参赛编号:YRDMCM202200945
选题: __B__ (A 或 B 或 C)　参赛赛道: ___本科生__（本科生、专科生或研究生）

2022 年第二届长三角高校数学建模竞赛

基于一维卷积神经网络的齿轮箱故障诊断

摘　　要：

本文研究了齿轮箱的故障诊断模型，根据附件 1 提供的数据进行了正常状态和故障状态特征值的刻画，同时建立了齿轮箱的分类模型。通过研究该问题可以对齿轮箱振动信号得到统计学规律，通过分类模型判断未知测试数据的故障类型，可以有效提高故障判断的效率和精度。

针对问题一，首先将探测的数据使用**时域平均法**进行降噪处理，提高信号数据的信噪比。之后使用 matlab 对振动信号进行了时域、频域分析，获得了**倒频谱**，**功率谱**等图像。我们选取了相对重要的 12 个信号指标进行对正常状态和 4 种故障状态的特征进行分析，获得特征数据后，使用 SPSSPRO 提供的**因子分析法**对特征进行选择，以降低数据维度，获取其权重。最终获取了刻画不同故障类型的最重要五个特征，并分别计算出各自的权重。

针对问题二，我们建立了**一维卷积神经网络模型**，将附件 1 的数据分为 145 组，109 组为训练集，36 组为测试集。通过对数据进行时域训练和频域训练来获取检测模型，激活函数采用 tanh，其更适用于处理加速度的振动信号，准确率较高。对于判断模型的评价，使用**二分类混淆矩阵**来进行效果评价，通过精确率，召回率，分类准确率，F 度量，损失函数等指标从不同角度来评价分类结果的优劣。得到模型的正确率为 0.94，较为理想，可以很好的判断是否处于故障状态。

针对问题三，要求在问题二的基础上进一步对处于故障状态的齿轮箱建立故障诊断模型。我们对第二问的神经网络模型进行改进，对于模型性能的评价，我们建立了**多分类混淆矩阵**，对相关参数进行了求解分析，得到该模型判断正常状态和 4 中故障状态的正确率分别为 0.92，0.85，0.87，0.87。

针对问题四，我们对故障诊断模型进行了具体运用，将附件 2 提供的测试数据进行了检测和诊断分析，得到诊断结果表。

最后，本文对模型进行了优、缺点评价，在对缺点提出相应的改进方式的基础上，将该模型推广到其他领域。

关键词：时域平均、傅里叶变换、因子分析法、神经网络、混淆矩阵

目录

一、问题重述

齿轮箱是用于增加输出扭矩或改变电机速度的机械装置，被广泛应用于如汽车、输送机、风机等机械设备中。它由两个或多个齿轮组成，其中一个齿轮由电机驱动。电机的轴连接到齿轮箱的一端，并通过齿轮箱的齿轮内部构件，提供由齿轮比确定的输出扭矩和速度。典型的齿轮箱剖面如图 1 所示。在齿轮箱的运行过程中，可以通过加装加速度传感器采集振动信号来判断齿轮箱是否出现异常。本题旨在通过建立相关数学模型对齿轮箱采集到的振动信号进行分析。

在本题中，我们通过安装在齿轮箱不同部位的四个加速度传感器，采集了 5 种状态下齿轮箱的振动信号，具体数据见附件 1。其中表单 gearbox00 为齿轮箱正常工况下采集到的振动信号;表单 gearbox10 为故障状态 1 下采集到的振动信号;表单 gearbox20 为故障状态 2 下采集到的故障信号;表单 gearbox30 为故障状态 3 下采集到的故障信号;表单 gearbox40 为故障状态 4 下采集到的振动信号。信号的采样频率为 6.4kHz。请利用这些数据，建立数学模型解决以下问题:

1、对齿轮箱各个状态下的振动数据进行分析，研究正常和不同故障状态下振动数据的变化规律及差异，并给出刻画这些差异的关键特征。

2、建立齿轮箱的故障检测模型，对其是否处于故障状态进行检测，并对模型的性能进行评价。

3、建立齿轮箱的故障诊断模型，对其处于何种故障状态进行判断，并对模型的性能进行评价。

4、结合所建立的故障检测和诊断模型对附件 2 中另行采集的 12 组测试数据进行检测和诊断分析，将分析结果填写到下表中（注:测试数据中可能存在除以上 4 种故障之外的故障状态，若存在，则将对应的诊断结果标记为:其它故障)，并将此表格放到论文的正文中。

二、问题分析

2.1 问题一的分析

对于问题一，我们需要对题目所给的附件一中的时域信号进行分析，获取其关键特征值，以及不同故障与正确情况下的差异性。在进行分析之前，我们需要对其进行预处理，来消除噪声干扰。除了对时域信号进行分析外，我们还需要对其进行变化，得到频率谱，功率谱，倒频谱等，获取其特征值，同样进行对比和分析。当把不同特征提取出来后，便进行特征选择，刻画正常状态与故障态之间的差异。

2.2 问题二的分析

问题二要求建立判断齿轮是否处于故障状态的模型，并且进行性能评价。首先可以分析附件一已知的数据进行分析，得到齿轮振动信号特征，并利用一维神经网络建立深度学习模型，把正常状态数据和发生故障数据作为数据集来训练，再将附件一的数据带入求解，来判断模型的准确度。之后通过评价神经网络常用的方法来计算相关参数，评价模型

2.3 问题三的分析

问题三要求在问题二的基础上，进一步判断齿轮箱的状态，可以采用上一问深度学习模型的基础，考虑题中 4 种故障类型和其他故障类型，将附件 1 中不同状态数据作为数据集来训练，进行模型的改进与优化，之后建立多分类混淆矩阵来进行对模型的评价和分析。

2.4 问题四的分析

利用第二问、第三问所建立的诊断模型，带入附件 2 的数据进行拟合与运行，得到诊断结果表。可以对最终结果进行一定的分析和修正，提高模型的分析正确率。

三、模型假设

1、假设在前三问中，齿轮箱只存在四种故障情况，其他故障情况不予考虑。

2、假设所给数据处于一定的噪声环境中，且可以通过某种方法来减小噪声干扰。

3、假设数据集按照时间序列逐步测试，不存在数据的缺失。

4、不同故障之间存在的区别特征不同，不存在同时处于两种故障的数据集。

5、故障不会出现相互转换，即一个数据集的所有数据全都用于刻画一种故障。

四、符号说明

符号	说明
Fn	取样频率
N	样本周期数目
M	一个周期中的采样数目
p(x)	信号样本的概率密度函数
$F(\omega)$	信号样本的频率函数

i	传感器的编号
j	不同故障类型
Q	因子分析法的化简函数
F	不同故障的权重表达函数
Pi	P0 为正常概率，P1 为发生故障的概率
$Pmax$	发生某种故障最大的概率

五、模型的建立与求解

5.1 问题一

5.1.1 信号预处理

回转和往复机械在运行过程中,反映其运行状态的各种信号是随机器运转而周期性重复的,其频率是机器回转频率的整倍数。[1]但这些信号又往往被伴随产生的噪声干扰,在噪声较强时,不但信号的时间历程显示不出规律性,而且由于常用的谱分析不能略去任何输入分量,在频谱图中这些周期分量很可能被淹没在噪声背景中。因此我们引用了时域平均法来去除噪声，提取感兴趣的周期分量。

时域平均法是从混有噪声的复杂周期信号中提取感兴趣周期分量的常用方法，可以消除与给定频率无关的信号分量，包括噪声和无关的周期信号，提取与给定频率有关的周期信号，因此能在噪声环境下工作，提高分析信号信噪比。

时域平均的基本原理如图 1 所示，经过滤波后的原始信号,用适当的周期 T 截取 N 段,进行时域信号平均后输出。周期 T 可根据实际结构的运行规律定出，也可对信号求其自相关系数原理定出。

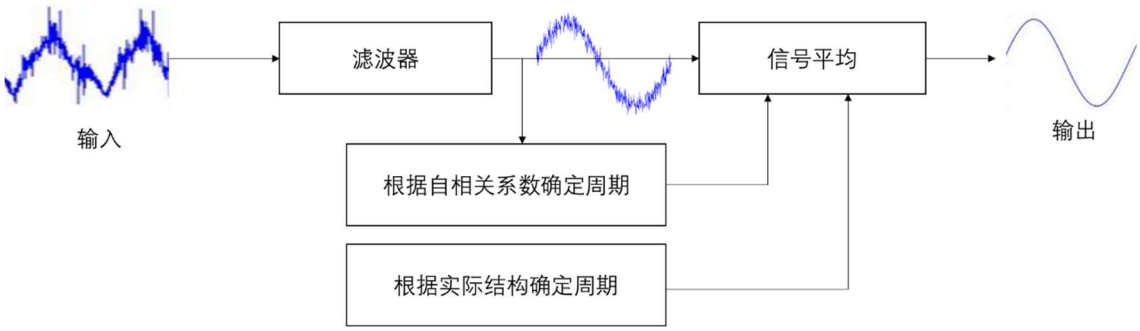


图 1：时域平均的原理

采用时域平均法提取周期分量 $\mathrm{f(t)}$, 也可看作是梳状滤波器进行滤波的过程。以 Δt 为时间间隔对 $\mathrm{x(t)}$ 进行离散采样, 得到离散值 $\mathrm{x(n\Delta t), n = 0,1,2,\cdots\cdots}$ 。设 N 为叠加、平均的周期段数目, M 为一个周期中的采样数目, Δt 为采样间隔。 $\mathrm{x(n\Delta t)}$ 为滤波器的输入, 则滤波器的输出 $\mathrm{y(n\Delta t)}$ 为

$$y(n\Delta t)=\frac{1}{N}\sum_{r=0}^{N-1}[x(n-rM)\Delta t] \tag{1}$$

对（4）式进行 z 变换，设 $y(n\Delta t), x(n\Delta t)$ 的 z 变换为 Y(z)，有

$$Y(z)=\frac{1}{N}\sum_{p=0}^{N-1}X(z)z^{-m}=\frac{X(z)}{N}\cdot\frac{1-z^{-MN}}{1-z^{-M}} \tag{2}$$

