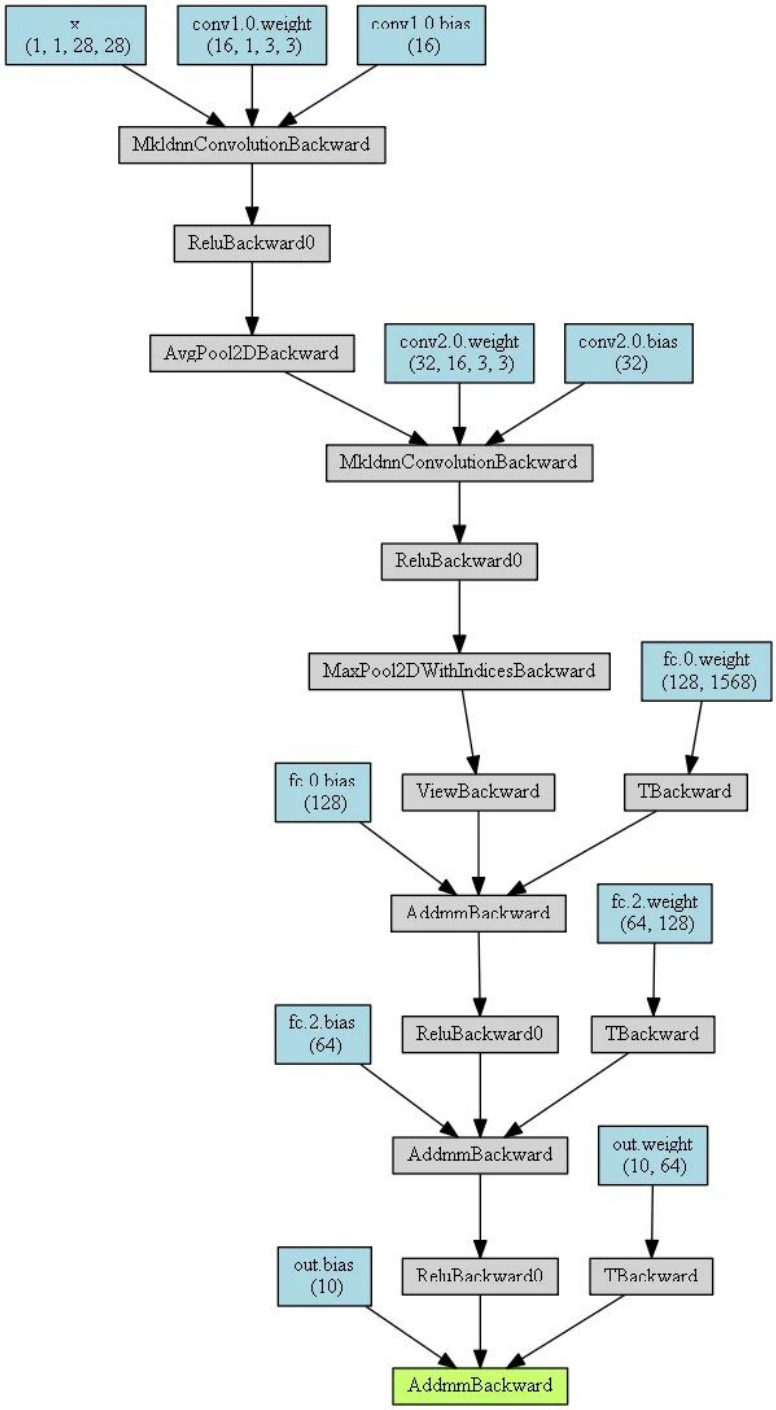
这里我们只使用可视化函数 `make_dot()` 来获取绘图对象，基本使用和 `HiddenLayer` 差不多，不同的地方在于 `PyTorch` 绘图之前可以指定一个网络的输入值和预测值。

```
from torchviz import make_dot
x = torch.randn(1, 1, 28, 28).requires_grad_(True)  # 定义一个网络的输入值
y = MyConvNet(x)    # 获取网络的预测值

MyConvNetVis = make_dot(y, params=dict(list(MyConvNet.named_parameters()) + [('x', x)]))
MyConvNetVis.format = "png"
# 指定文件生成的文件夹
MyConvNetVis.directory = "data"
# 生成文件
MyConvNetVis.view()
```

打开与上述代码相同根目录下的data文件夹，里面会有一个 `.gv` 文件和一个 `.png` 文件，其中的 `.gv` 文件是Graphviz工具生成图片的脚本代码， `.png` 是 `.gv` 文件编译生成的图片，直接打开 `.png` 文件就行。

> 默认情况下，上述程序运行后会自动打开.png文件

生成图片：



二、训练过程可视化

观察我们的网络的每一步的损失函数或准确率的变化可以有效地帮助我们判断当前训练过程的优劣。如果能将这些过程可视化，那么我们判断的准确性和舒适性都会有所增加。

此处主要讲通过可视化神器 `tensorboardX` 和刚刚用到的 `HiddenLayer` 来实现训练过程的可视化。

为了训练网络，我们先导入训练网络需要的数据，此处就导入MNIST数据集，并做训练前的一些基本的数据处理。

```
import torchvision
import torch.utils.data as Data
# 准备训练用的MNIST数据集
train_data = torchvision.datasets.MNIST(
    root = "./data/MNIST",  # 提取数据的路径
    train=True, # 使用MNIST内的训练数据
    transform=torchvision.transforms.ToTensor(),    # 转换成torch.tensor
    download=False   # 如果是第一次运行的话，置为True，表示下载数据集到root目录
)

# 定义loader
train_loader = Data.DataLoader(
    dataset=train_data,
    batch_size=128,
    shuffle=True,
    num_workers=0
)

test_data = torchvision.datasets.MNIST(
    root="./data/MNIST",
    train=False,    # 使用测试数据
    download=False
)

# 将测试数据压缩到0-1
test_data_x = test_data.data.type(torch.FloatTensor) / 255.0
test_data_x = torch.unsqueeze(test_data_x, dim=1)
test_data_y = test_data.targets

# 打印一下测试数据和训练数据的shape
print("test_data_x.shape:", test_data_x.shape)
print("test_data_y.shape:", test_data_y.shape)

for x, y in train_loader:
    print(x.shape)
    print(y.shape)
    break
```

结果：

```
test_data_x.shape: torch.Size([10000, 1, 28, 28])
test_data_y.shape: torch.Size([10000])
```

```
torch.Size([128, 1, 28, 28])
torch.Size([128])
```

2.1 通过tensorboardX可视化训练过程

tensorboard 是谷歌开发的深度学习框架tensorflow的一套深度学习可视化神器，在pytorch团队的努力下，他们开发出了tensorboardX来让pytorch的玩家也能享受tensorboard的福利。

