数据挖掘竞赛报告----轴承故障诊断训练赛

饶逸卓 计算机学院 XS19020003

一、 问题的描述

1. 问题背景

轴承是在机械设备中具有广泛应用的关键部件之一。由于过载，疲劳，磨损，腐蚀等原因，轴承在机器操作过程中容易损坏。所以我们需要对轴承的状态进行监测和分析。通常我们会监测轴承的振动信号。

本次比赛提供一个真实的轴承振动信号数据集，需要我们使用机器学习技术判断轴承的工作状态。

轴承的故障一般有 3 种故障：外圈故障，内圈故障，滚珠故障，外加正常的工作状态。如表 1 所示，结合轴承的 3 种直径（直径 1,直径 2,直径 3），轴承的工作状态有 10 类如表 1 所示。

表 1 轴承故障类别

	外圈故障	内圈故障	滚珠故障	正常
直径 1	1	2	3	0
直径 2	4	5	6	
直径 3	7	8	9	

我们需要设计一个模型根据轴承运行过程中的振动信号对轴承的工作状态进行分类。

2. 问题数据

本次竞赛提供了两个用 csv 形式存放的数据集 train.csv 和 test_data.csv。训练集数据的 1 到 6000 为按时间序列连续采样的振动信号数值，每行数据是一个样本，共 792 条数据，第一列 id 字段为样本编号，最后一列 label 字段为标签数据，即轴承的工作状态，用数字 0 到 9 表示。测试集数据，共 528 条数据，除无 label 字段外，其他字段同训练集。

3. 模型评价指标

本次任务的评价指标采用的是 F1 指标的算术平均值，它是 Precision 和 Recall 的调和平均数。其中，Pi 是表示第 i 个种类对应的 Precision， Ri 是表示第 i 个种类对应 Recall，如（1）式所示。

$$\langle F1\rangle = \frac{1}{n}\sum_{n}^{1} F1_i = \frac{1}{n}\sum_{n}^{1}\frac{2\cdot P_i\cdot R_i}{P_i + R_i} \tag{1}$$

二、 求解思路

首先根据问题的描述，可以看出，这是一个多分类的问题，并且数据维度只有一维，所以我们可以采用卷积神经网络来解决它。

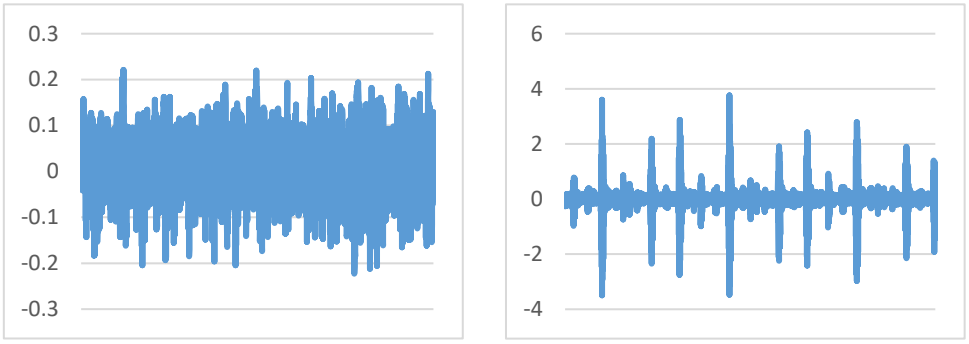
我们的设计思路就是设计一个没有反馈回路的简单序贯神经网络模型，将每一行 6000 个数据作为神经网络的一个输入，然后得到 10 个不同的分类。我们假定 6000 行

数据是独立同分布的。

三、 求解过程

1. 数据预处理

首先观察数据，我们提取出一次采样的一组数据一共 6000 个按序列画出来如下图 1 的(a)和(b)所示。



(a) (b)

图 1 一组振动数据，(a) 和(b)分别为不同类型的振动数据

一组数据波动的范围很大，不同组数据的动态分别范围也很大。有的组的数据波动范围很小在[-0.2-,0.2]之间如图 1(a)，有的波动很大如图 1(b)，这可能是由于不同数据的不同特征造成的，所以我们继续下一步的分析。