滤波器的传递函数 H(z) 为

$$H(z)=\frac{Y(z)}{X(z)}=\frac{1}{N}\cdot\frac{1-z^{-M}}{1-z^{-M}} \tag{3}$$

通过时域分析法处理信号后，我们发现其时域波形曲线更加光滑

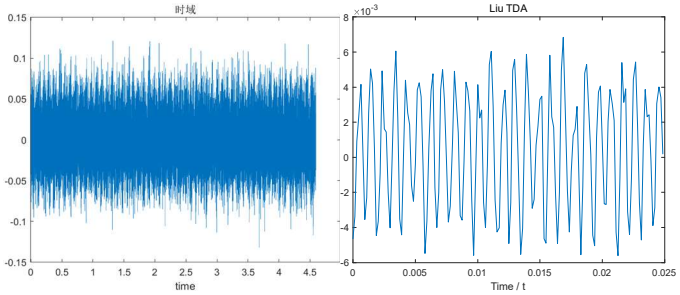


图 2：时域分析法处理前后对比

此外，由于传感器 4 的数据集中有明显的杂乱数据，我们将将其进行了剔除处理，以便于后续特征值的提取，以及模型的建立。

5.1.2 特征提取

在获得处理之后的数据后，我们希望对感兴趣的特征进行提取，来比较正常状态下和故障状态下的不同点和相似点。常用的特征提取方法，包括时域特征提取，频域特征提取。[2]

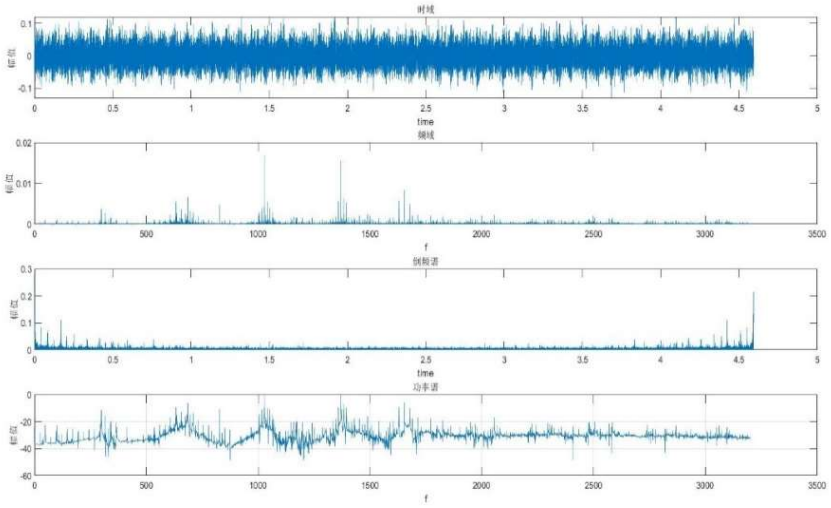


图 3：时域谱、频域谱、倒频谱、功率谱的求解

将信号的特征谱表示出来后，发现虽然可以看出故障和正常状态确实存在差异，但无法确切的将其量化表示，因此希望可以确定信号的一些参数，来进行量化特征对比。

时域特征提取通常包括的参数较多，比如有 RMS（有效值）、峰峰值、峭度、裕度、歪度、均值、均方根、脉冲因数、波形因数、波峰因数等等。

确定性时域信号分析的基本参数有峰值、均值、均方根值（有效值）、方差等; 随机性时域信号除上述四个基本参数外还有方根幅值、平均幅值、偏度、峭度等, 对于信号 x(t), 这些参数的计算式为:

$$\text{峰值: } X_p = \max|x(t)|$$

$$\text{平均值: } \mu_x = \frac{1}{T}\int_0^T x(t)dt$$

$$\text{均方根值: } x_{rms} = \sqrt{\frac{1}{T}\int_0^T x^2(t)dt}$$

$$\text{方根幅值 } x_r = \left(\frac{1}{T}\int_0^T |x(t)|^{0.5}dt\right)^2$$

$$\text{标准偏差: } \sigma_x = \sqrt{\frac{1}{T}\int_0^T (x(t)-\mu_x)^2 dt}$$

$$\text{偏斜度指标（偏度） } \alpha_3 = \int_{-\infty}^{+\infty} x^3 p(x)dx$$

$$\text{峭度指标（峭度） } \alpha_4 = \int_{-\infty}^{+\infty} x^4 p(x)dx$$

式中, T 为采样时间, $p(x)$ 为 $x(t)$ 的概率密度函数;

在这些指标中, 均值表示了信号的常值分量; 均方根植描述了信号的强度; 方 差描述了信号的波动分量。在实际应用中, 均方根值作为故障诊断的判断依据是最简单、最常用的一种方法, 多适用作稳态振动的情况。

当时间信号中包含的信息不是来自一个零件或部件, 而是属于多个元件时, 例 如在多级齿轮的振动信号中往往包含有来自高速齿轮、低速齿轮以及轴承等部件的 信息, 在这种情况下, 可利用下列的一些无量纲示性指标进行故障诊断或趋势分析。

$$\text{波形因数: } K = \frac{x_{rms}}{X_p}$$

$$\text{脉冲因数: } I = \frac{\mu_x}{X_p}$$

$$\text{峰值因数: } C = \frac{\mu_x}{x_{rms}}$$

$$\text{裕度因数: } L = \frac{u_x}{x_r}$$

峰值、峭度和峰值因数在一定程度上反映出振动信号是否含有冲击成分。一般说来，随着故障的发生和发展，均方根值、方根幅值、平均值以及峭度均会增大，其中峭度对大幅值非常敏感，当其概率增加时，峭度值迅速号中含有脉冲的故障。峭度、裕度因数和脉冲因数对于冲击脉冲类的故障比较敏感，特别是当故

障早期，其稳定性较差。而均方根值对故障不敏感。所以为取得较好的效果，常将它们同时使用，以兼顾敏感性和稳定性。

信号的频谱分析是工程中最普遍应用的方法, 频谱分析是计算采集数据的幅、相频特性曲线, 基本出发点是数字信号的快速傅立叶变换（FFT)。[3]

非周期连续时间信号 x(t) 的傅立叶变换得到连续频谱 $X(f)$, 其积分关系式为:

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi f}\, dt \qquad (4)$$

重心频率能够描述信号在频谱中分量较大的信号成分的频率，反映信号功率谱的分布情况。换句话说，对于给定的频带范围，低于重心频率的频率范围内包含的能量是信号总能量的一半。重心频率公式为：

$$FC = \frac{\int_0^{+\infty} fP(f)df}{\int_0^{+\infty} P(f)df} \qquad (5)$$

与重心频率不同，均方频率是信号频率平方的加权平均，同样以功率谱的幅值为权。它有一个辅助理解的物理含义：如果把功率谱曲线围起来的区域看作一块木板，那么当这块木板围绕 y 轴旋转时，均方根频率所在位置到原点的距离就是旋转体的惯性半径（指物体质量假设的集中点到转动轴间的距离），其公式为：

$$MSF = \frac{\int_0^{+\infty} f^2P(f)df}{\int_0^{+\infty} P(f)df} \qquad (6)$$

频率方差是重心频率为中心的惯性半径的平方。若重心附近的频谱幅值较大，则频率方差较小；若重心附近的频谱较小，则频率方差较大。 频率方差描述功率谱能量分布的分散程度。

$$VF = \frac{\int_0^{+\infty} (f - FC)^2P(f)df}{\int_0^{+\infty} P(f)df} \qquad (7)$$

式中 $P(f)$ 为信号的功率谱。

功率谱熵刻画了被分析信号的谱形结构情况，当信号的频率组成比较简单、谱线较少时，其对应的组分概率越大，计算得到的功率谱熵越小，表示信号的不确定性和复杂性越小;反之，若信号能量在整个谱形结构上分布的越均匀，则功率谱熵越大，信号的复杂性和不确定性越大。因此，功率谱熵体现了信号频谱的不确定性及复杂程度。其公式为：

$$I(f) = \int_{-\pi}^{\pi} \log f(\omega)d\omega \qquad (8)$$

5.1.3 特征的选择

在确定信号在各谱系中所需要提取的特征后，便需要进行特征的选取和降维。特征选取指的是从已有的 M 个特征(Feature)中选择 N 个特征使得系统的特定指标最优化，是从原始特征中选择出一些最有效特征以降低数据集维度的过程。

特征选择的目的是提升模型输入与建模目标的相关性并降低冗余度，避免“维度灾难”，同时为后续数据处理提供更好的理解。特征选择还有助于减少传感器的安装数量，比如，当评估轴承的健康状态时，若振动特征更能够体现轴承状态，可以只增加振动传感器，而不选择温度或其他类型的传感器。此外，通过特征选择还可以提高算法的计算效率。

为了降低数据的维度，我们选择因子分析法进行特征值的处理。

因子分析是一种数据简化的技术。它通过研究众多变量之间的内部依赖关系，探求观测数据中的基本结构，并用少数几个假想变量来表示其基本的数据结构。这几个假想变量能够反映原来众多变量的主要信息。原始的变量是可观测的显在变量，而假想变量是不可观测的潜在变量，称为因子。