先安装相关的库：

```
pip install tensorboardX
pip install tensorboard
```

并将tensorboard.exe所在的文件夹路径加入环境变量path中（比如我的tensorboard.exe的路径为 D:\Python376\Scripts\tensorboard.exe ，那么就在path中加入 D:\Python376\Scripts ）

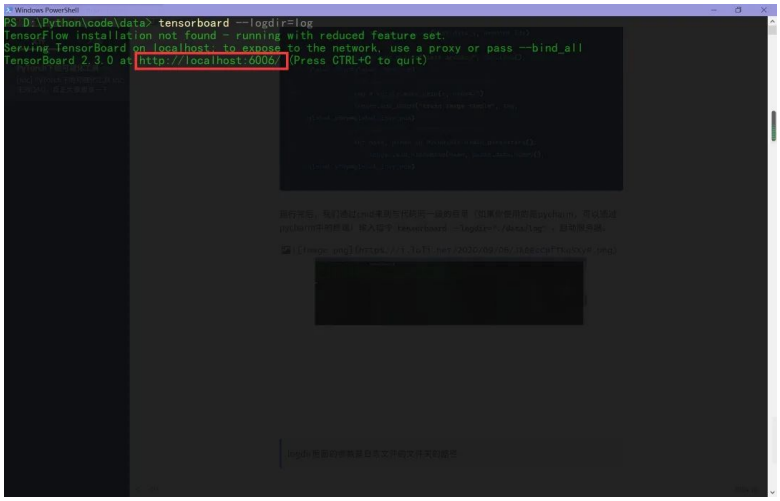
下面是 tensorboardX 的使用过程。基本使用为，先通过 tensorboardX 下的 SummaryWriter 类获取一个日志编写器对象。然后通过这个对象的一组方法往日志中添加事件，即生成相应的图片，最后启动前端服务器，在localhost中就可以看到最终的结果了。

训练网络，并可视化网络训练过程的代码如下：

```
from tensorboardX import SummaryWriter
logger = SummaryWriter(log_dir="data/log")

# 获取优化器和损失函数
optimizer = torch.optim.Adam(MyConvNet.parameters(), lr=3e-4)
loss_func = nn.CrossEntropyLoss()
log_step_interval = 100       # 记录的步数间隔

for epoch in range(5):
    print("epoch:", epoch)
    # 每一轮都遍历一遍数据加载器
    for step, (x, y) in enumerate(train_loader):
        # 前向计算->计算损失函数->(从损失函数)反向传播->更新网络
        predict = MyConvNet(x)
        loss = loss_func(predict, y)
```

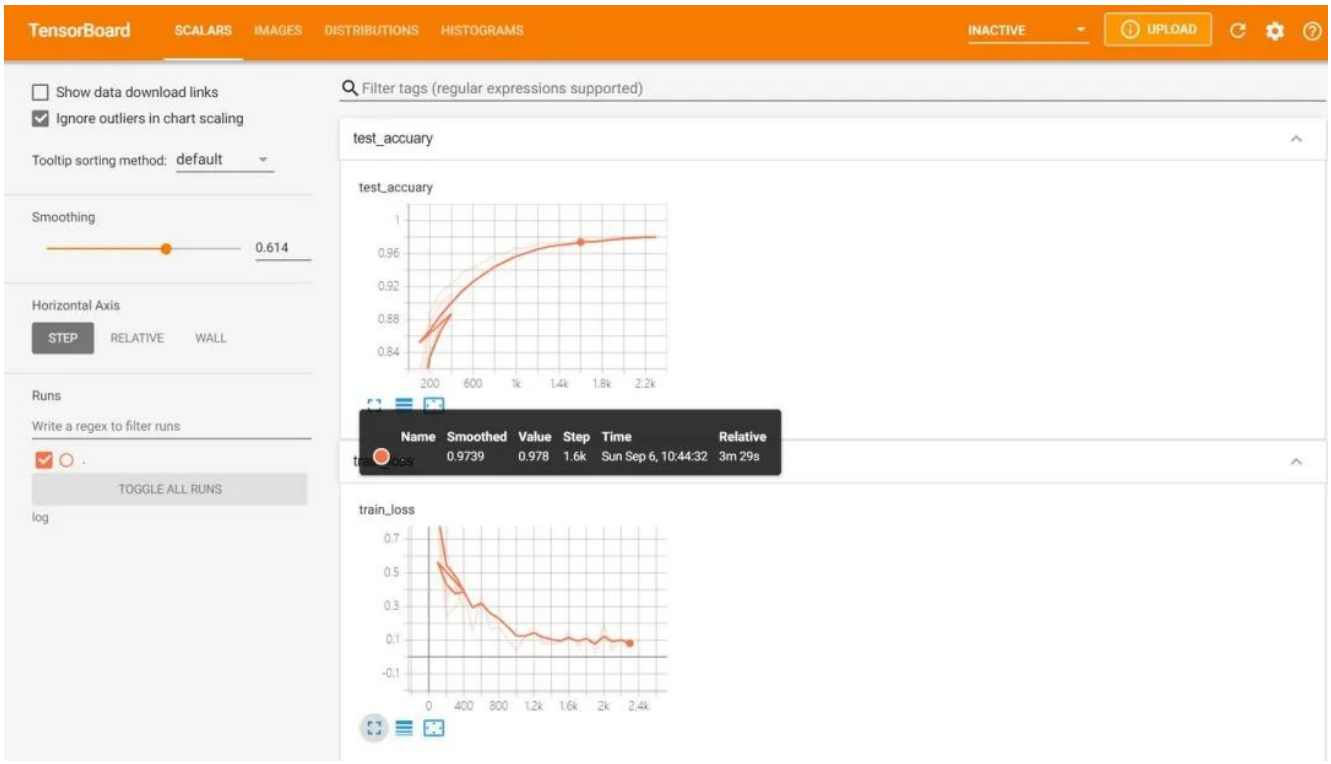
```
        optimizer.zero_grad()   # 清空梯度（可以不写）
        loss.backward()     # 反向传播计算梯度
        optimizer.step()    # 更新网络
        global_iter_num = epoch * len(train_loader) + step + 1  # 计算当前是从训练开始时的第
        if global_iter_num % log_step_interval == 0:
            # 控制台输出一下
            print("global_step:{}, loss:{:.2}".format(global_iter_num, loss.item()))
            # 添加的第一条日志：损失函数-全局迭代次数
            logger.add_scalar("train loss", loss.item() ,global_step=global_iter_num)
            # 在测试集上预测并计算正确率
            test_predict = MyConvNet(test_data_x)
            _, predict_idx = torch.max(test_predict, 1)     # 计算softmax后的最大值的索引，即
            acc = accuracy_score(test_data_y, predict_idx)
            # 添加第二条日志：正确率-全局迭代次数
            logger.add_scalar("test accuary", acc.item(), global_step=global_iter_num)
            # 添加第三条日志：这个batch下的128张图像
            img = vutils.make_grid(x, nrow=12)
            logger.add_image("train image sample", img, global_step=global_iter_num)
            # 添加第三条日志：网络中的参数分布直方图
            for name, param in MyConvNet.named_parameters():
                logger.add_histogram(name, param.data.numpy(), global_step=global_iter_nu
```

