第八、九周工作总结

张泽宇

2022 年 4 月 24 日

过去两周的主要进行的工作包括有：

- **将之前尝试的灰度图像转换程序进行了完善，可由 PSCAD 输出的.out 文件直接转换成灰度图。**
- **利用灰度图进行网络训练，但是最终的识别效果不好，从不同方面尝试改进。**

以下为详细叙述

---

1 灰度图像转换程序

程序的主要构造思想如下：

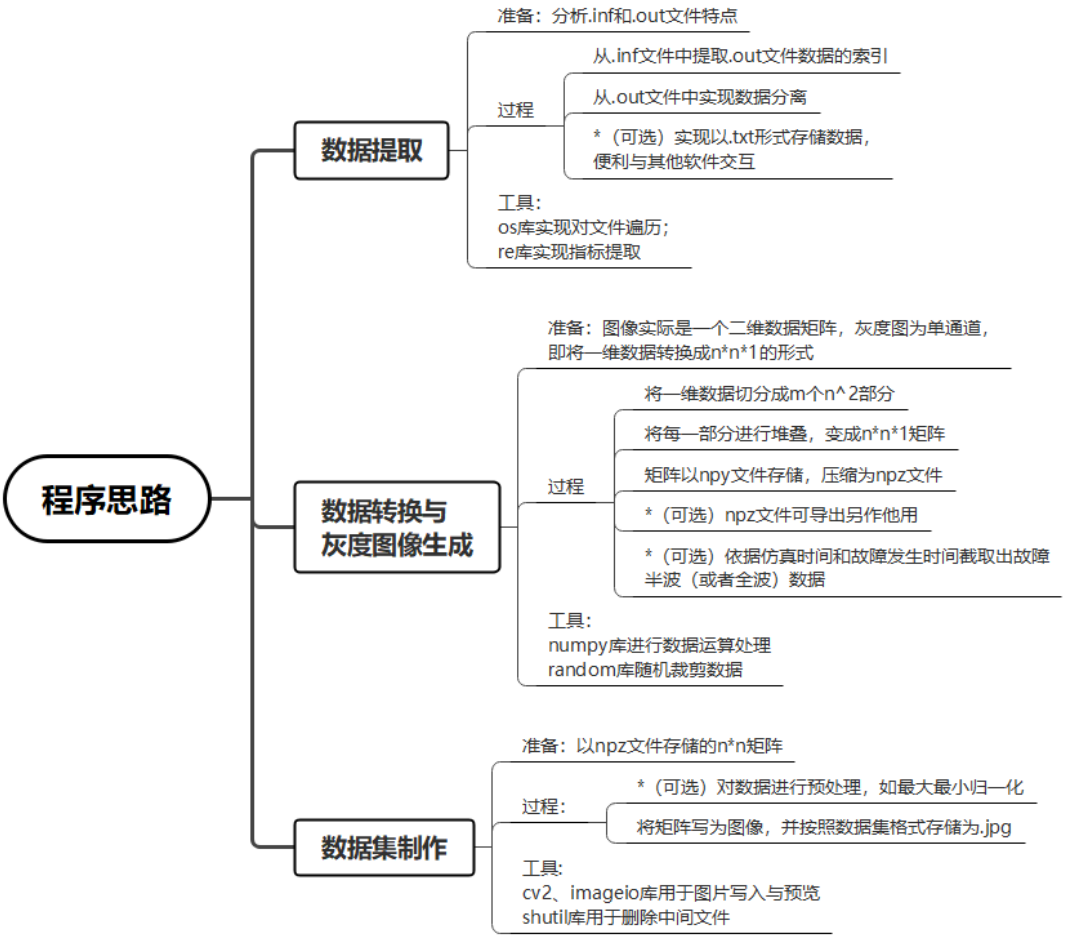

图 1: 程序思路

现就每一部分程序进行注解。

1.1 数据提取

首先分析一下 PSCAD 输出数据文件的特点。上几周处理数据时，一直都在关注.infx 文件，如下图 2所示，其中的数据还是比较繁琐的，有很多不需要关注的信息。后来，通过查阅书籍，发现应该从 inf 文件中提取相关数据索引。