设 $X_i(i=1,2,\cdots,p)$ 个变量, 如果表示为

$$X_i=\mu_i+a_{i1}F_1+\cdots+a_{im}F_m+\varepsilon_i,(m\le p) \tag{9}$$

或

$$\begin{bmatrix}X_1\\X_2\\\vdots\\X_p\end{bmatrix}=\begin{bmatrix}\mu_1\\\mu_2\\\vdots\\\mu_p\end{bmatrix}+\begin{bmatrix}a_{11}&a_{12}&\cdots&a_{1m}\\a_{21}&a_{22}&\cdots&a_{2m}\\\vdots&\vdots&&\vdots\\a_{p1}&a_{p2}&\cdots&a_{pm}\end{bmatrix}\begin{bmatrix}F_1\\F_2\\\vdots\\F_m\end{bmatrix}+\begin{bmatrix}\varepsilon_1\\\varepsilon_2\\\vdots\\\varepsilon_p\end{bmatrix} \tag{10}$$

$$X-\mu=AF+\varepsilon$$

其中

$$X=\begin{bmatrix}X_1\\X_2\\\vdots\\X_p\end{bmatrix},\mu=\begin{bmatrix}\mu_1\\\mu_2\\\vdots\\\mu_p\end{bmatrix},A=\begin{bmatrix}a_{11}&a_{12}&\cdots&a_{1m}\\a_{21}&a_{22}&\cdots&a_{2m}\\\vdots&\vdots&&\vdots\\a_{p1}&a_{p2}&\cdots&a_{pm}\end{bmatrix},\varepsilon=\begin{bmatrix}\varepsilon_1\\\varepsilon_2\\\vdots\\\varepsilon_p\end{bmatrix} \tag{11}$$

得到因子参数后，需要对其进行因子旋转

因子分析的目的不仅仅是要找出公共因子以及对变量进行分组，更重要的是要知道每个公共因子的意义，以便进行进一步的分析。如果每个公共因子的含义不清，则不便于进行实际背景的解释。

初始因子的综合性太强，难以找出因子的实际意义。由于因子载荷阵是不唯一的，所以可以对因子载荷阵进行旋转，使因子载荷阵的结构简化，使其每列或行的元素平方值向 0 和 1 两极分化。

设 Γ 正交矩阵，做正交变换 $B = A\Gamma$ 。

这里我们使用的方法是四次方最大旋转。我们是从简化载荷矩阵的行出发，通过旋转初始因子，使每个变量只在一个因子上有较高的载荷，而在其它的因子上尽可能低的载荷。 如果每个变量只在一个因子上有非零的载荷，这时的因子解释是最简单的。四次方最大法通过使因子载荷矩阵中每一行的因子载荷平方的方差达到最大。

简化准则为:

$$Q = \sum_{i=1}^{p} \sum_{j=1}^{m} b_{ij}^{4} = MAX \qquad (12)$$

通过因子分析法，我们可以得到相关特征值的重要程度，从而刻画出每个特征值在判断故障类型时的权重。

5.1.4 模型的求解

	均值	方差	峰峰值	…	均方频率	频率方差	相对功率谱熵
正常	0.0005	0.001	0.253	…	1601.831	3217783	0.578996
				…			
故障 1	0.001	0.0015	0.2849	…	1774.677	3777247	0.536222

表 1：信号的部分特征值

完整特征值表格请参照附件 1

KMO 检验和 Bartlett 的检验		
KMO 值		0.637
Bartlett 球形度检验	近似卡方	835.669
	df	45.000
	p	0.000***

注：***、**、*分别代表 1%、5%、10%的显著性水平

表 2：KMO 检验和 Bartlett 的检验

对变量进行 KMO 检验，得到其 KMO 值为 0.637，此外 P<0.01 或 P<0.05, 呈显著性，通过 Bartlett 检验：可以进行因子分析。

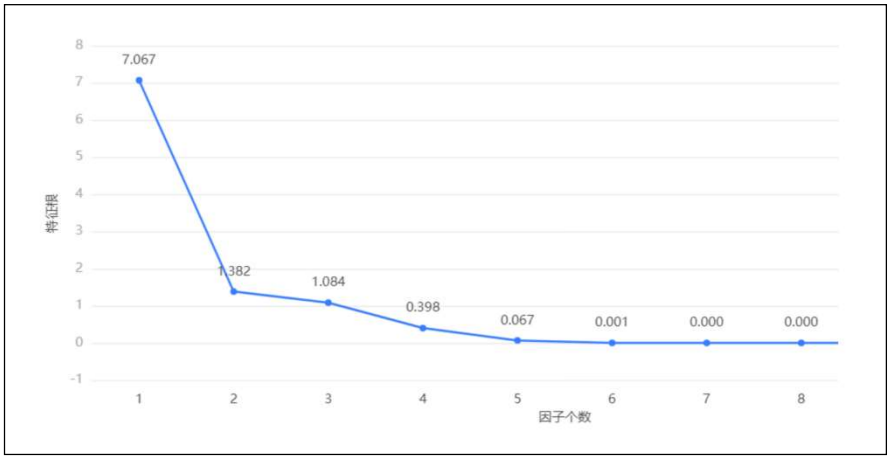


图 4：KMO 检验和 Bartlett 的检验

碎石图是根据各主成分对数据变异的解释程度绘制的图。其作用是根据特征值下降的坡度来确认需要选择的因子主成分个数，结合方差解释表可用于确认或调整因子主成分个数;每一个主成分为一个点，通过“坡度趋于平缓”的未知判断提取主成分的数量。

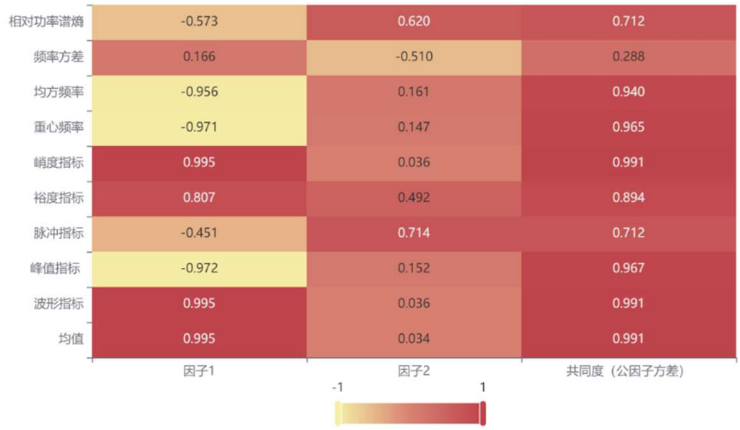


图 5：载荷矩阵热力图

上图为载荷矩阵热力图，可以分析到每个主成分中隐变量的重要性。同时可结合具体业务进行各因子的隐变量分析。结合热力图可以看到，峭度指标、波形指标、均值在因子 1 中作用显著，相对功率谱熵、脉冲指标在因子 2 中显著。说明这些指标都是影响故障类型的重要特征。

成分矩阵表		
名称	成分	
	成分 1	成分 2
均值	0.141	0.024
波形指标	0.141	0.026
峰值指标	-0.137	0.11
脉冲指标	-0.064	0.516
裕度指标	0.114	0.356
峭度指标	0.141	0.026
重心频率	-0.137	0.106
均方频率	-0.135	0.116
频率方差	0.023	-0.369
相对功率谱熵	-0.081	0.448

表 3：成份矩阵表

上表为成份矩阵表，意在说明各个成分的所包含的因子得分系数（主成分载荷），用于计算出成分得分。

由上可以得到主成分公式：

$$F = \left(\frac{0.698}{0.845}\right) \times F1 + \left(\frac{0.147}{0.845}\right) \times F2 \qquad (13)$$

通过以上分析， 我们选取了每种故障前五个重要的特征指标，重新使用因子分析法计算权重，得到如下表格，刻画出正常状态与不同故障状态下的特征差异。

故障状态 1 与正常状态的主要差异如下：

	传感器标号	特征	权重（%）
故障状态 1	传感器 1	均值	45.3
	传感器 1	均方频率	24.3
	传感器 2	频率方差	10.4
	传感器 3	峰值指标	15.4
	传感器 3	脉冲指标	4.6

故障状态 2 与正常状态的主要差异如下

	传感器标号	特征	权重（%）
故障状态 2	传感器 2	峰值指标	10.3
	传感器 3	峰值指标	10.6
	传感器 3	裕度指标	64.7
	传感器 3	峭度指标	5.9
	传感器 4	均方频率	8.5

故障状态 3 与正常状态的主要差异如下

	传感器标号	特征	权重（%）
故障状态 3	传感器 1	相对功率谱熵	13.5
	传感器 2	重心频率	10.5
	传感器 2	频率方差	32.5
	传感器 3	频率方差	33.7
	传感器 4	均方频率	9.8

故障状态 4 与正常状态的主要差异如下

	传感器标号	特征	权重（%）
故障状态 4	传感器 1	均方频率	23.1
	传感器 2	裕度指标	45.4
	传感器 3	脉冲指标	11.6
	传感器 3	峭度指标	7.5
	传感器 4	重心频率	12.4

表 4： 四种故障状态的特征值权重

通过上述表格可以看出，不同故障的特征值有着不同的重要性，其中裕度指标，频率方差等特征值具有较高的权重，因此在后续的模型建立中可重点考虑。

5.2 问题二

5.2.1 深度学习模型简介

我们采用一维卷积神经网络训练的方法，其智能故障诊断算法分为特征提取、特征选择和故障分类 3 个部分特征提取的目的是提取原始振动信号中与故障特性相关的特征用于后续的故障识别，特征选择策略在特征提取后进行，主要是摒弃特征中的不敏感、无用的特征，实现对特征的进一步精雕，使有效特征的个数进一步缩减，达到去伪存真的目的。[4]