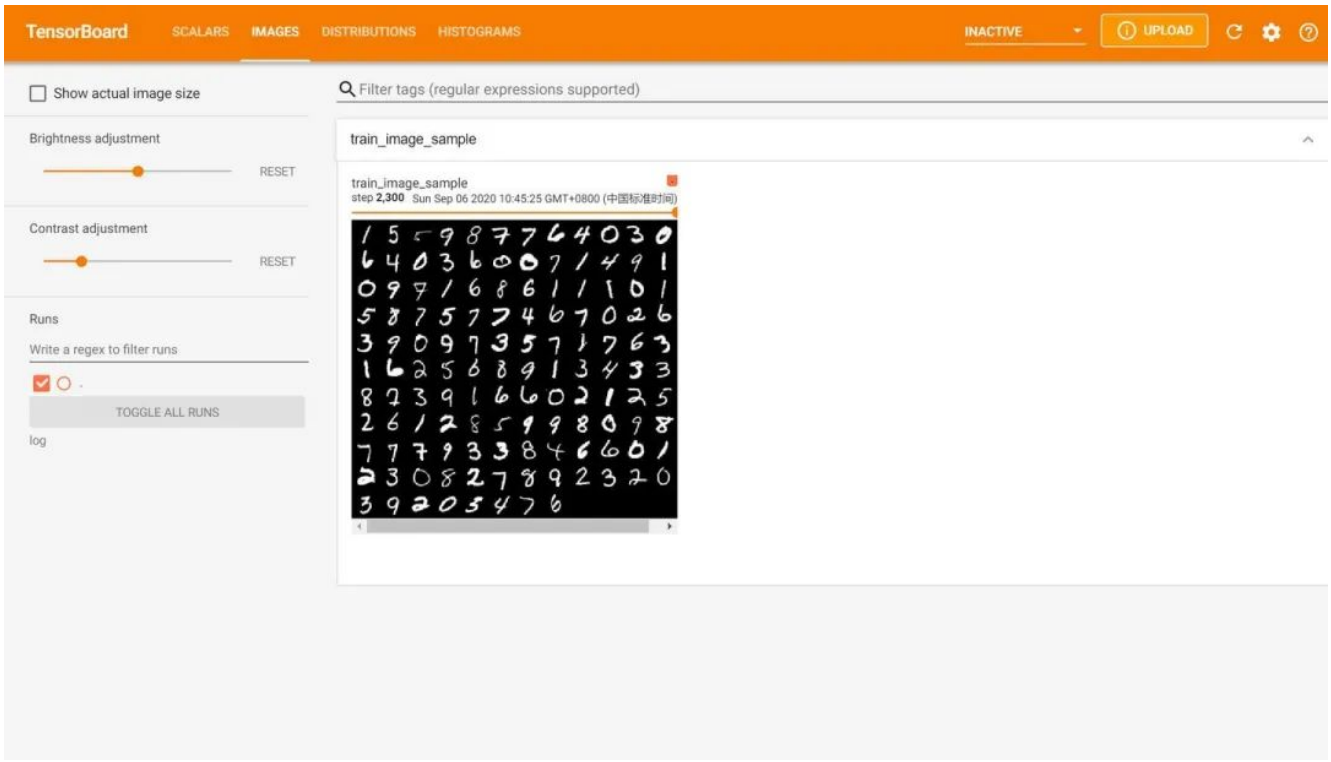
运行完后，我们通过cmd来到与代码同一级的目录（如果你使用的是pycharm，可以通过pycharm中的终端）输入指令 `tensorboard --logdir="./data/log"` ，启动服务器。

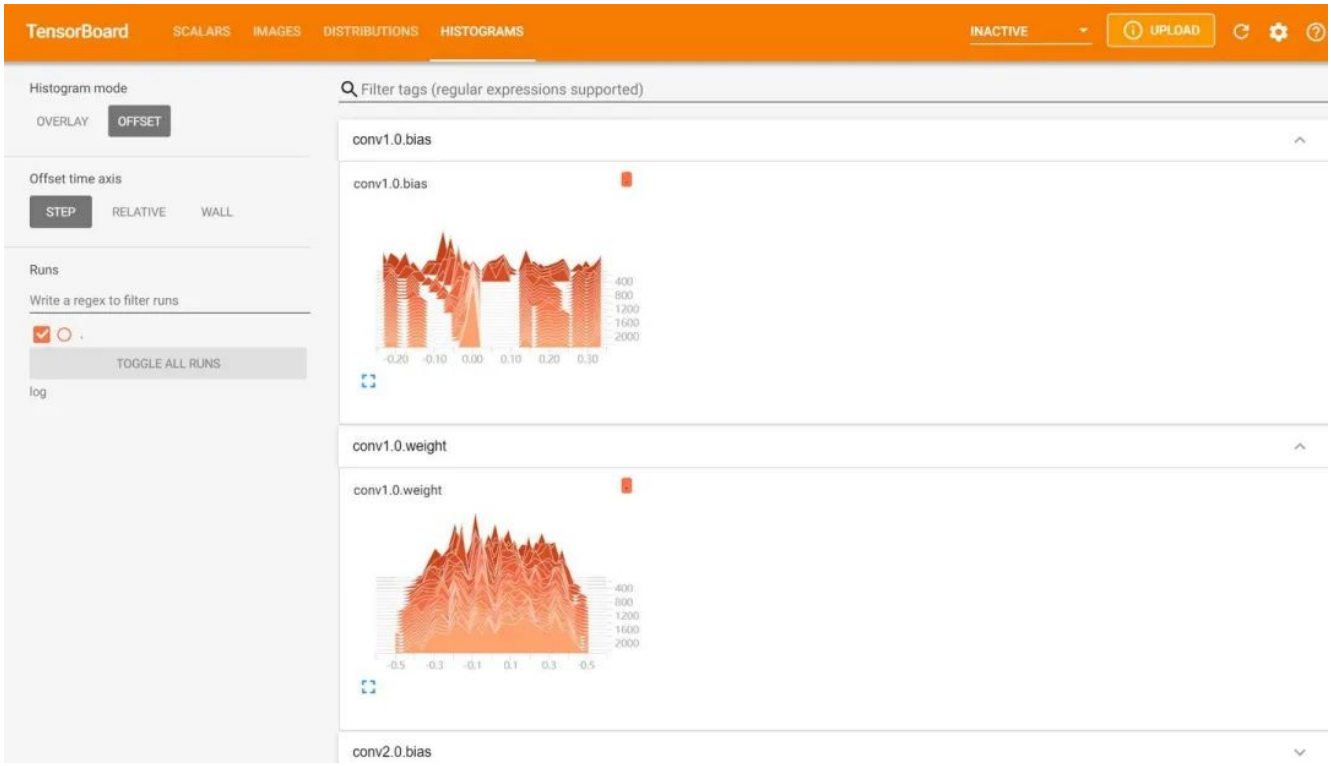


> logdir后面的参数是日志文件的文件夹的路径

然后在谷歌浏览器中访问红框框中的url，便可得到可视化界面，点击上面的页面控件，可以查看我们通过 `add_scalar` 、 `add_image` 和 `add_histogram` 得到的图像，而且各方面做得都很丝滑。