如图 3所示，其中 PGB(i) 表示.out 文件中的第 i+1 列（第 1 列是时间）；Desc 表示该列数据名称，Unit 表示单位。因此，利用 os 库对数据文件进行遍历打开读取操作，采用正则表达式实现对数据名称的提取，并利用列索引将具体数值提取出来，最后以字典的形式暂时存储数据，即可实现对数据的提取。

这部分代码如下所示：

```
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
<Output device="EMTDC" version="2010" date="2022/04/17" time="09:00:38.000000">
  <Domain name="Time" unit="s" mult="0.0" skew="0.0">
    <Sample rate="100000.0" end="10000" />
  </Domain>
  <List classid="Analog">
    <Analog name="Main(0):Is" index="0" id="1763691693:0" label="" dim="3" unit="" min="-2.0" max="2.0" />
    <Analog name="Main(0):Us" index="3" id="1026695447:0" label="" dim="3" unit="" min="-2.0" max="2.0" />
    <Analog name="Main(0):Fault Type" index="6" id="1077324046:0" label="Fault Monitoring" dim="1" unit="" mir
    <Analog name="Main(0):Fault Delay" index="7" id="637018475:0" label="Fault Monitoring" dim="1" unit="s" mi
    <Analog name="Main(0):Is" index="8" id="1622570220:0" label="Bus Current" dim="3" unit="kA" min="-10" ma
    <Analog name="Main(0):Vs" index="11" id="1545599719:0" label="Bus Voltage" dim="3" unit="kV" min="-200"
  </List>
  <List classid="Digital" />
</Output>
```

图 2: .infx 文件

```
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
PGB(1)      Output  Desc="FaultType"  Group="Main"  Max=2.0  Min=-2.0  Units=""
PGB(2)      Output  Desc="FDelay"  Group="Main"  Max=2.0  Min=-2.0  Units=""
PGB(3)      Output  Desc="I:1"  Group="Bus Current"  Max=10  Min=-10  Units="kA"
PGB(4)      Output  Desc="I:2"  Group="Bus Current"  Max=10  Min=-10  Units="kA"
PGB(5)      Output  Desc="I:3"  Group="Bus Current"  Max=10  Min=-10  Units="kA"
PGB(6)      Output  Desc="U:1"  Group="Bus Voltage"  Max=200  Min=-200  Units="kV"
PGB(7)      Output  Desc="U:2"  Group="Bus Voltage"  Max=200  Min=-200  Units="kV"
PGB(8)      Output  Desc="U:3"  Group="Bus Voltage"  Max=200  Min=-200  Units="kV"
```

图 3: .inf 文件

```python
def read_variable_name(f_path):
    L = {}
    f = open(f_path, 'r', encoding='utf-8')
    for line in f:
        reformat = re.compile(r"PGB\((?P<PGB>.*?)\)(.*?)"
        r"(.*?)Desc=\"(?P<Desc>.*?)\"", re.S)
        index = reformat.finditer(line)
        for item in index:
            L[item.group("PGB")] = item.group("Desc").replace(":", "_")
    f.close()
    return L

def read_simulation_time(f_path):
    f = open(f_path, 'r', encoding='utf-8')
    L = []
    for line in f:
        line = line.replace("\n", '')
        line = re.split(r"[ ]+", line)
        if line[1]:
            L.append(line[1])
        else:
            continue
```

```
        f.close()
        return L

    def read_variable_value(f_path, index):
        f = open(f_path, 'r', encoding='utf-8')
        data_temp = []
        for line in f:
            line = line.replace("\n", '')
            line = re.split(r"[ ]+", line)
            data_temp.append(line)
        f.close()
        column_number = list(index.keys())
        column_name = list(index.values())
        dir_temp = {}
        for i in range(len(column_name)):
            L = []
            for line in data_temp:
                if line[1]:
                    L.append(eval(line[eval(column_number[i]) + 1]))
                else:
                    continue
            dir_temp[column_name[i]] = L
        return dir_temp
```