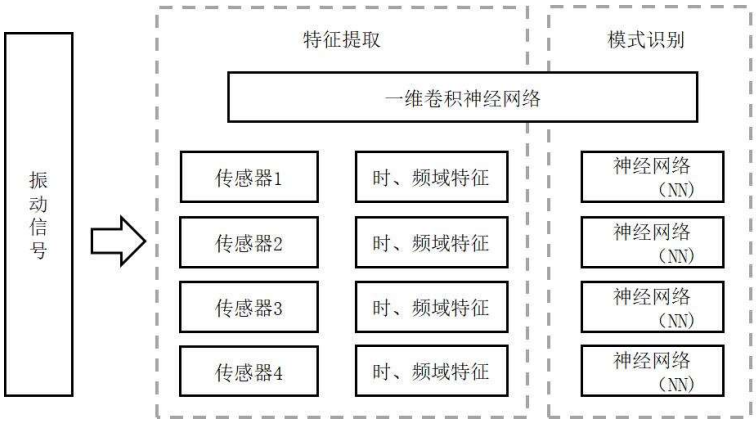


图 6：振动信号处理模型

在模型一中，我们已经提取了振动信号中的时、频域特征，然后利用因子分析法挑选出了敏感、实用的特征。

为了优化模型的运算速度，在有限数据样本内包含更多周期的信号，对原信号进行降采样处理。我们通过分析振动信号可以发现，一个周期数据包含大概 260 采样点。选取样本数据段长度为 1000，可以包含 3 到 4 周期的振动数据。降采样后，每隔 1000 点为起点取一组样本，这样每种故障模式就可以得到 29400/1000≈29 组数据样本，加上正常状态一共可以得到 145 组数据样本，其中选 109 组为训练集，36 组为测试集。

对于神经网络的激活函数，如果使用 ReLU 函数，效果不好，准确率非常低，后来使用激活函数 tanh，其更适用于处理加速度的振动信号，准确率大大提高。

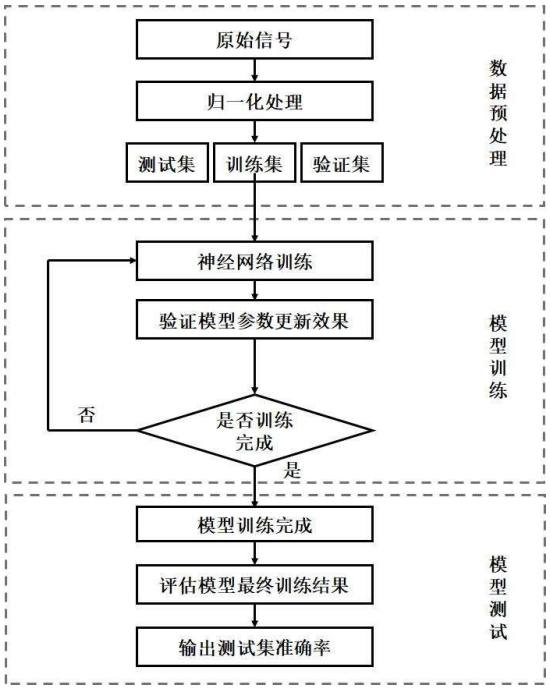


图 7：深度学习流程图

(1) 对随机信号进行标准化处理，使其各个指标处于同一数量级和量纲，从而加快模型学习效率。

(2) 将数据标准化后随机打乱处理并按一定的比率分为训练集、验证集与测试集。
(3) 构建 1D-CNN 神经网络模型，初始化模型参数，方便模型参数更新。
(4) 开始第一轮次训练，当所有批样本训练完后用验证集对模型进行初步评估，并开始下一轮次训练，重复该过程进行直至所有轮次迭代完毕。
(5) 训练完成，使用测试集评估模型最终训练效果。
训练模型经过 25 次迭代之后，其准确率不断升高，达到 60%，损失函数不断下降，说明拟合效果较好

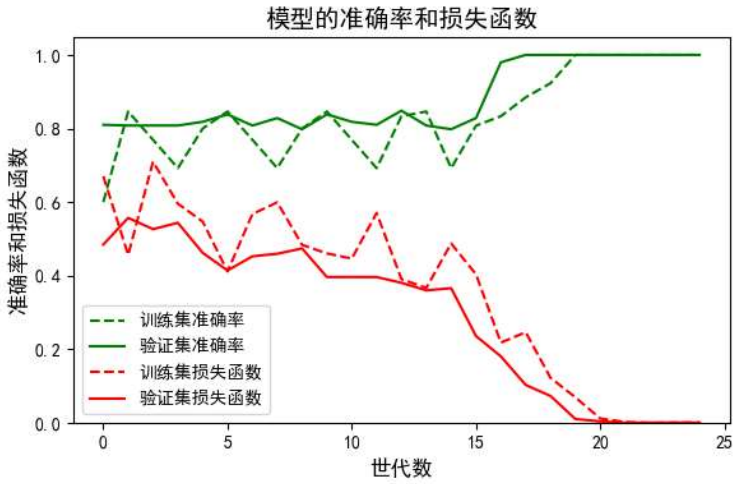


图 8：神经网络训练迭代图

迭代图反映了训练集和验证集的准确度和损失函数随迭代次数的变化，可以明显发现当其迭代到 15 次就可以完全正确区分齿轮箱是否发生故障。

5.2.2 训练模型结果测试

在输入样本时，我们将附件一中故障的四组数据作为齿轮箱故障的数据集，将正常状态数据作为齿轮箱正常的数据集，然后分别得到四类传感器得到的训练模型结果，在使用测试样本进行验证时，将模型得到样本分类的准确率记为 Pij (i=1,2,3,4, j=0,1），i 为第 i 个传感器，当 j=0 时，认为样本正常，当 j=1 时，认为样本发生故障。

我们将 36 个样本记为 Si (i=1,2,3,…36)，在四类传感器模型分类结果的概率之和来作为样本的分类结果。

对任意一个样本，记其正常的概率为 P0，发生故障的概率记为 P1，则其满足公式：

$$P_0 = \sum_{i=1}^{4} P_{i0} \tag{14}$$

$$P_1 = \sum_{i=1}^{4} P_{i1} \tag{15}$$

若 P0>P1，则认为其正常，若 P1>P0，则认为其发生故障
我们将 36 组测试组带入模型中求解，得到下表：

二分类	实际为正常	实际为故障	实际总量
预测正确	7	28	35

预测错误	**1**	**0**	**1**
预测总量	**8**	**28**	**36**

表 5：36 个测验组测试结果

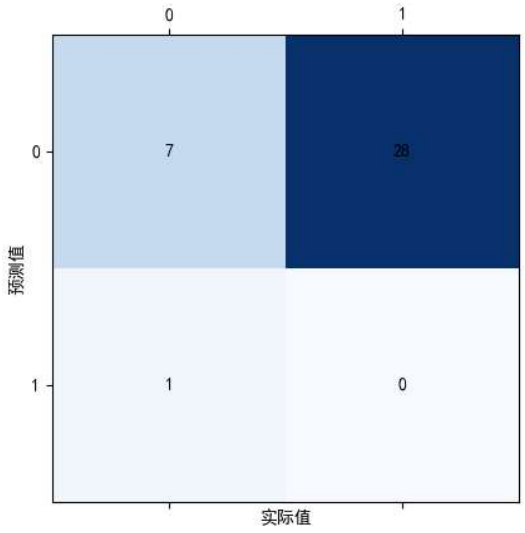


图 9：二分类混淆矩阵热力图

5.2.3 二分类混淆矩阵评价模型

（1） 混淆矩阵的基本概念

机器学习中对于分类模型常用混淆矩阵来进行效果评价，混淆矩阵中存在多个评价指标，这些评价指标可以从不同角度来评价分类结果的优劣。

在二分类模型中，假设结果只有 0 和 1 两类，最终的判别结果只有四种情况：实际为 0 被正确预测为 0，实际为 0 被错误预测为 1，实际为 1 被错误误测为 0，实际为 1 被正确预测为 1。

同理，齿轮箱也只有处于故障和不处于故障两种状态，于是便可以得到如下矩阵：

二分类	实际为正常	实际为故障
预测正确	TP	TN
预测错误	FP	FN

表 6：二分类法的混淆矩阵

（2）混淆矩阵的评价指标：

（1） 精确率[5]

精确率的含义是指：所有被预测为正常的测试数据中，真正是正常的比率。

$$\text{精确率} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{16}$$

（2） 召回率

召回率的含义是指：正确识别的正常个数在实际为正常的样本数中的占比。

$$召回率 = \frac{TP}{TP + FN} \tag{17}$$

（3） 分类准确率

分类准确率的含义是指：预测正确的结果在整体中的占比例

$$准确率 = \frac{TP + TN}{TP + TN + FN + FN} \tag{18}$$

（4） F 度量

F 度量是综合考量精确率和召回率的计算指标。

$$F\ 度量 = 2\ 精确度 * \frac{召回率}{精确度 + 召回率} \tag{19}$$

显然这些指标能够用来评价模型的能否准确判别齿轮箱的状态，并且这四种指标越大，说明建立的模型越好。

5.2.4 评价模型求解

我们用建立的分类模型对附件一中的正常状态和四种故障状态的数据进行分类，然后将分类模型预测的结果与实际情况相比较，以预测结果与真实的符合程度为依据来评价分类模型的性能。