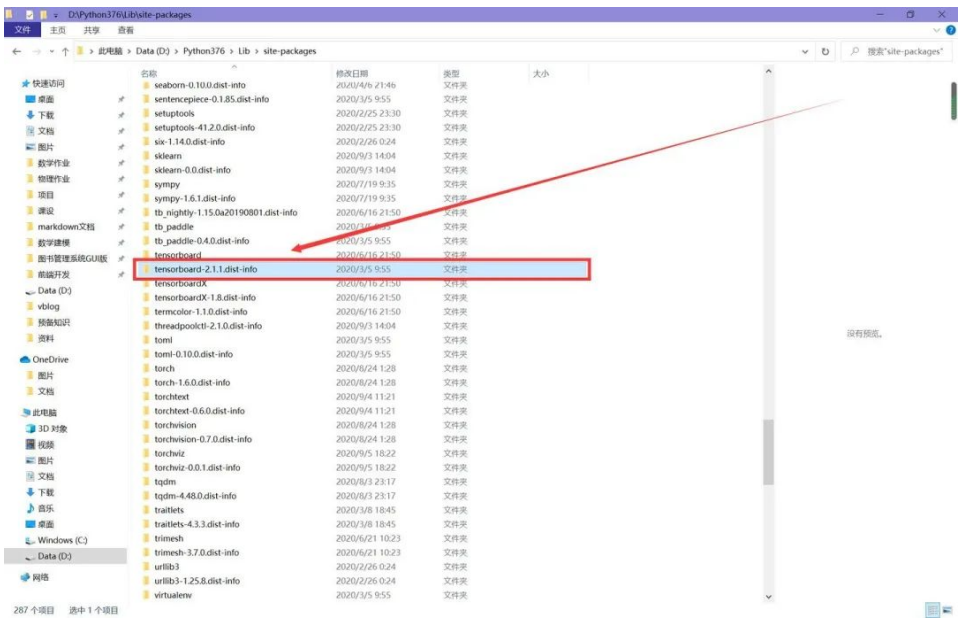


> 以下是笔者安装使用tensorboard时遇到的一些错误。

好，作为一名**没有装过TensorFlow**的windows玩家，笔者下面开始踩坑。踩完后，直接把几个可能的错误呈上。

第一个错误，运行 `tensorboard --logdir="./data/log"` ，遇到报错，内容为有重复的tensorboard的包。

解决方法：找到site-packages（如果你是像我一样全局安装的，那么找到解释器那一级目录的site-packages，如果是在项目虚拟环境中安装的，那么找到项目中的site-packages），删去下图中红框框标出来的文件夹。



第二个错误，在解决第一个错误后，再次运行命令，还是报错，内容为编码出错。由于笔者做过一点前端，在学习webpack项目时，曾经被告知项目路径不能含有中文，否则会有编码错误，而刚才的报错中涉及到了前端服务器的启动，因此，笔者想到从文件名入手。

解决方法：确保命令涉及的文件路径、所有程序涉及到文件不含中文。笔者是计算机名字含有中文，然后tensorboard的日志文件是以本地计算机名为后缀的，所以笔者将计算机名修改成了英文，重启后再输入指令就ok了。

2.2 HiddenLayer可视化训练过程

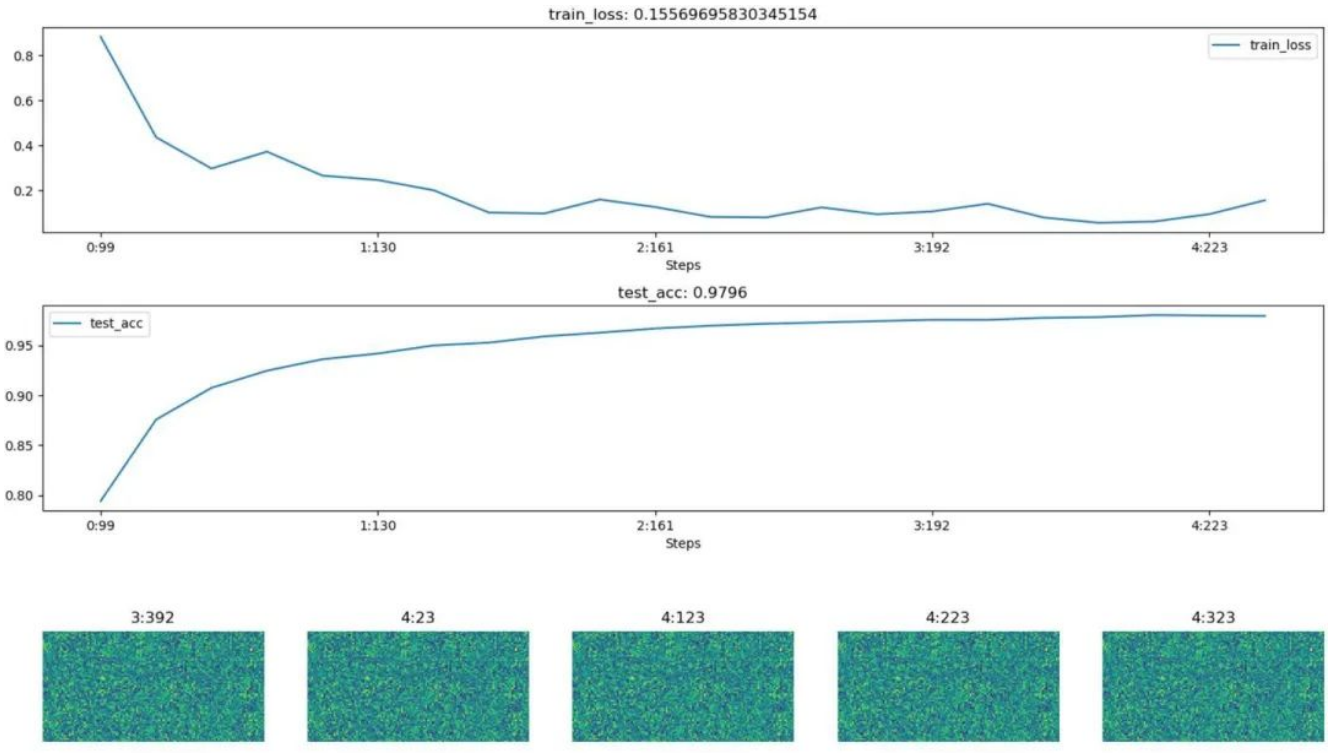
tensorboard的图像很华丽，但是使用过程相较于其他的工具包较为繁琐，所以小网络一般没必要使用tensorboard。

```
import hiddenlayer as hl
import time

# 记录训练过程的指标
history = hl.History()
# 使用canvas进行可视化
canvas = hl.Canvas()

# 获取优化器和损失函数
optimizer = torch.optim.Adam(MyConvNet.parameters(), lr=3e-4)
loss_func = nn.CrossEntropyLoss()
log_step_interval = 100       # 记录的步数间隔

for epoch in range(5):
    print("epoch:", epoch)
    # 每一轮都遍历一遍数据加载器
    for step, (x, y) in enumerate(train_loader):
        # 前向计算->计算损失函数->(从损失函数)反向传播->更新网络
        predict = MyConvNet(x)
        loss = loss_func(predict, y)
        optimizer.zero_grad()   # 清空梯度（可以不写）
        loss.backward()      # 反向传播计算梯度
        optimizer.step()     # 更新网络
        global_iter_num = epoch * len(train_loader) + step + 1  # 计算当前是从训练开始时的第
        if global_iter_num % log_step_interval == 0:
            # 控制台输出一下
            print("global_step:{}, loss:{:.2}".format(global_iter_num, loss.item()))
```

```
# 在测试集上预测并计算正确率
test_predict = MyConvNet(test_data_x)
_, predict_idx = torch.max(test_predict, 1)  # 计算softmax后的最大值的索引，即预测
acc = accuracy_score(test_data_y, predict_idx)

# 以epoch和step为索引，创建日志字典
history.log((epoch, step),
            train_loss=loss,
            test_acc=acc,
            hidden_weight=MyConvNet.fc[2].weight)

# 可视化
with canvas:
    canvas.draw_plot(history["train_loss"])
    canvas.draw_plot(history["test_acc"])
    canvas.draw_image(history["hidden_weight"])
```