程序中有一些步骤是为了处理文件中的细节，如开头空行、间隔不等等问题，待完善注释后上传到 Github 中。最终返回的 dir_temp 是一个字典类型，key 是每个项目，如 I_1，value 是对应的值。

1.2 数据转换与图像生成

首先把一维数据信号截成需要的长度（这里以 64×64=4096 为例）

```
    def turn_grayscale(data):
        lens = len(data)
        max_start = lens - dimension_grayscale ** 2
        starts = []
        gray_scales = []
        for i in range(sampling_value):
        while True:
            start = random.randint(0, max_start)
            if start not in starts:
                starts.append(start)
                break
            temp = data[start: start + dimension_grayscale ** 2]
            temp = np.array(temp)
            gray_temp = temp.reshape(dimension_grayscale, dimension_grayscale)
            gray_scales.append(gray_temp)
        return gray_scales

```

这里面 starts 用来存储开始点,共有 sampling_value 个开始点,开始点往后 dimension_grayscale ** 2 个数据 (这里以 64^2 为例) 提取出来，并转换成 ndarray 数据形式，再利用 reshape 函数变成二维矩阵。最终由 gray_scales 收集所有的二维矩阵。

其次考虑矩阵数据存贮的问题。这里之所以单独将矩阵数据存储出来，是为了方便其他程序的调用。例如，如果采用 PyTorch 读取数据，直接读.npz 文件要由于读取.img 文件。所以加入这段程序, 最终数据存在 npz. 压缩文件中。需要使用时，直接采用 load 函数即可解压为 npy 文件并读取成 ndarray 数据类型。

```
    def draw_grayscale(graydata, name_npz, up_name, upp_name, uppp_name):
        path_name = r''
        if os.path.exists(path_name):
            np.savez(path_name + '{}'.format(name_npz), *graydata)
        else:
            os.makedirs(path_name)
            np.savez(path_name + '{}'.format(name_npz), *graydata)
        return
```

1.3 数据集制作

这部分相对而言较为灵活，利用 os 库依据需求创建文件夹，并利用 imageio 向其中写入文件，即可得到所需要的数据集。这里数据集的组成方式，也可以灵活设定，有如下两种方式:

- 分为 Train 和 Test 两个文件夹，其下有已分类依据（如故障类型）为名的子文件夹，子文件夹下存储相应的图像文件。

  这类方法主要适用于 Matlab 平台的神经网络。

- 分为 Train_Image,Test_Image,Train_Label,Test_Label 四个文件夹，其中前两个文件夹用于存储照片，后两个以 txt 文本形式存储标签信息，如故障类型、故障时间、故障地点等。图像文件与文本文件依靠同名的形式相关联。

  这种方式更多应用于 PyTorch 框架。

```
    def divide_dataset(filename, up_name, upp_name, uppp_name):
        npz_file = np.load('',allow_pickle=True)
        for file in npz_file:
            temp = npz_file[file]
            temp = temp.astype(np.float64)
            if up_name == 1:
                path_dir = 'DataSet\\1\\'
```

```
8                elif up_name == 8:
                     path_dir = 'DataSet\\8\\'
10               else:
                     continue
12               img_name = "{0}_{1}_{2}_{3}".format(uppp_name, up_name, upp_name, filename)
                 if os.path.exists(path_dir):
14                   imageio.imwrite(path_dir + img_name + '{}.jpg'.format(file), temp)
                 else:
16                   os.makedirs('' + path_dir)  imageio.imwrite(path_dir + img_name +  ...
         '{}.jpg'.format(file), temp)
             return
18
```