求解的模型的指标如下：

精确度	召回率	F 度量	准确度
0.875	1	0.97	0.94

表 7：模型的指标

可以看出模型的正确率很高，可以用来判断齿轮箱是否处于故障状态

5.3 问题三

5.3.1 诊断模型建立

我们仍采用一维卷积神经网络训练的方法，在输入样本时，我们将附件一中五种状态的数据作为训练模型的数据集，然后分别得到四类传感器得到的训练模型结果。

训练模型经过 25 次迭代之后，其准确率不断升高，达到 60%，损失函数不断下降，说明拟合效果较好，迭代图像为：

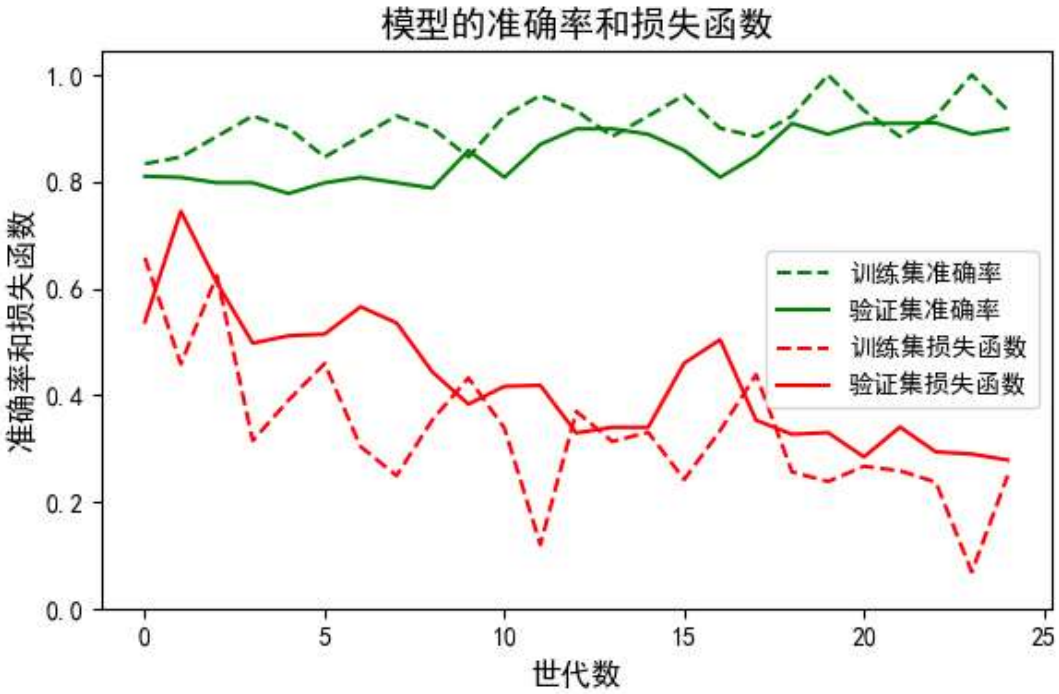


图 10：模型的迭代图

迭代图反映了训练集和验证集的准确度和损失函数随迭代次数的变化， 通过迭代图可以发现，模型的准确度最终能否达到 90%左右，能够很好的对齿轮箱的故障类型进行分类。

在使用测试样本进行验证时，模型得到样本分类的准确率记为 Pij（i=1,2,3,4, j=1,2,3,4,5），i 为第 i 个传感器，j 为附件一中的五种状态。

我们将 36 个样本记为 Si (i=1,2,3,…36)，在四类传感器模型分类结果的加权平均数来作为样本的分类结果。

对任意一个样本，记其为 j 种状态的概率为w_j，则其满足公式：

$$w_j = \sum_{i=1}^{4} Pij \qquad (20)$$

取 $$Pmax = \{W_1, W_2, W_3, W_4, W_5\} \qquad (21)$$

若$Pmax$ <0.5,则认为样本不明显属于这五种状态，故该样本为第六种状态，即发生其他故障。

若 max>0.5，则该样本对应为 wj 最大的 j 种状态；

4.3.2 训练模型结果测试

我们将 36 组测试组带入模型中求解，得到下表：

多分类	实际为正常	实际为故障 1	实际为故障 2	实际为故障 3	实际为故障 4	预测总量
预测为正常	7	0	0	0	0	7
预测为故障 1	0	6	0	1	0	7

预测为故障 2	0	1	7	0	0	8
预测为故障 3	0	0	0	6	0	6
预测为故障 4	0	0	1	1	5	7
预测为其他故障	0	1	0	0	0	1
实际总量	7	8	8	8	0	36

表 8：36 个测验组测试结果

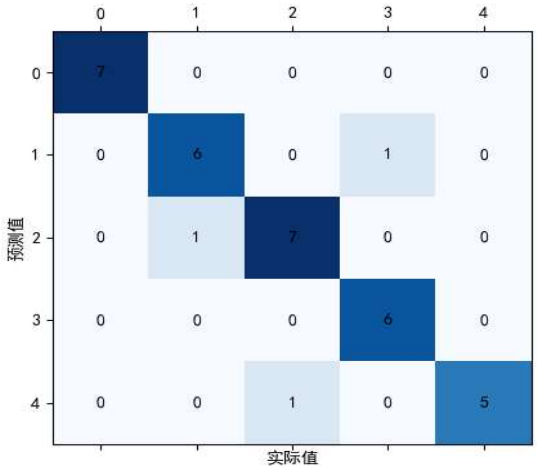


图 11：多分类混淆矩阵热力图

5.3.2 建立多分类混淆矩阵评价模型

多分类模型的预测结果有多种类型，而不只正常和故障两种，虽然如此，前面说的精确的、召回率、准确度同样适用于多分类问题。[6]

在附件一中，样本的状态一共有五种，分别为故障 1、故障 2、故障 3、故障 4、正常工况。但是也有可能被认为是其他故障。

对于故障 1 而言，可以建立如下表格

多分类	预测结果为故障 1	预测结果为非故障 1
预测结果是真实的	TP	TN
预测结果是虚假的	FP	FN

表 9：多分类法的混淆矩阵

该模型的评价指标于二分类模型相同，可以利用上述公式便可以求解出模型对判断各种状态的精确率，召回率以及分类准确率。

5.3.3 评价模型的求解

将 36 组测试集数据带入神经网络训练模型，求解如下：

	精确率	召回率	准确率
正常工况	1	1	1
故障 1	0.91	0.86	0.884
故障 2	0.79	0.86	0.823
故障 3	0.83	0.82	0.824
故障 4	0.86	0.94	0.898

表 10：模型的各项指标

可以发现模型对齿轮箱的各种情况都能较好的判别，但是，在测试中，有一个实际为故障 1 的样本被认为成其他故障，说明模型有一定的缺陷。[7]

5.4 问题 4

对 12 个测试数据判别是否发生故障，P0 为齿轮箱正常的概率，P1 为齿轮箱发生故障的概率，将数据集带入训练模型中，可以得到每种测试发生故障的概率如下：

	tset1	tset2	tset3	tset4	tset5	tset6
P0	0.0523	0.0217	0.9907	0.9887	0.0154	0.0157
P1	0.9477	0.9783	0.0093	0.0113	0.9846	0.9843

	tset7	tset8	tset9	tset10	tset11	tset12
P0	0.0285	0.0372	0.0115	0.0157	0.0185	0.0273
P1	0.9715	0.9628	0.9885	0.9843	0.9815	0.9727

表 11：12 个测试数据发生故障的概率

从表中可以发现 test3、test4 的 P0>>P1，其余测试的 P1>>P0，故可以认为 test3、test4 为正常工况，其余均为发生故障。

对 12 个测试数据的齿轮箱状态进行判别，P0 为齿轮箱正常的概率，P1~P4 分别为齿轮箱发生 1~4 种故障的概率，将数据集带入训练模型中，可以得到每种测试发生故障的概率如下：

	tset1	tset2	tset3	tset4	tset5	tset6
P0	0.1614	0.0442	0.9275	0.9512	0.1247	0.0445
P1	0.0814	0.7875	0.0127	0.0134	0.6434	0.0957
P2	0.6843	0.1164	0.0340	0.0098	0.0357	0.7664
P3	0.0345	0.0357	0.0143	0.0167	0.0994	0.0557
P4	0.0384	0.0162	0.01137	0.0089	0.0998	0.0365

	tset7	tset8	tset9	tset10	tset11	tset12
P0	0.0872	0.0324	0.0772	0.0314	0.0443	0.2149
P1	0.1148	0.0728	0.0431	0.0424	0.0327	0.0724
P2	0.1942	0.0413	0.0270	0.0770	0.0413	0.2642
P3	0.2117	0.1111	0.7924	0.8175	0.0514	0.1342
P4	0.3921	0.7424	0.0603	0.0317	0.8303	0.3142

表 12：12 个测试数据不同状况的概率

从表中可以发现 test3、test4 的 P0 为 0.95 左右，说明上一模型结构正确，为正常工况，可以认为 test3、test4 为正常工况，其余均为发生故障；Test 1、2、5、6、8、9、10、11 的 P1~5 有明显的一个值很大 ，故可以判定其状态，test 7、12 的概率均小于 0.4，故无明显相似于已知的五种状态，故认为是其他故障。

综上，12 个 test 最终状态为：

	tset1	tset2	tset3	tset4	tset5	tset6
状态	故障 2	故障 1	正常工况	正常工况	故障 1	故障 2
	tset7	tset8	tset9	tset10	tset11	tset12
状态	其他故障	故障 4	故障 3	故障 3	故障 4	其他故障

表 13：12 个测试样本的状况

六、模型的评价

6.1 模型优点

1. 本文在建模之前都对数据进行了预处理，使得数据更加真实可靠。
2. 本文建立的模型都经过检测，可以得到较高的准确度。
3. 本文在解决问题时，使用的模型较为简单、易于理解，简化了问题的求解。

6.2 模型缺点

1. 本题数据量较大，建立神经网络模型时没有使用专门的数据处理软件，数据处理时操作繁琐，等待时间较长。
2. 对于传感器 4 的神经网络模型准确度较低，不能很好的判断齿轮箱的状态，存在错误判断齿轮箱故障状态的可能
3. 在判断处于其他故障的数据集时，没有建立准确的诊断模型，具有一定的主观性。

6.3 模型的推广

将模型的缺点进行改进后，本文建立的模型除判断齿轮箱故障状态后，还可以用来判断其他振动信号的故障问题，具有一定的实际应用价值。

参考文献

[1]李晓虎，贾民平，许飞云．频谱分析法在齿轮箱故障诊断中的应用[J]．振动，测试与诊断，2003, 23(3): 168-170.