> 不同于tensorboard，hiddenlayer会在程序运行的过程中动态生成图像，而不是模型训练完后

下面为模型训练的某一时刻的截图：



三、使用Visdom进行可视化

Visdom是Facebook为pytorch开发的一块可视化工具。类似于tensorboard，visdom也是通过在本地启动前端服务器来实现可视化的，而在具体操作上，visdom又类似于matplotlib.pyplot。所以使用起来很灵活。

首先先安装visdom库，然后补坑。由于启动前端服务器需要大量依赖项，所以在第一次启动时可能会很慢（需要下载前端三板斧的依赖项），解决方法请见这里。

先导入需要的第三方库：

```
from visdom import Visdom
from sklearn.datasets import  load_iris
import torch
import numpy as np
from PIL import Image
```

matplotlib里，用户绘图可以通过plt这个对象来绘图，在visdom中，同样需要一个绘图对象，我们通过 `vis = Visdom()` 来获取。具体绘制时，由于我们会一次画好几张图，所以visdom要求用户在绘制时指定当前绘制图像的窗口名字（也就是 `win` 这个参数）；除此之外，为了到时候显示的分块，用户还需要指定绘图环境 `env` ，这个参数相同的图像，最后会显示在同一张页面上。

绘制线图（相当于matplotlib中的 `plt.plot` ）

```
# 绘制图像需要的数据
iris_x, iris_y = load_iris(return_X_y=True)

# 获取绘图对象，相当于plt
vis = Visdom()

# 添加折线图
x = torch.linspace(-6, 6, 100).view([-1, 1])
sigmoid = torch.nn.Sigmoid()
sigmoid_y = sigmoid(x)
tanh = torch.nn.Tanh()
tanh_y = tanh(x)
relu = torch.nn.ReLU()
relu_y = relu(x)
# 连接三个张量
plot_x = torch.cat([x, x, x], dim=1)
plot_y = torch.cat([sigmoid_y, tanh_y, relu_y], dim=1)
```

```
# 绘制线性图
vis.line(X=plot_x, Y=plot_y, win="line plot", env="main",
         opts={
             "dash" : np.array(["solid", "dash", "dashdot"]),
             "legend" : ["Sigmoid", "Tanh", "ReLU"]
         })
```

绘制散点图：

```
# 绘制2D和3D散点图
# 参数Y用来指定点的分布，win指定图像的窗口名称，env指定图像所在的环境，opts通过字典来指定一些样式
vis.scatter(iris_x[ : , 0 : 2], Y=iris_y+1, win="windows1", env="main")
vis.scatter(iris_x[ : , 0 : 3], Y=iris_y+1, win="3D scatter", env="main",
            opts={
                "markersize" : 4,    # 点的大小
                "xlabel" : "特征1",
                "ylabel" : "特征2"
            })
```

绘制茎叶图：

```
# 添加茎叶图
x = torch.linspace(-6, 6, 100).view([-1, 1])
y1 = torch.sin(x)
y2 = torch.cos(x)

# 连接张量
plot_x = torch.cat([x, x], dim=1)
plot_y = torch.cat([y1, y2], dim=1)
# 绘制茎叶图
vis.stem(X=plot_x, Y=plot_y, win="stem plot", env="main",
         opts={
             "legend" : ["sin", "cos"],
             "title" : "茎叶图"
         })
```

绘制热力图：

```
# 计算鸢尾花数据集特征向量的相关系数矩阵
iris_corr = torch.from_numpy(np.corrcoef(iris_x, rowvar=False))
# 绘制热力图
vis.heatmap(iris_corr, win="heatmap", env="main",
            opts={
                "rownames" : ["x1", "x2", "x3", "x4"],
                "columnnames" : ["x1", "x2", "x3", "x4"],
                "title" : "热力图"
            })
```