以上即为程序的主要部分（除去一些细枝末节），个人认为这个程序还是具有一定的普适性。当然，程序在一些处理上也存在有可以继续优化提升速率的地方。

2 对灰度图的训练

灰度图数据的来源依然是上一次搭建的 10kV 接地小电阻模型。

首先，设置了 9 个故障点，每个故障点发生 A 相短路接地、AB 相短路接地、三相短路接地、AB 相短路四种故障；故障发生时间由起始时间 0.01s 和延迟时间决定，延迟时间从 0 开始以 0.001 为步长变化到 0.01. 因此总的故障发生时间在 0.01 0.02s 之间；故障持续时间设置为 0.01s，即半个周期。

然后，收集 0 0.05s 的故障电流与电压数据。利用 Multiple Run 模块，设置自动运行，最终得到了 396 次仿真结果，取其中的 I_1 即 A 相电流作为指标生成灰度图。每次运行数据长度为 5003 个，随机取其中 4096 个数据生成 100 个 64 像素灰度图，用于数据增强。

最后，为简便起见，先判断故障类型。取发生于 T1 处的数据生成的灰度图作为数据集。如下图所示：

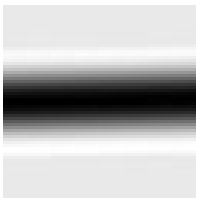

图 5: A 相短路接地

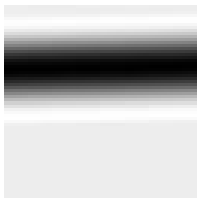

图 6: AB 相短路接地

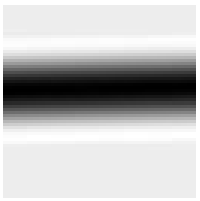

图 7: ABC 三相短路接地

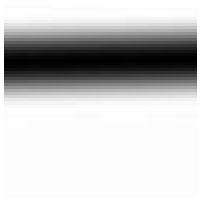

图 8: AB 相短路

发现问题，每个图像之间似乎相似度比较高，只是黑色带出现的位置稍有不同。怀着侥幸心理，利用 matlab 搭建了网络，结构如下图：

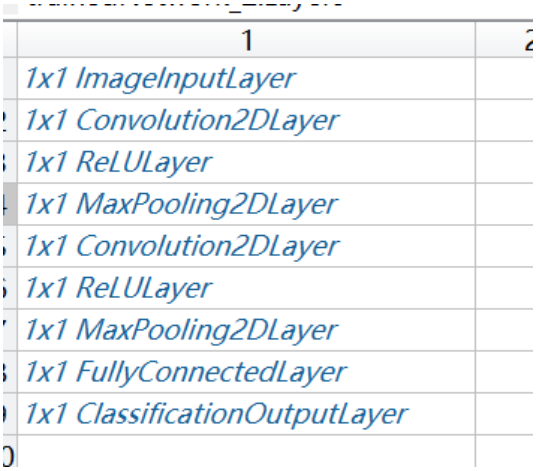

图 8: 神经网络

进行训练，最初出现了损失值为 NAN 的问题。显示是损失值达到 NAN，寻来你只进行了两轮迭代就停止。通过网上论坛，可能的原因一个是学习率太大，我把学习率由 0.01 降低为 0.005，但依旧无法解决；另一个原因可能是梯度爆炸的问题发生。解决梯度爆炸，由于这个结构中已经采用了 ReLU 激活函数，在不改变网络结构的前提下，只能进行梯度裁剪。所以我设定了一个梯度阈值，最终不再报错。但是尽管如此，说明网络的设计还是有一些问题。