接下来我们对我们检验一下数据是否的缺失值。经检验，所有数据中不存在缺失数据（None），但是存在很多 0 数据。于是我们画出数据统计直方图，如图 2 所示。从数据统计直方图中我们可以看出数据基本符合正态分布特征，所以很多 0 数据是正常数据，故不需要处理。

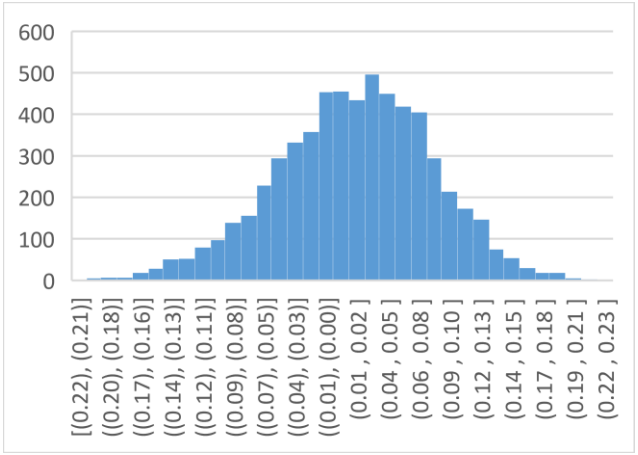


图 2 统计数据直方图

然后画出同一组数据的箱型图进行分析，从数据的箱型图图 3 发现数据存在部分离群点，统计得到这一组数据的上限以上离群点有 10 个，下限以下的离群点有 36 个，占数据总数的 0.7%；但是取出另外的数据如图 4 所示，它上限以上的离群点有 412 个，下限以下的离群点有 405 数据总数的 13.6%；综合统计所有数据的离群点个数占数据总数的 4.9%，在 5%以下，所以不对离群点进行处理。

经过以上分析基本可以看出本次竞赛给定数据质量较高，有利于我们进行神经网络的学习与建模。

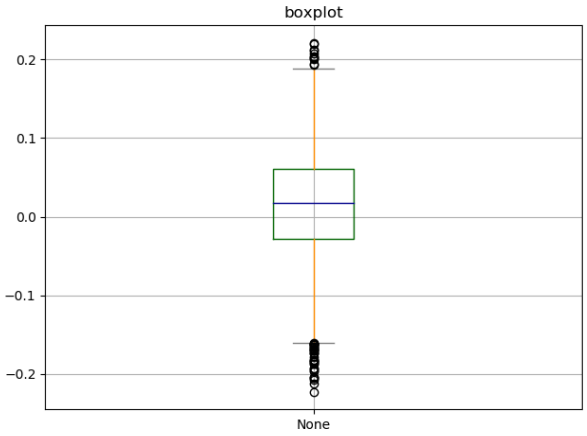


图 3 数据统计箱型图 1，离群点占比 0.7%

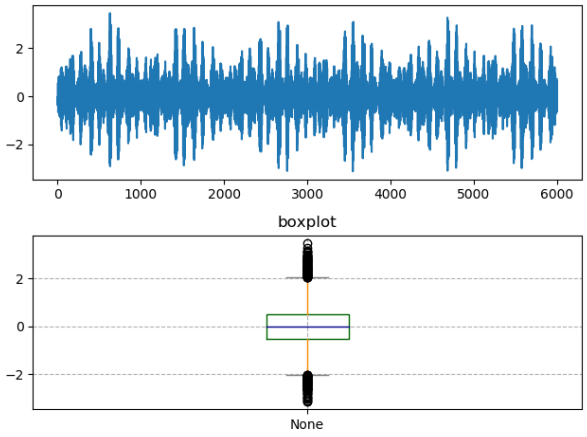


图 4 数据统计箱型图 2，离群点占比 13.6%

由于数据是振动信号，所以我们继续使用信号分析的方法进一步探索数据。首先假设采样频率为 1Hz，对数据进行傅里叶变化得到如下图，从图中可以看出每一组数据的特征频率非常明显，但是同时也包含了很多噪声频率如图 5 所示。