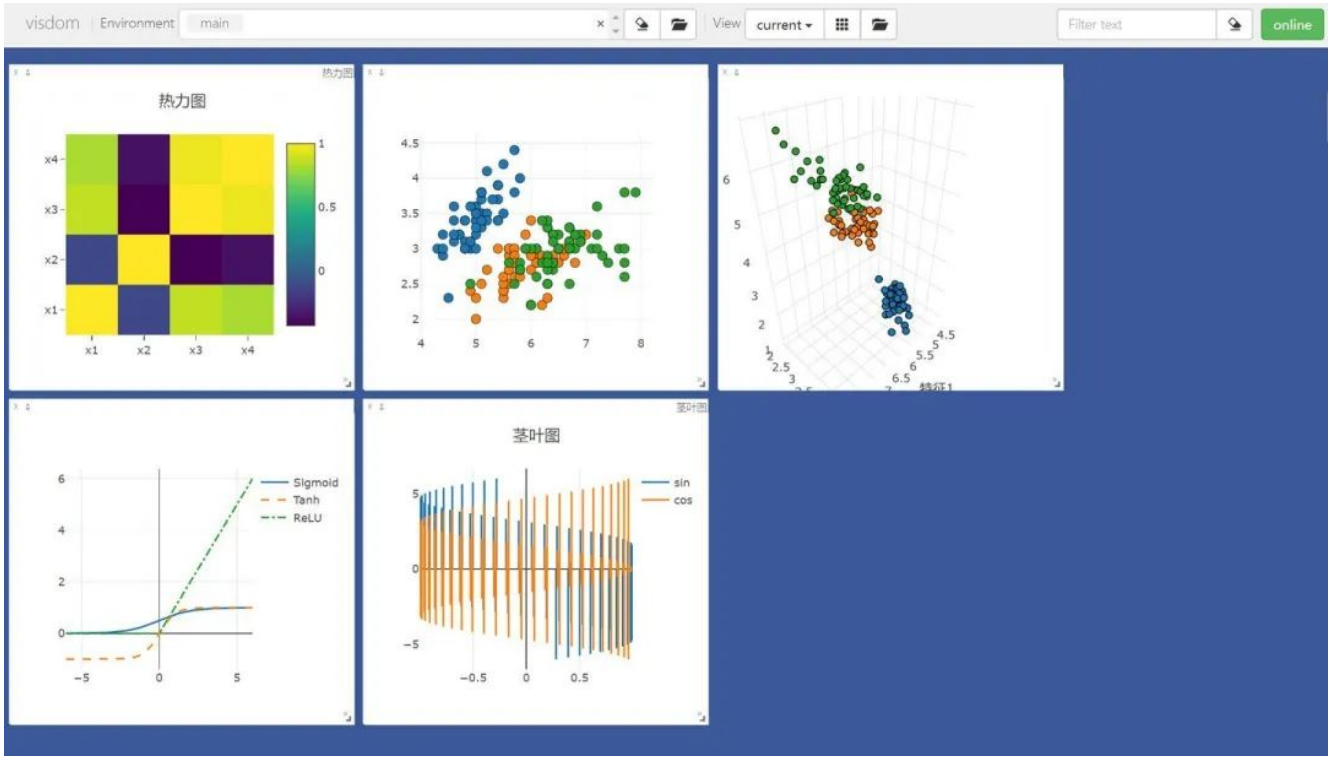
可视化图片，这里我们使用自定义的env名MyPlotEnv

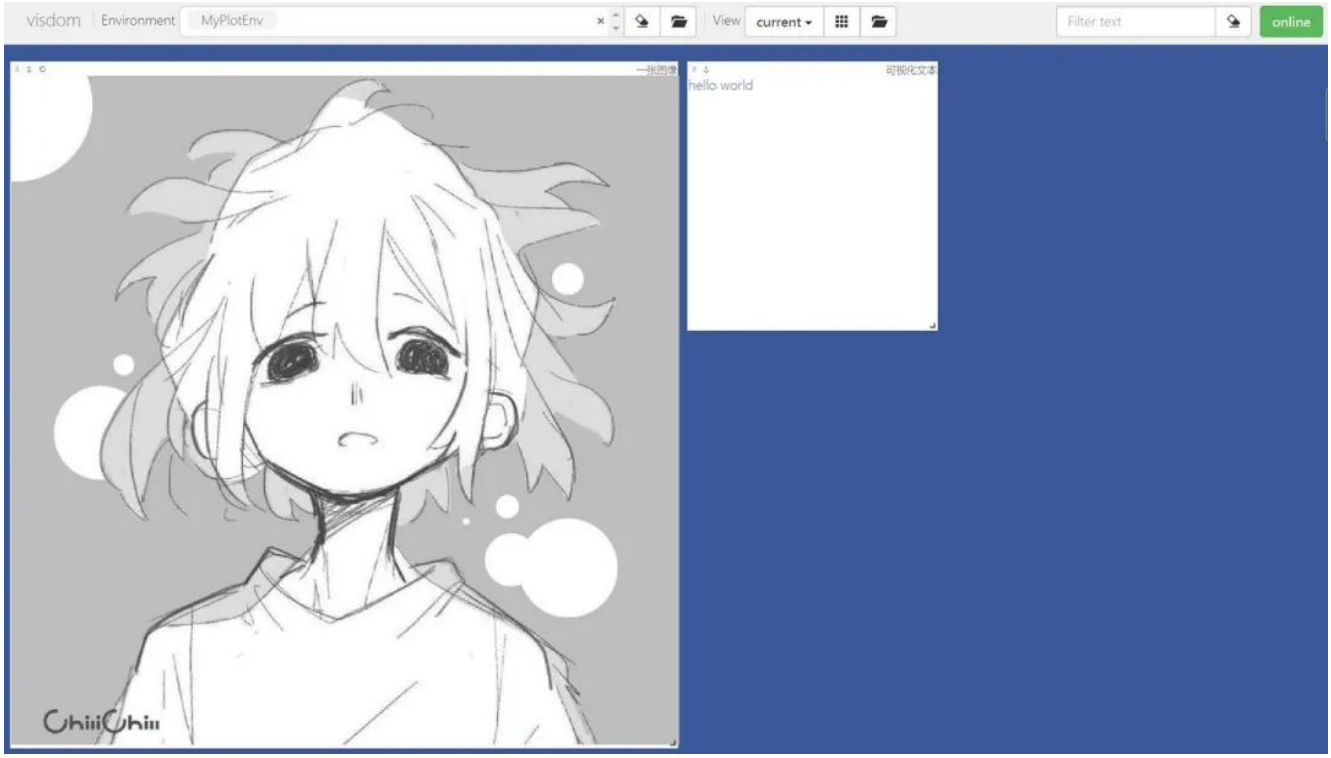
```
# 可视化图片
img_Image = Image.open("./example.jpg")
img_array = np.array(img_Image.convert("L"), dtype=np.float32)
img_tensor = torch.from_numpy(img_array)
print(img_tensor.shape)

# 这次env自定义
vis.image(img_tensor, win="one image", env="MyPlotEnv",
          opts={
              "title" : "一张图像"
          })
```

可视化文本，同样在MyPlotEnv中绘制：

```
# 可视化文本
text = "hello world"
vis.text(text=text, win="text plot", env="MyPlotEnv",
         opts={
             "title" : "可视化文本"
         })
```

运行上述代码，再通过在终端中输入 `python3 -m visdom.server` 启动服务器，然后根据终端返回的URL，在谷歌浏览器中访问这个URL，就可以看到图像了。