训练结果也说明了这个问题。最后的损失值不收敛，训练集准确率基本不变化，在 20% 到 30% 之间浮动，验证集的准确率是 25%。

显然，25% 恰为 $\frac{1}{4}$，模型识别不了故障类型。

出现这样的表现，我认为可能的问题出在两点，也有可能两方面都有：

- 数据集的问题。

  事实上，单纯人工去看数据集，发现每个图片的区别实在不能算作显著。下面两幅图是 A 相单相接地和 AB 相短路的图：

  发现图像之间太过于相像，自然训练不出什么结果。输出 B，C 相电流的灰度图，也是基本如此；而 ABC 三相的电压灰度图，只有三相短路接地的情况下图像有两条黑纹，其他也是类似。我返回去重新看了一下数据结果，发现出现故障的地方，无论是电压还是电流量，大小与正常情况下有大概 10 的二次方倍数量级，我猜测是这里出现的问题。

  因为在转换成灰度图的过程中，矩阵中的每个值按照一定的算法映射在 0-256 之间，正常与故障之间相差太大，就会导致映射之后正常的数据集中在 256 附件，故障数据集中在 0 附近，这样造成了图片中黑白分明，都有一道黑纹的现象。

  为了验证我的猜想，我用 opencv 读取灰度图，得到的是 ndarray 类型数据，然后将灰度矩阵中的每个元素按照最大最小归一化处理再乘 256，两个矩阵进行比较，发现每个元素数值相近。这就说明在生成灰度图的过程中，imageio 库采用了最大最小算法或者类似的算法，对每一个数据进行处理，这样的处理是等比例间距的，即使在 0-256 的空间上，依

图 10: A 相短路接地

图 11: AB 相短路接地

然保留了每个元素在原始空间上的相对位置。所以，要想解决，一方面可以通过模型的修改，使这个数据之间的数量级差别降低，另一方面可以通过算法的修改，（比方采用标准差方式映射到 0-256 之间？只是一个猜想）。这两个方向下一周计划进行尝试。

- 另一个方面，在神经网络上。

  梯度爆炸的产生是因为反向传播的过程中梯度不断累计，以指数级增长，导致网络无法训练。采用 ReLU 激活函数是解决的一个办法，但是在本例中显然 ReLU 没有发挥应有的作用。但是反过来看网络的结构，参数设置并没有很复杂，网络深度也不大，那出现梯度爆炸的原因，会不会是由于数据的问题？这一点还待探讨。

  关于灰度图的问题，我在网上也找了一些资料，其中有一个“基于卷积神经网络的数据驱动故障预测方法”[1]，这个方法也是采用灰度图进行故障检测，我利用它的数据生成了灰度图如下：

  能看得出来，这两类的灰度图之间差别还是比较明显的。因此我认为主要的问题还是出现在对数据的处理上，可能神经网络是没有问题，而是由于数据太过于特殊导致无法正常训

[1] https://www.sci-hub.ren/10.1109/tie.2017.2774777

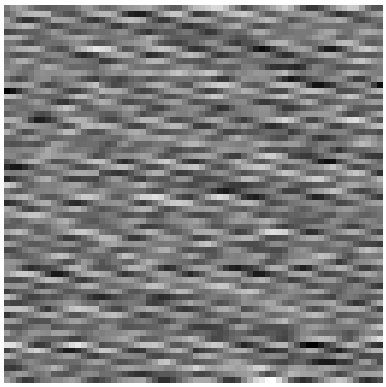

图 12: A 类

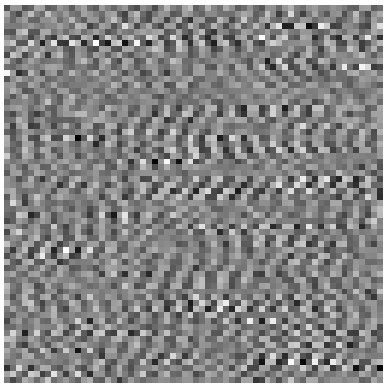

图 13: B 类

练。

总而言之，下一周将会对数据集的处理进行进一步探讨。