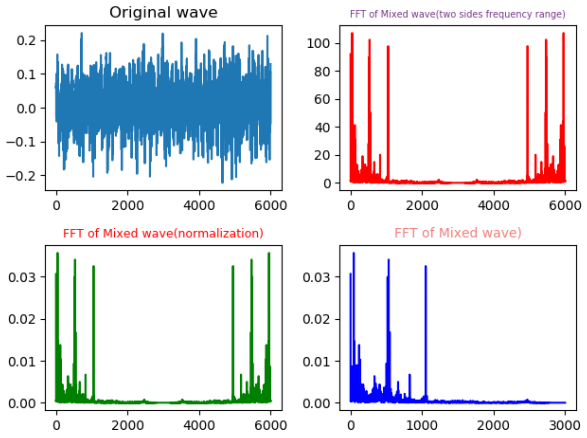


图 5 数据傅里叶变化图

同样的，我们采用信号分析的方法，首先使用小波变换对杂波进行滤波处理，使用 db8 小波，阈值 0.04 对噪声进行过滤，结果如图 6 所示。从处理之后的频谱图看到，部分高频信号被滤除了，但是低平零碎的信号没有过滤干净，所以下一步我打算直接对信号进行带通滤波。

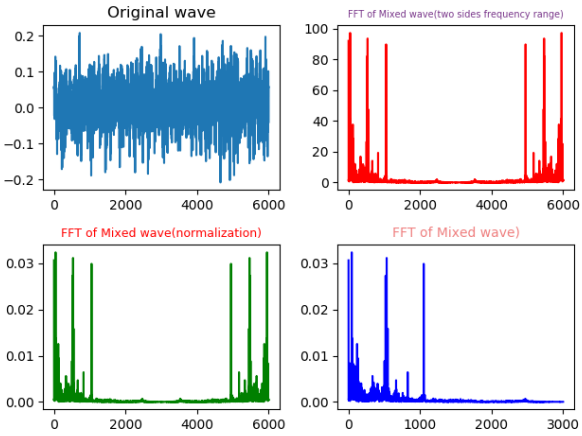


图 6 小波变换后数据频谱图

使用傅里叶变换，对低于 20%以下的频谱分量直接置零，然后重构得到滤波后的数据，结果如图 7 所示。数据就变得比较干净，然后我们将处理后的数据可以直接输入神经网络进行学习了。

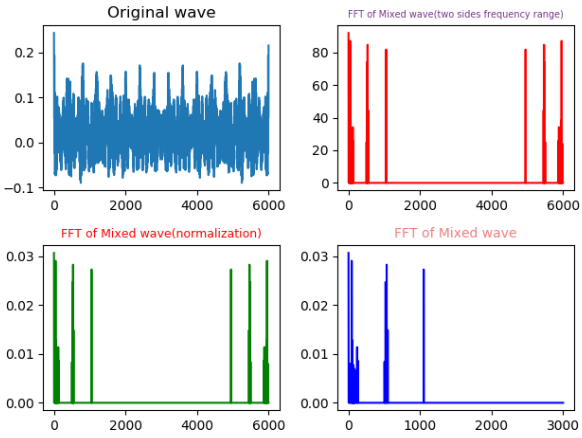


图 7 带通滤波后数据频谱图

但是由于我们的滤波可能会滤去了信号的关键信息，神经网络可以自己挖掘数据特征，所以我们实验的时候同时对处理后和处理前的数据都输入神经网络进行训练。最终检测他们训练模型的准确性。

2. 构建模型

我们建立神经网络模型使用的是基于 python 和 tensorflow 的 keras 库，这个库自己内置了许多预训练的模型，方便我们调用，同时也可以通过很简单的操作自己搭建自己的神经网络模型。我们没有使用预训练的模型，只是自己使用一维卷积搭建了一个神经网络模型。主要调用了以下三个库，分别是网络层，模型和训练优化器的几个库。

```
from keras.layers import *
from keras.models import *
from keras.optimizers import *
```

图 8 实验用的的主要库函数

由于数据量巨大，所以我们先写了一个生成器，分批将数据输入神经网络，基于设备性能考虑，我的 batch_size 设置为 40 个，将原始数据集分 640 组数据用于训练，剩下 154 组数据用于测试。生成器模型如下图 9 所示。

```python
def xs_gen(path=MANIFEST_DIR,batch_size = Batch_size,train=True,Lens=Lens):

    img_list = pd.read_csv(path,header=None)
    img_list = np.array(img_list)[:Lens]
    print("Found %s train items."%len(img_list))
    print("list 1 is",img_list[0,-1])
    steps = math.ceil(len(img_list) / batch_size)    # 确定每轮有多少个batch
    while True:
        for i in range(steps):

            batch_list = img_list[i * batch_size : i * batch_size + batch_size]
            np.random.shuffle(batch_list)
            batch_x = 1000*np.array([file for file in batch_list[:,:-1]])
            batch_y = np.array([convert2oneHot(label,10) for label in batch_list[:,-1]])

            yield batch_x, batch_y
```