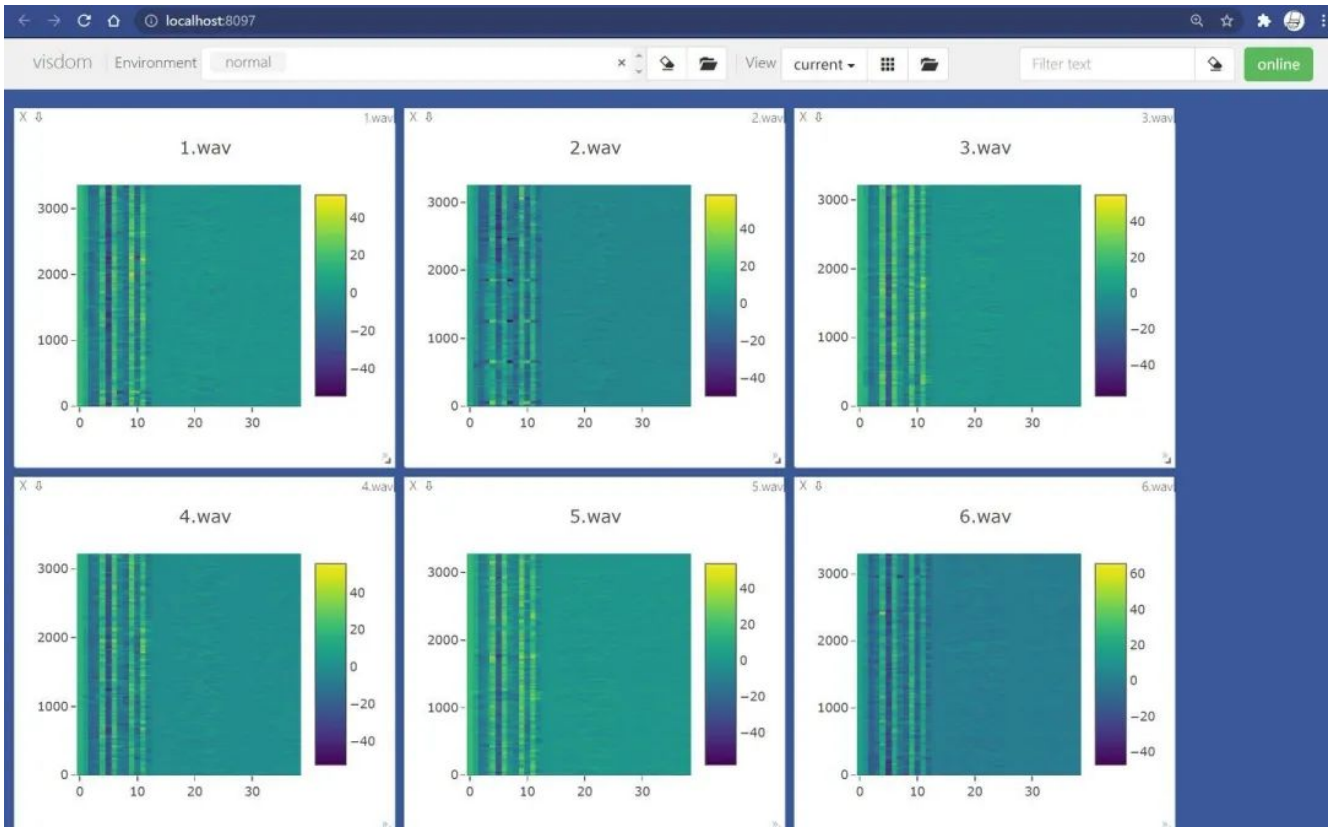
> 在Environment中输入不同的env参数可以看到我们在不同环境下绘制的图片。对于分类图集特别有用。

在终端中按下Ctrl+C可以终止前端服务器。

进一步

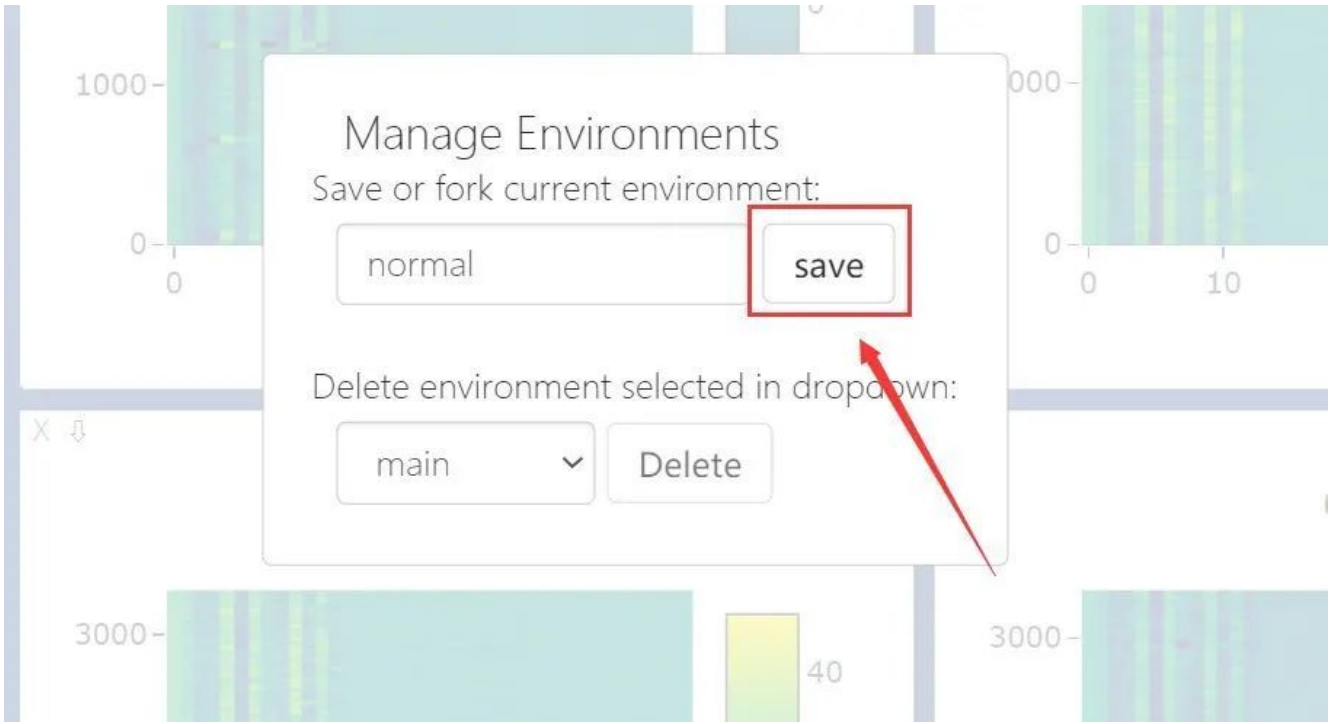
需要注意，如果你的前端服务器停掉了，那么所有的图片都会丢失，因为此时的图像的数据都是驻留在内存中，而并没有dump到本地磁盘。那么如何保存当前visdom中的可视化结果，并