[2]雷亚国，何正嘉，林京，等．行星齿轮箱故障诊断技术的研究进展[J]．机械工程学报，2011, 47(19): 59-67.

[3]谢向荣，陈洪峰，朱石坚，等．关联维数在齿轮箱振动信号特征提取中的应用[J]．海军工程大学学报, 2005, 17(6): 102-107.

[4]Asi O. Fatigue failure of a helical gear in a gearbox[J]. Engineering failure analysis, 2006, 13(7): 1116-1125.

机械工程学报, 2014, 50(17): 61-68.

[5]Shalev-Shwartz, & Shai. (2014). Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press.

[6]龙泉，刘永前，杨勇平．基于粒子群优化 BP 神经网络的风电机组齿轮箱故障诊断方法[J]．太阳能学报, 2012, 33(1): 120-125.

[7]雷亚国，汤伟，孔德同，等．基于传动机理分析的行星齿轮箱振动信号仿真及其故障诊断[J].

附录 1：计算结果

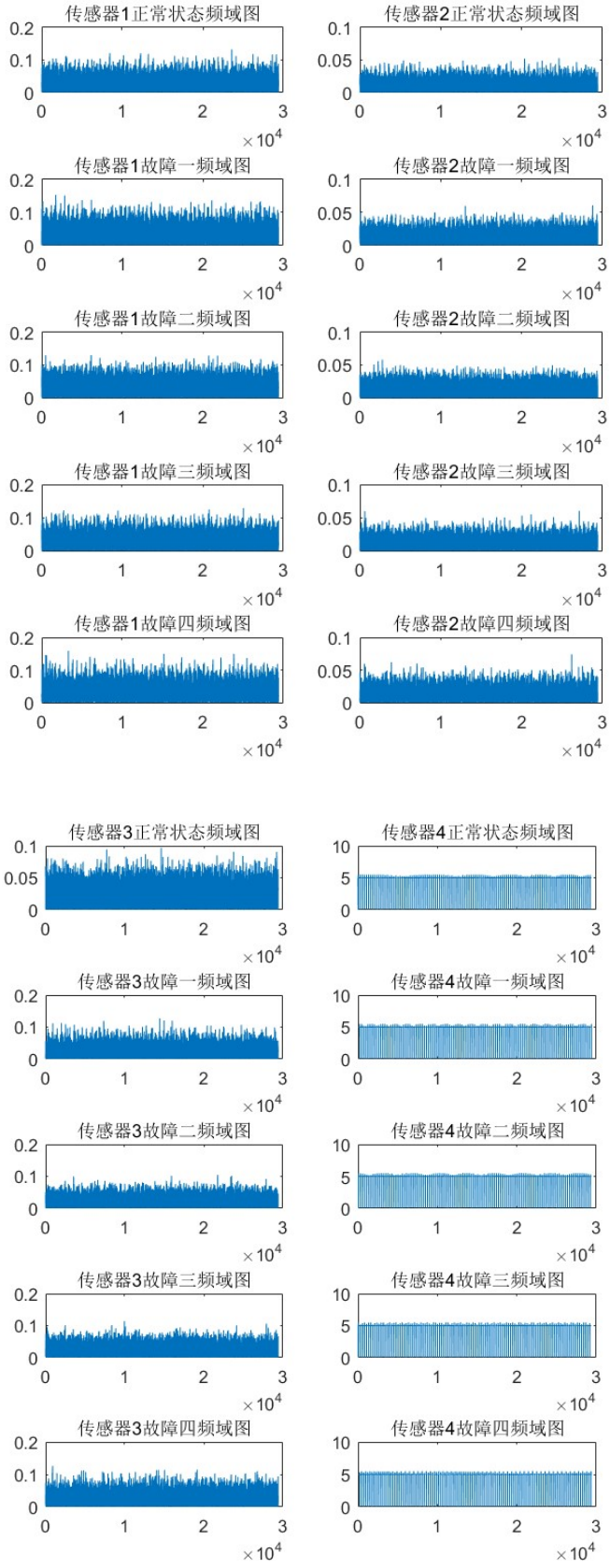
时域信号特征

		均值	方差	峰峰值	波形指标	峰值指标	脉冲指标	裕度指标	峭度指标
传感器1	正常	0.0005	0.001	0.253	1.2528	3.7595	4.7097	5.5658	2.94
	故障 1	0.001	0.0015	0.2849	1.2456	3.9585	4.9309	5.8043	2.8562
	故障 2	0.0003	0.0012	0.2558	1.2365	3.6126	4.467	5.225	2.7738
	故障 3	0.0003	0.0011	0.2553	1.2482	3.8426	4.7965	5.6444	2.9136
	故障 4	0.0003	0.0011	0.2553	1.2482	3.8426	4.7965	5.6444	2.9136
传感器2	正常	0.0004	0.0002	0.1018	1.2463	3.671	4.5753	5.385	2.8989
	故障 1	-0.0001	0.0002	0.1103	1.2537	3.6144	4.5314	5.3493	3.0095
	故障 2	0.0024	0.0002	0.1063	1.2537	3.9737	4.9817	5.8953	2.9164
	故障 3	0.0007	0.0002	0.1138	1.2515	3.8234	4.7849	5.6411	2.9719
	故障 4	0.0001	0.0002	0.1365	1.2574	4.9185	6.1847	7.3108	3.0751
传感器3	正常	0.0013	0.0006	0.1909	1.2508	3.8771	4.8495	5.721	2.9337
	故障 1	0.001	0.0008	0.247	1.2577	4.4935	5.6513	6.6845	3.1083
	故障 2	0.001	0.0006	0.2067	1.2509	4.1531	5.1949	6.1226	2.9546
	故障 3	0.0007	0.0007	0.2153	1.2545	3.9218	4.9199	5.808	3.0118
	故障 4	0.001	0.0009	0.2399	1.2538	4.2512	5.3301	6.2915	3.0354
传感器4	正常	4.944	0.2846	5.784	4.2099	1.0432	4.3916	7.1296	78.4193
	故障 1	4.943	0.2756	5.7443	4.2513	1.0781	4.5832	7.4534	80.5085
	故障 2	4.9442	0.2759	5.7475	4.2509	1.0777	4.5811	7.4519	80.4223
	故障 3	4.9439	0.2746	5.7393	4.2582	1.0804	4.6008	7.4882	80.7192
	故障 4	4.9418	0.2744	5.736	4.2513	1.0747	4.5689	7.4353	80.6038