图 9 生成器函数

```python
def build_model(input_shape=(TIME_PERIODS,),num_classes=10):
    model = Sequential()
    model.add(Reshape((TIME_PERIODS, 1), input_shape=input_shape))
    model.add(Conv1D(16, 8,strides=2, activation='relu',input_shape=(TIME_PERIODS,1)))
    model.add(Conv1D(16, 8,strides=2, activation='relu',padding="same"))
    model.add(MaxPooling1D(2))
    model.add(Conv1D(64, 4,strides=2, activation='relu',padding="same"))
    model.add(Conv1D(64, 4,strides=2, activation='relu',padding="same"))
    model.add(MaxPooling1D(2))
    model.add(Conv1D(256, 4,strides=2, activation='relu',padding="same"))
    model.add(Conv1D(256, 4,strides=2, activation='relu',padding="same"))
    model.add(MaxPooling1D(2))
    model.add(Conv1D(512, 2,strides=1, activation='relu',padding="same"))
    model.add(Conv1D(512, 2,strides=1, activation='relu',padding="same"))
    model.add(MaxPooling1D(2))
    model.add(GlobalAveragePooling1D())
    model.add(Dropout(0.3))
    model.add(Dense(num_classes, activation='softmax'))
    return(model)
```

图 10 贯序神经网络模型结构

接着我们使用使用 keras 自带的 sequential 模型建立一个简单的 14 层的神经网络，激活函数使用 Relu 函数每层做两次一维卷积，再做一次 maxpooling 池化，最后在做一次全局均值池化，然后做一层是全连接层，为防止模型陷入局部最优，我们的 dropout 参数设置为 0.3，模型生成的参数如图 11 所示。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 6000, 1)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 2997, 16)          144
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1499, 16)          2064
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 749, 16)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 375, 64)           4160