在将来复用呢？其实很简单，比如我现在有一堆来之不易的Mel频谱图：

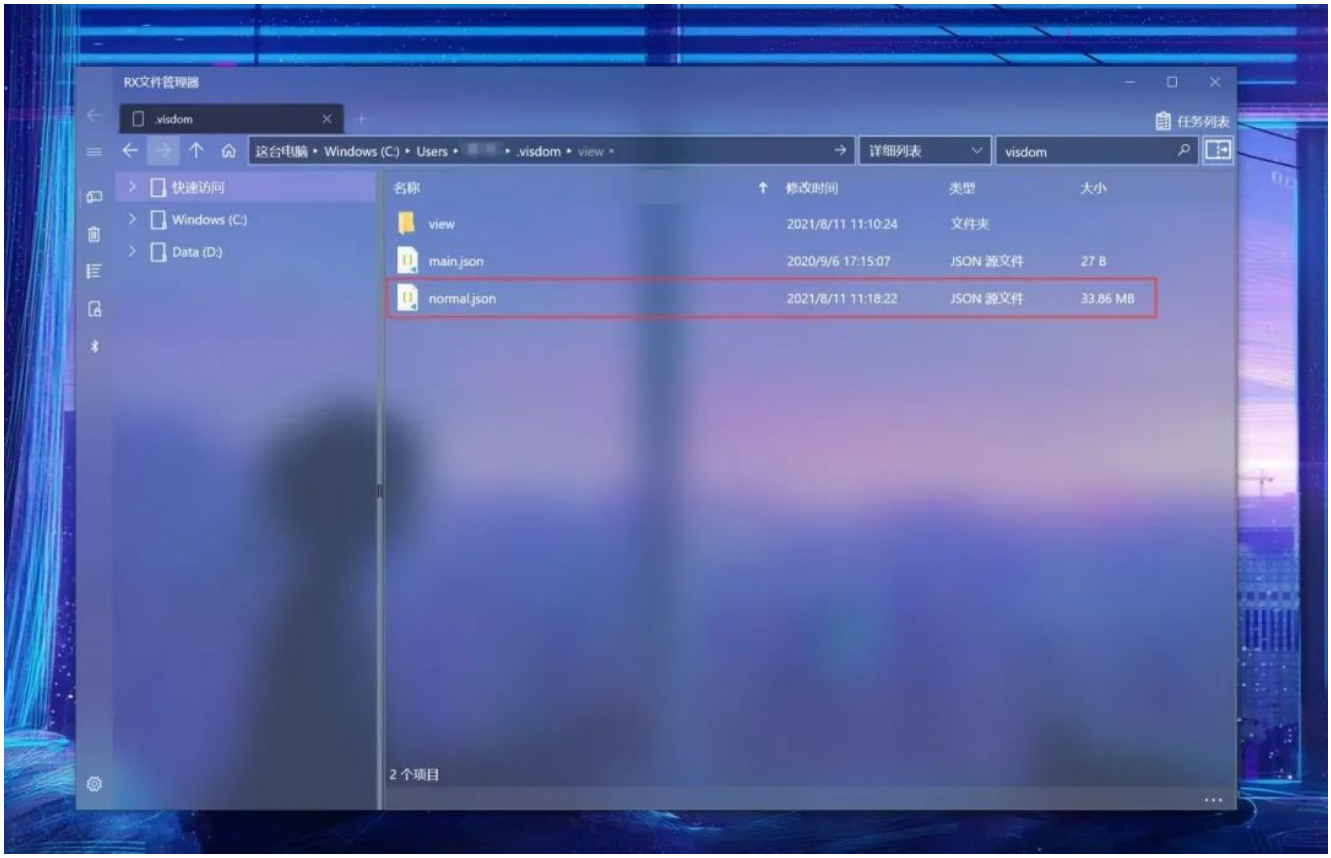


点击Manage Views

点击fork->save:（此处我只保存名为normal的env）



接着，在你的User目录下（Windows是C:\Users\账户.visdom文件夹，Linux是在~.visdom文件夹下），可以看到保存好的env：



它是以json文件格式保存的，那么如果你保存完后再shut down当前的前端服务器，图像数据便不会丢失。

好的，现在在保存完你珍贵的数据后，请关闭你的visdom前端服务器。然后再启动它。

如何查看保存的数据呢？很简答，下次打开visdom前端后，visdom会在.visdom文件夹下读取所有的保存数据完成初始化，这意味着，你直接启动visdom，其他什么也不用做就可以看到之前保存的数据啦！

那么如何服用保存的数据呢？既然你都知道了visdom保存的数据在哪里，那么直接通过python的json包来读取这个数据文件，然后做解析就可以了，这是方法一，演示如下：

```
import json

with open(r"...\.visdom\normal.json", "r", encoding="utf-8") as f:
    dataset : dict = json.load(f)

jsons : dict = dataset["jsons"]       # 这里存着你想要恢复的数据
reload : dict = dataset["reload"]     # 这里存着有关窗口尺寸的数据

print(jsons.keys())      # 查看所有的win
```

out:

```
dict_keys(['jsons', 'reload'])
dict_keys(['1.wav', '2.wav', '3.wav', '4.wav', '5.wav', '6.wav', '7.wav', '8.wav', '9.wav'
```

但这么做不是很优雅，所以visdom封装了第二种方法。你当然可以通过访问文件夹.visdom来查看当前可用的env，但是也可以这么做：

```
from visdom import Visdom

vis = Visdom()
print(vis.get_env_list())
```

out:

```
Setting up a new session...
['main', 'normal']
```

在获取了可用的环境名后，你可以通过get_window_data方法来获取指定env、指定win下的图像数据。请注意，该方法返回str，故需要通过json来解析：

```
from visdom import Visdom
import json

vis = Visdom()

window = vis.get_window_data(win="1.wav", env="normal")
window = json.loads(window)          # window 是 str，需要解析为字典

content = window["content"]
data = content["data"][0]
print(data.keys())
```

out:

```
Setting up a new session...
dict_keys(['z', 'x', 'y', 'zmin', 'zmax', 'type', 'colorscale'])
```

通过索引这些keys，相信想复用原本的图像数据并不困难。

公众号后台回复“CVPR2023”获取最新论文分类整理资源

极市干货

极视角动态：「无人机+AI」光伏智能巡检，硬核实力遇见智慧大脑！ | 「AI 警卫员」上线，极视角守护龙大食品厂区安全！ | 点亮海运指明灯，极视角为海上运输船员安全管理保驾护航！

CVPR2023：CVPR'23 最新 125 篇论文分方向整理 | 检测、分割、人脸、视频处理、医学影像、神经网络结构、小样本学习等方向

数据集：自动驾驶方向开源数据集资源汇总 | 医学影像方向开源数据集资源汇总 | 卫星图像公开数据集资源汇总

点击阅读原文进入CV社区

收获更多技术干货

阅读原文

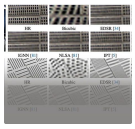
喜欢此内容的人还喜欢

YOLOv5帮助母猪产仔？南京农业大学研发母猪产仔检测模型并部署到Jetson Nano开发板

极市平台

ICCV 2023 | 南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）

极市平台



实践教程 | 使用 OpenCV 进行特征提取（颜色、形状和纹理）

极市平台