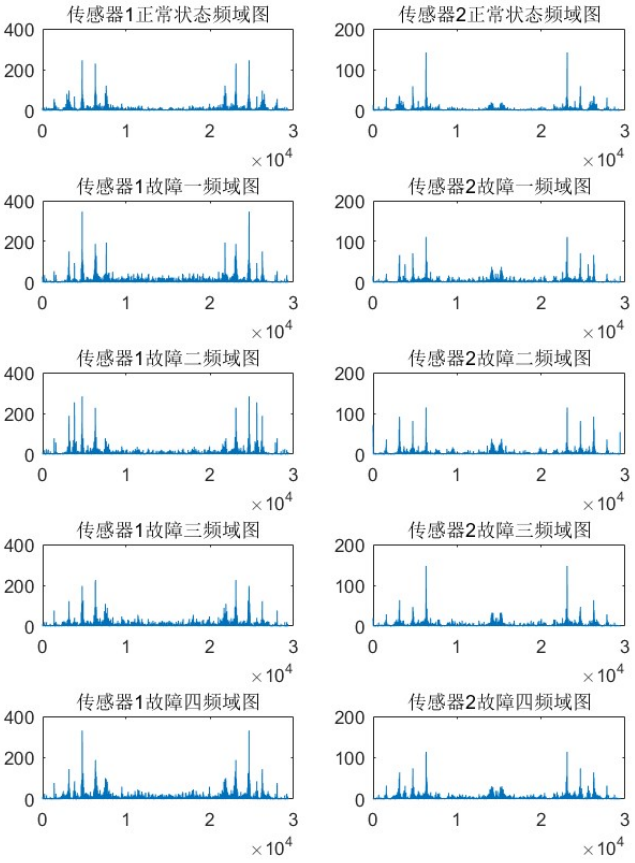
频域信号特征

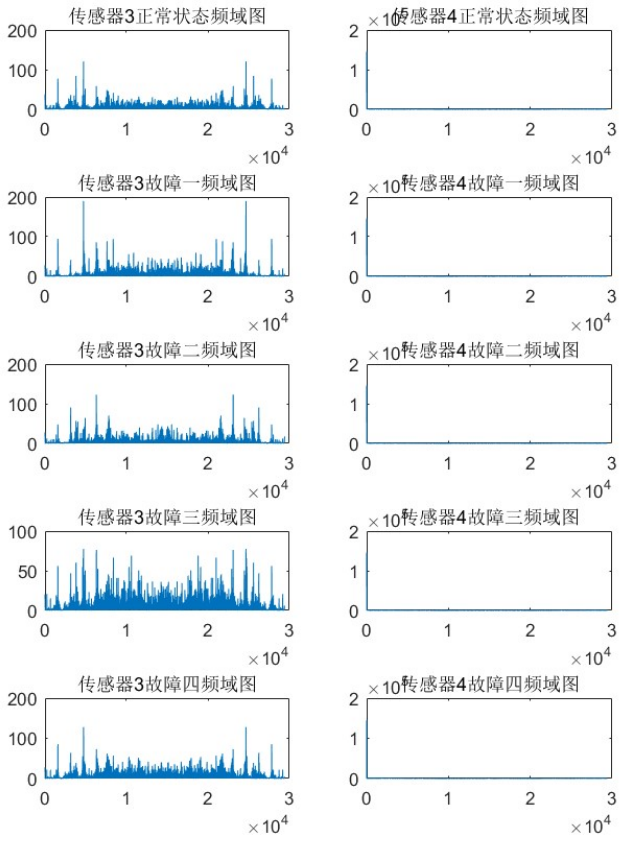
		重心频率	均方频率	频率方差	相对功率谱熵
传感器1	正常	1601.831486	3217783.27	651919.1609	0.578995646
	故障 1	1774.676716	3777247.188	627769.7415	0.536222207
	故障 2	1645.085163	3321854.187	615548.9951	0.522793044
	故障 3	1746.604739	3670229.648	619601.5319	0.594980275
	故障 4	1795.160181	3820410.637	597810.5603	0.579844129
传感器2	正常	1775.455699	4011541.373	859298.4323	0.555307415
	故障 1	1829.724058	4135136.096	787245.9695	0.555392667
	故障 2	1764.948024	3989077.959	874036.4317	0.542839756
	故障 3	1814.517516	4109162.808	816688.9907	0.542266698
	故障 4	1878.889829	4332783.681	802556.6917	0.590085174
传感器3	正常	1926.896311	4401569.287	688639.8939	0.758549643
	故障 1	1961.088706	4452404.921	606536.0083	0.678229186
	故障 2	1917.065661	4310142.973	635002.226	0.736665119
	故障 3	1952.262064	4437574.634	626247.4679	0.750494712
	故障 4	2017.886774	4664295.879	592428.8441	0.748255674
传感器4	正常	1031.516826	1805852.968	741826.0056	0.53184696
	故障 1	993.0189886	1682484.825	696398.1128	0.523519072
	故障 2	991.5257334	1679039.717	695916.4373	0.52328679
	故障 3	995.4946787	1688037.564	697027.9086	0.524011582
	故障 4	1281.327384	2456113.546	814313.6803	0.460607065

时域特性分析图谱


传感器1正常状态频域图
传感器2正常状态频域图
传感器1故障一频域图
传感器2故障一频域图
传感器1故障二频域图
传感器2故障二频域图
传感器1故障三频域图
传感器2故障三频域图
传感器1故障四频域图
传感器2故障四频域图
传感器3正常状态频域图
传感器4正常状态频域图
传感器3故障一频域图
传感器4故障一频域图
传感器3故障二频域图
传感器4故障二频域图
传感器3故障三频域图
传感器4故障三频域图
传感器3故障四频域图
传感器4故障四频域图
×10^4

频域特征图谱


传感器1正常状态频域图
传感器2正常状态频域图
传感器1故障一频域图
传感器2故障一频域图
传感器1故障二频域图
传感器2故障二频域图
传感器1故障三频域图
传感器2故障三频域图
传感器1故障四频域图
传感器2故障四频域图


传感器3正常状态频域图
传感器4正常状态频域图
传感器3故障一频域图
传感器4故障一频域图
传感器3故障二频域图
传感器4故障二频域图
传感器3故障三频域图
传感器4故障三频域图
传感器3故障四频域图
传感器4故障四频域图

附录 2：程序代码

Matlab 处理信号程序

```
%matlab 分析信号程序
load('matlab.mat');%读入表格数据
xn=a1;
fs=6400;N=29400;
bianhuan(xn);
f0=20;
out = tad(xn,fs,f0);
figure;
plot(abs(fft(out)));
```

Matlab 对时域信号的时域平均降噪处理

```
% 实现时域信号的时域平均降噪函数
function out = tad(x,fs,f0)
% 获取 x 的数据长度
len = length(x);
x = x(:)';
% 计算一个周期有多少个数据点
N = round(fs/f0);
Nfr = fs/f0;
% 计算原数据点有包含多少个整周期
M = floor(len/Nfr);
% 结果存储数组
out = zeros(1,N);

% 计算每个周期的起始点
ind = zeros(1,M);
for i=0:M-1
ind(i+1) = round(i*Nfr)+1;
end
for i=1:M
out=out+x(ind(i):ind(i)+N-1);
end
out = out/M;

% figure
figure;
plot((0:N-1)/fs,out);
title('TDA');
xlabel('Time / t');
```

```
29.
30.     % % 数据段划分演示
31.     % figure
32.     % plot((0:len-1)/fs,x);
33.     % hold on;
34.     % plot((ind-1)/fs,x(ind),'.');
35.     % title('Liu TDA illustration');
36.     % xlabel('Time / t');
37.     end
```

Matlab 对信号进行时域，频域等的图像显示

```
1.  %求时域图，频域图，倒频谱，功率谱的函数
2.  function bianhuan(xn)
3.  fs=6400;N=29400;%采样频率和采样点数
4.  L=length(xn);
5.  t=0:1/fs:(N-1)/fs;
6.  temp=fft(xn,L);
7.  A=abs(temp)/(N/2);
8.  A(1)=A(1)/2;
9.  f=(0:N-1)/N*fs;
10. subplot(4,1,1);plot(t,xn);
11. xlabel('time');ylabel('幅值');
12. title('时域');
13.
14. subplot(4,1,2)
15. plot(f(1:N/2),A(1:N/2));
16. xlabel('f');ylabel('幅值');
17. title('频域');
18. z = rceps(xn);
19. subplot(4,1,3);plot(t,abs(z));
20. title('倒频谱');ylim([0 0.3]);
21. xlabel('time');ylabel('幅值');
22.
23. cxn = xcorr(xn,'unbiased');
24. CXk = fft(cxn,N);
25. psd2 = abs(CXk);
26. index = 0:round(N/2-1);
27. k = index*fs/N;
28. psd2 = psd2/max(psd2);
29. psd2 = 10*log10(psd2(index+1));
30. subplot(4,1,4);plot(k,psd2);
31. title('功率谱');
32. xlabel('f');ylabel('幅值');
33. grid on
```

```
end
```

Matlab 对计算时域，频域的特征参数

```
function time=time_statistical_compute(x)
%%对时域信号进行统计量分析
%% p2,p10 返回有量纲指标，f1,f2,f3,f4,f5 返回无量纲指标
N=length(x);
p1=mean(x); %均值
x=x-p1;
p2=sqrt(sum(x.^2)/N); %均方根值,又称有效值(!)
p3=(sum(sqrt(abs(x)))/N).^2; %方根幅值(!)
p4=sum(abs(x))/N; %绝对平均值
p5=sum(x.^3)/N; %歪度
p6=sum(x.^4)/N; %峭度
p7=sum((x).^2)/N; %方差
p8=max(x);%最大值
p9=min(x);%最小值
p10=p8-p9;%峰峰值
%%以上都是有量纲统计量，以下是无量纲统计量
f1=p2/p4; %波形指标
f2=p8/p2; %峰值指标
f3=p8/p4; %脉冲指标
f4=p8/p3; %裕度指标
f5=p6/((p2)^4); %峭度指标
time=[p1,p2,p3,p4,p5,p6,p7,p10,f1,f2,f3,f4,f5];
```

Python 对神经网络模型进行测试

```
import pandas as pd
import math
from keras.layers import *
from keras.models import *
import warnings
from xlutils.copy import copy
import xlrd as xr

num = 2  # 1，2，3，4 分别表示对应的信号
TEST_MANIFEST_DIR = r'D:\Desktop\two.xls'
TIME_PERIODS = 1000  # 表示一个组中有多少数据，将 29400 个数据分为 29 组，每组 1000 个
Batch_size = 10  # 表示一次读入多少个数据
Long = 145  # 数据中组的个数
Lens = 36  # 表示验证集的个数