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 188, 64)           16448
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 94, 64)            0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 47, 256)           65792
_________________________________________________________________
```

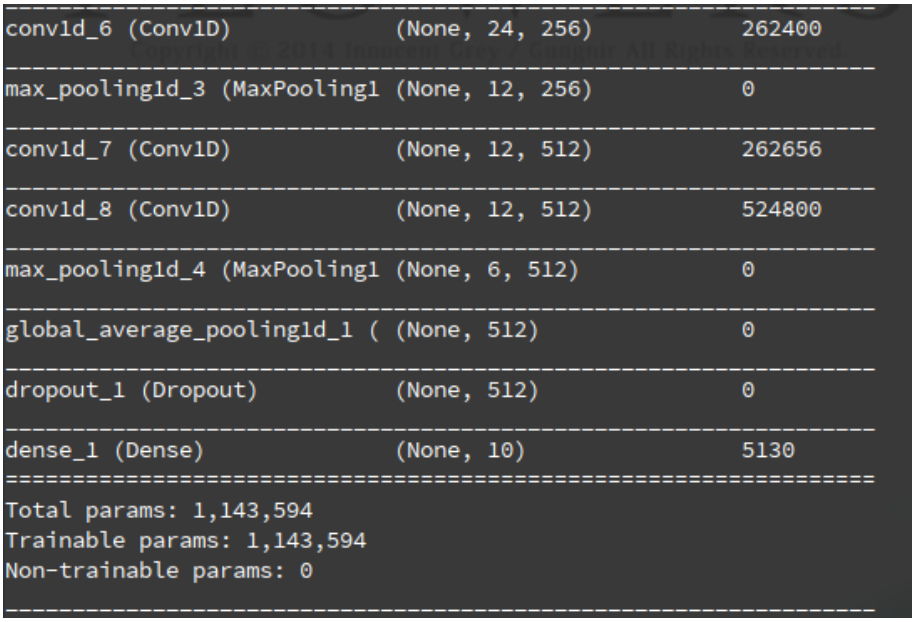
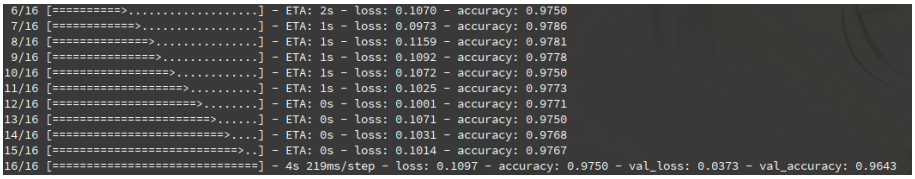


图 11 神经网络参数

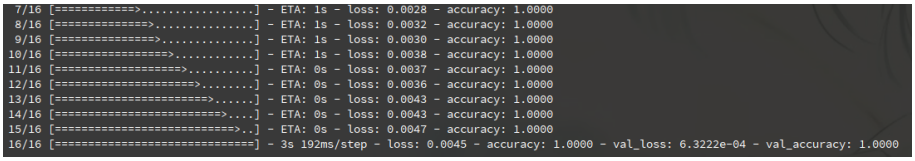
设置好以上参数之后开始训练，分别将经过预处理的模型和未经预处理的模型送入网络训练。

3. 结果分析

神经网络训练的结果如图 12(a)和(b)所示。



(a)



(b)

图 12 神经网络训练结果，(a)为经过数据预处理的数据训练结果，(b)为原始数据训练结果

从图 12(b)看到未处理的数据在训练集和测试集上的准确度达到了 100%，但是处理后的数据只有 97%左右。造成这种情况的原因一方面有可能是处理后的数据损失了数据特征信息，另一方面可能是由于网络参数的调整不匹配

图 13 给出了训练过程 loss 的下降曲线，可以看出由于训练数据量较大，所以训练的 loss 下降平稳但是测试数据由于测试集样本较少会产生波动。Loss 在大约 20 次迭代的时候基本收敛，说明我们的模型参数设置比较好，而且最终也并未出现过拟合的情况。

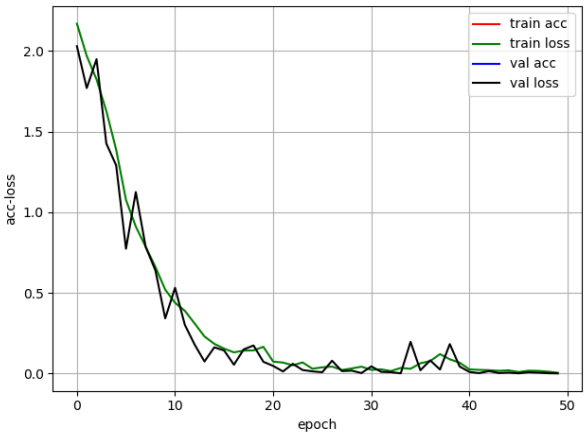
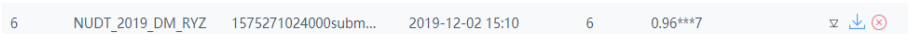
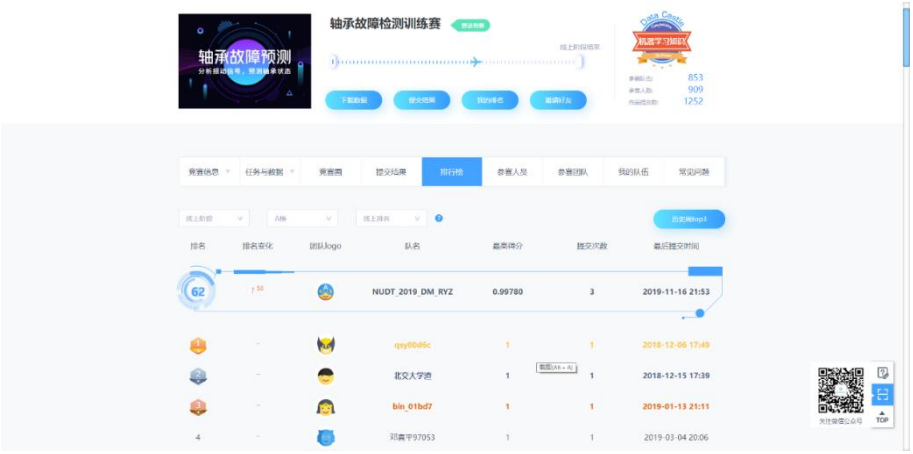


图 13 loss 下降曲线



(a)



(b)

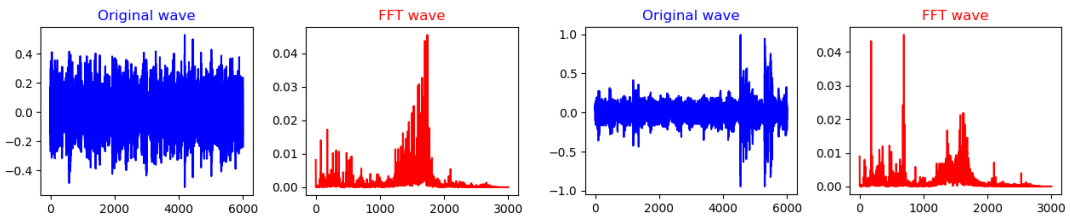
图 14 竞赛成绩，(a)是数据预处理后训练的模型，(b)是原始数据训练的模型

我分别将预处理训练的模型和未预处理训练的模型用于竞赛成绩提交检验，结果如图 14 所示，最终准确度最高的是原始数据直接训练的模型为 0.997，排名 62 名。

四、 总结

1. 问题总结 1----为什么数据预处理不管用？

对于为什么数据经过预处理没有原始数据训练的模型更优，询问老师之后发现了原因，因为傅里叶变换本质就是做卷积运算提取特征，而在神经网络中设置了卷积层，作用与傅里叶变换是一样的，并且多次提取特征进行学习了，所以不需要再做傅里叶变换对数据进行操作了，而经过小波滤波的操作在一定程度上可以减小数据量大小，加速神经网络训练，而傅里叶变换阈值滤波很大程度上使得数据特征损失了，所以使得精度降低了。



(0) (1)

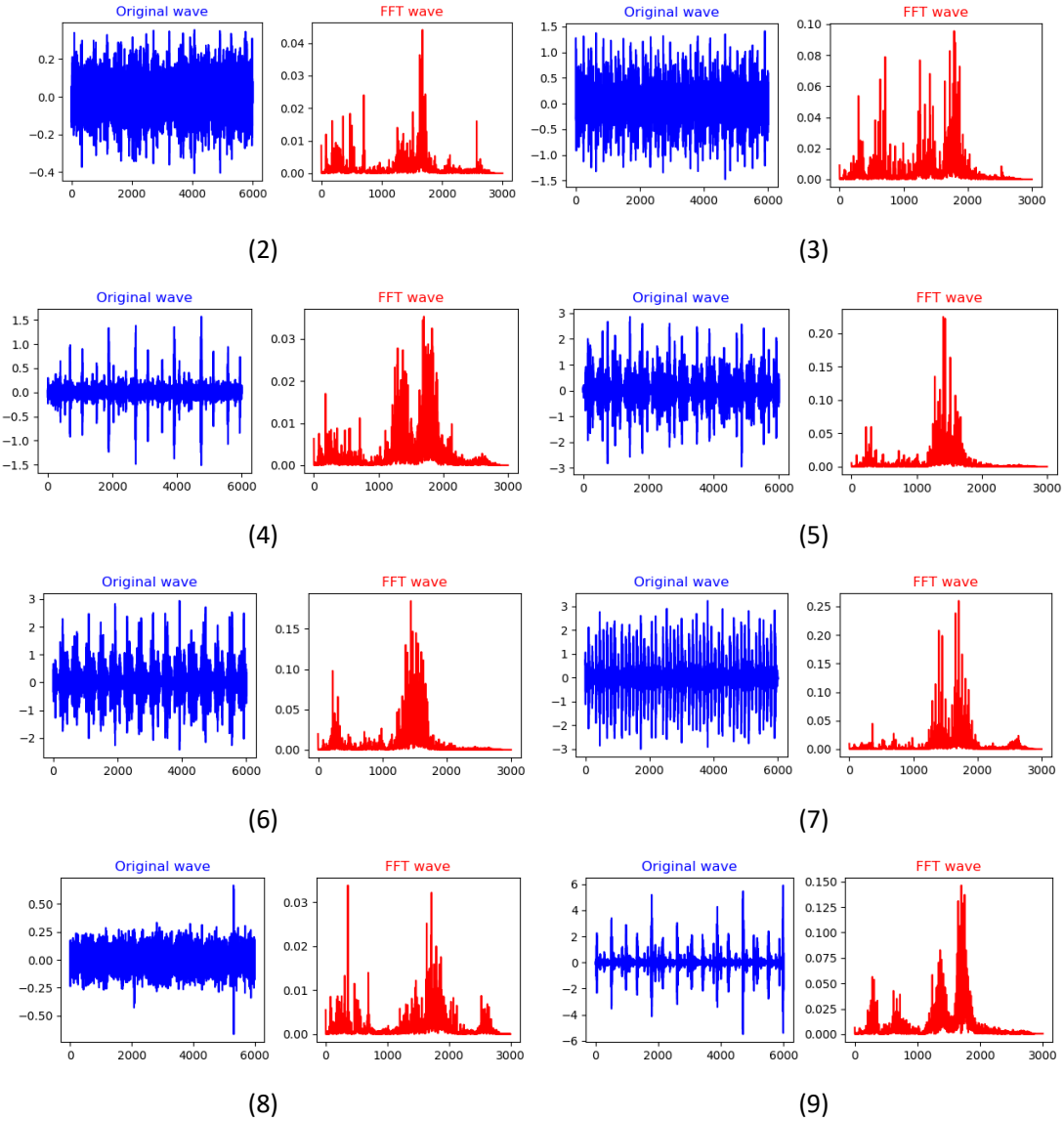


图 15 各类故障及其对应的频谱图，(1)~(9)为 1~9 种故障(0)为正常数据

但是使用傅里叶变换之后我们可以看出更多数据特征，比如图 15 我们展示了 1 到 9 种不同的故障情况以及正常轴承振动的频谱图，从图中我们基本可以看出不同情况对应的频谱特征有关联，可以进行下一步的关联分析。虽然关联分析对数据分类的精确度不及神经网络的精确度，但是关联分析可以使我们学习的数据具有可解释性，从而引导故障检测者进一步分析问题。

同时从这些数据中我们也可以发现，经过傅里叶滤波后的数据损失了大部分信息，这也是为什么数据处理之后训练准确度降低。

2. 问题总结 2----对数据的进一步分析

我又重新对数据进行了分析，发现给定的样本类不平衡，如图 16 所示，0，7，9 类样本数目明显多于其他样本，而我们划分训练集的时候采用的随机划分的策略， 可能会导致训练样本和测试样本都存在类不平衡的问题，解决问题的方法可以手动重新划分数据集，删去一部分数据集，使其平衡。

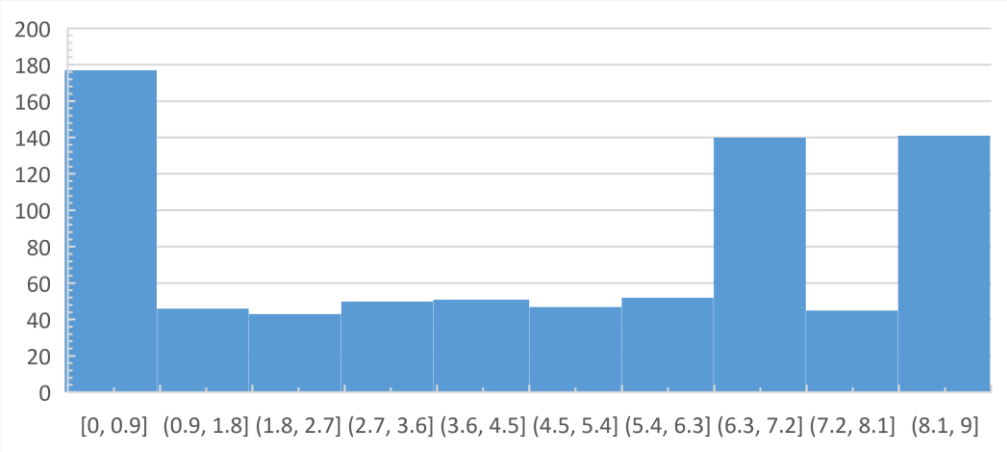


图 16 统计各类样本个数

在继续分析数据的途中，我发现我对数据没有进行一致性检验，和一致性处理。从图 17 上可以看出，即使是同 一组数据，其特征频率也可能不尽相同，所以在输入神经网络之前应该对数据进行一致性处理。

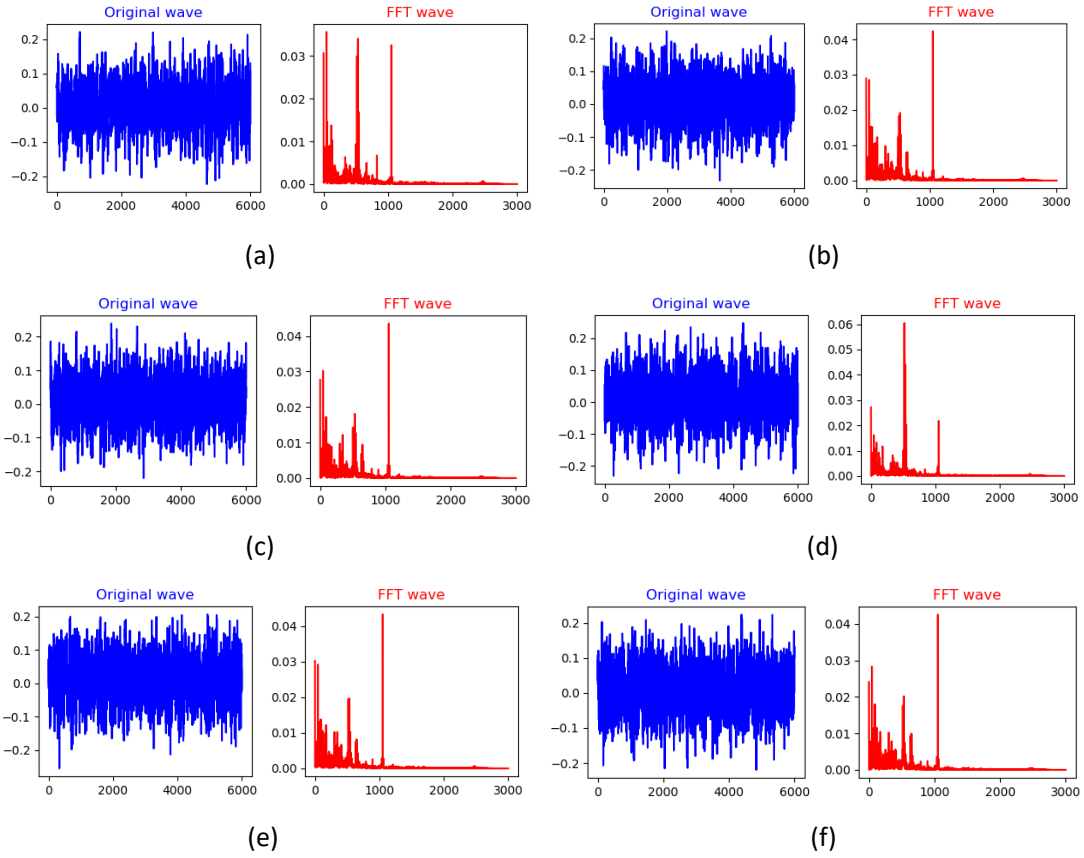


图 17 同一类数据一致性分析

3. 个人小结

我本科学习的专业是电气相关的，做过信号分析，而这次比赛的项目也是我第一次使用神经网络进行数据处理，也是我第一次独立完成了一个完整的神经网络的项目，做了数据的预处理，提取标签转换成 onehot，构造生成器，最后建立模型，使我学到了许多。

得益于 keras 的集成性，我能很快使用 keras 搭建一个神经网络模型，同时 keras 也包含许多内置的预训练模型，所以我自己也能尝试写一个简单的神经网络模型。

虽然我数据的预处理没有做好，但是训练的准确度依旧高的，我觉得很大原因是因为这次训练赛的数据质量较高，而且模型参数设置合理（我是参照网上其他人的帖子设置的参数），最后训练速度也很快。

接下来我准备用 keras 做一下图像数据的分类实验，在公开的 caltech256 数据集上进行实验来进一步学习这个工具，同时也应该对数据工程上多下功夫考虑。