```

```
16.
17.     def convert2oneHot(index, lens):
18.         hot = np.zeros((lens,))
19.         hot[int(index)] = 1
20.         return (hot)
21.
22.
23.     # 读入数据
24.     def ts_gen(path=TEST_MANIFEST_DIR, batch_size=Batch_size):
25.         img_list = []
26.         tablename = ['test1', 'test2', 'test3', 'test4', 'test5', 'test6', 'test7', 'test8', 'test9', 'test10', 'test
    11',
27.                      'test12']
28.         for n in range(12):
29.
30.             df = pd.read_excel(path, sheet_name=tablename[n], names=['value'], usecols=[num]
    )
31.             data = df.values
32.             temp = []
33.             i = 0
34.             for j in range(i * 1000, i * 1000 + 1000):
35.                 temp.append(data[j][0])
36.             temp.append(data[999][0])
37.             img_list.append(np.array(temp))
38.         np.random.shuffle(img_list)
39.         np.random.shuffle(img_list)
40.         np.random.shuffle(img_list)
41.         img_list = np.array(img_list)
42.         img_list = np.array(img_list)
43.         print(len(img_list))
44.         print(len(img_list[1]))
45.         print(img_list)
46.         img_list = np.array(img_list)[:Lens]
47.         print("Found %s test items." % len(img_list))
48.         print("list 1 is", img_list[0, -1])
49.         steps = math.ceil(len(img_list) / batch_size)
50.         while True:
51.             for i in range(steps):
52.                 batch_list = img_list[i * batch_size:i * batch_size + batch_size]
53.                 batch_x = np.array([file for file in batch_list[:, 1:]])
54.                 yield batch_x
55.
56.
57.     # 加载模型，测试，得出结果，并保存
```

```
if __name__ == "__main__":
    test_iter = ts_gen()
    string = "D:\Desktop\data\\best_model.20-1.2187.h5"
    model = load_model(string, compile=False)
    pres = model.predict_generator(generator=test_iter, steps=math.ceil(528 / Batch_size), verbose=1)
    print(pres.shape)
    ohpres = np.argmax(pres, axis=1)
    print(ohpres.shape)
    df = pd.DataFrame()
    df["id"] = np.arange(1, len(ohpres) + 1)
    df["label"] = ohpres
    df.to_csv("D:\Desktop\data\\predicts.csv", index=None)
    tesft_iter = ts_gen()
    list = model.predict_generator(generator=test_iter, steps=math.ceil(520 / Batch_size), verbose=1)[:12]
    print(list)
    temp = []
    pres = []
    for i in list:
        temp = i.tolist()
        pres.append(temp)
    print(pres)
    file = "D:\Desktop\gailv.xls"
    oldwb = xr.open_workbook(file)
    newwb = copy(oldwb)
    newws = newwb.get_sheet(num - 1)
    for i in range(len(pres)):
        for j in range(len(pres[0])):
            newws.write(i + 1, j + 1, pres[i][j])
    ohpres = np.argmax(pres, axis=1)
    ohpres = ohpres[:12]
    ohpres = ohpres.tolist()
    for i in range(len(ohpres)):
        newws.write(i + 1, 0, ohpres[i])
    newwb.save(file)  # 保存修改
    print(ohpres[:12])
```

Python 对频域模型进行训练

```
import keras
import matplotlib.pyplot as plt
import pandas as pd
import math
from keras.layers import *
```

```
from keras.models import *
from keras.callbacks import ModelCheckpoint
import warnings
from scipy.fftpack import fft, ifft

warnings.filterwarnings(module='sklearn*', action='ignore', category=DeprecationWarning)

MANIFEST_DIR = r'D:\Desktop\one.xls'#训练集与验证集地址
TIME_PERIODS = 1000#表示一个组中有多少数据，将 29400 个数据分为 29 组，每组 1000 个
Batch_size = 10#表示一次读入多少个数据
Long = 145#数据中组的个数
Lens = 36#表示验证集的个数

def convert2oneHot(index, lens):
    hot = np.zeros((lens,))
    hot[int(index)] = 1
    return(hot)

#读入训练集与验证集，进行傅里叶转换，并预处理
def xs_gen(path=MANIFEST_DIR, batch_size=Batch_size, train=True, Lens=Lens):
    img_list = []
    tablename = ['gearbox00', 'gearbox10', 'gearbox20', 'gearbox30', 'gearbox40']
    for n in range(5):

        df = pd.read_excel(path, sheet_name=tablename[n], names=['value'], usecols=[2])
        data = df.values
        # print(data)
        for i in range(29):
            temp = []
            for j in range(i * 1000, i * 1000 + 1000 + 2):
                temp.append(data[j][0])
            y = temp
            fft_y = fft(y)
            N = len(y)
            x = np.arange(N)
            half_x = x[range(int(N / 2))]

            abs_y = np.abs(fft_y)
            angle_y = np.angle(fft_y)
            normalization_y = abs_y / N
            normalization_half_y = normalization_y[range(int(N / 2))]
            temp = []
```

```
                for j in normalization_half_y:
                    temp.append(j)
                temp.append(n)
                img_list.append(np.array(temp))

        np.random.shuffle(img_list)
        if train:
            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)
        else:
            img_list = np.array(img_list)[Lens:]
            print("Found %s test items." % len(img_list))
            print("list 1 is", img_list[0, -1])
            steps = math.ceil(len(img_list) / batch_size)
        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 = np.array([file for file in batch_list[:, 1:-1]])
                batch_y = np.array([convert2oneHot(label, 5) for label in batch_list[:, -1]])
                yield batch_x, batch_y

    #卷积模型神经网络
    def build_model(input_shape=(TIME_PERIODS,), num_classes=5):
        model = Sequential()
        model.add(Reshape((TIME_PERIODS, 1), input_shape=input_shape))

        model.add(Conv1D(128, 8, strides=1, activation='tanh', input_shape=(TIME_PERIOD
    S, 1)))
        model.add(Conv1D(128, 8, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))
        model.add(Conv1D(256, 8, strides=1, activation='tanh', padding="same"))
        model.add(Conv1D(256, 8, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))
        model.add(Conv1D(512, 8, strides=1, activation='tanh', padding="same"))
        model.add(Conv1D(512, 8, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))
        model.add(Conv1D(1024, 2, strides=1, activation='tanh', padding="same"))
        model.add(Conv1D(1024, 2, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))

        model.add(GlobalAveragePooling1D())
```

```
        model.add(Dropout(0.3))
        model.add(Dense(num_classes, activation='softmax'))
        return(model)

    #将结果绘图
    def show_train_history(train_history, train, validation):
        plt.plot(train_history.history[train])
        plt.plot(train_history.history[validation])
        plt.ylabel('Train History')
        plt.ylabel(train)
        plt.xlabel('Epoch')
        plt.legend(['train', 'validation'], loc='upper left')
        plt.show()

    if __name__ == "__main__":
        train_iter = xs_gen()
        val_iter = xs_gen(train=False)
        ckpt = ModelCheckpoint(
            filepath='D:\Desktop\data\\best_model.{epoch:02d}-{val_loss:.4f}.h5',
            monitor='val_loss', save_best_only=True, verbose=1
        )
        model = build_model()
        opt = keras.optimizers.Adam(0.0002)
        model.compile(loss='categorical_crossentropy',
                      optimizer=opt, metrics=['accuracy'])
        print(model.summary())
        train_history = model.fit_generator(
            generator=train_iter,
            steps_per_epoch=Lens // Batch_size,
            epochs=25,
            initial_epoch=0,
            validation_data=val_iter,
            validation_steps=(Long - Lens) // Batch_size,
            callbacks=[ckpt],
        )
        model.save("D:\Desktop\data\\finishModel.h5")
        #结果输出
        show_train_history(train_history, 'acc', 'val_acc')
        show_train_history(train_history, 'loss', 'val_loss')
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.figure(figsize=(6, 4))
        plt.plot(train_history.history['acc'], "g--", label="训练集准确率")
        plt.plot(train_history.history['val_acc'], "g", label="验证集准确率")
        plt.plot(train_history.history['loss'], "r--", label="训练集损失函数")
```

```
        plt.plot(train_history.history['val_loss'], "r", label="验证集损失函数")
        plt.title('模型的准确率和损失函数', fontsize=14)
        plt.ylabel('准确率和损失函数', fontsize=12)
        plt.xlabel('世代数', fontsize=12)
        plt.ylim(0)
        plt.legend()
        plt.show()
```

Python 对时域模型进行训练

```
import keras
import matplotlib.pyplot as plt
import pandas as pd
import math
from keras.layers import *
from keras.models import *
from keras.callbacks import ModelCheckpoint

MANIFEST_DIR = r'D:\Desktop\one.xls'#训练集与验证集地址
TIME_PERIODS = 1000#表示一个组中有多少数据，将 29400 个数据分为 29 组，每组 1000 个
Batch_size = 10#表示一次读入多少个数据
Long = 145#数据中组的个数
Lens = 36#表示验证集的个数

def convert2oneHot(index, lens):
    hot = np.zeros((lens,))
    hot[int(index)] = 1
    return(hot)

#读入训练集与验证集，并预处理
def xs_gen(path=MANIFEST_DIR, batch_size=Batch_size, train=True, Lens=Lens):
    img_list = []
    tablename = ['gearbox00', 'gearbox10', 'gearbox20', 'gearbox30', 'gearbox40']
    for n in range(5):
        pass
        df = pd.read_excel(path, sheet_name=tablename[n], names=['value'], usecols=[2])
        data = df.values
        # print(data)
        for i in range(29):
            temp = []

            for j in range(i * 1000, i * 1000 + 1000 + 1):
                temp.append(data[j][0])
            temp.append(n)
```

```
                img_list.append(np.array(temp))
        np.random.shuffle(img_list)

        if train:
            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)
        else:
            img_list = np.array(img_list)[Lens:]
            print("Found %s test items." % len(img_list))
            print("list 1 is", img_list[0, -1])
            steps = math.ceil(len(img_list) / batch_size)
        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 = np.array([file for file in batch_list[:, 1:-1]])
                batch_y = np.array([convert2oneHot(label, 5) for label in batch_list[:, -1]])
                yield batch_x, batch_y

    #建立卷积神经网络
    def build_model(input_shape=(TIME_PERIODS,), num_classes=5):
        model = Sequential()
        model.add(Reshape((TIME_PERIODS, 1), input_shape=input_shape))

        model.add(Conv1D(128, 8, strides=1, activation='tanh', input_shape=(TIME_PERIOD
    S, 1)))
        model.add(Conv1D(128, 8, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))

        model.add(Conv1D(256, 8, strides=1, activation='tanh', padding="same"))
        model.add(Conv1D(256, 8, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))

        model.add(Conv1D(512, 8, strides=1, activation='tanh', padding="same"))
        model.add(Conv1D(512, 8, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))

        model.add(Conv1D(1024, 2, strides=1, activation='tanh', padding="same"))
        model.add(Conv1D(1024, 2, strides=1, activation='tanh', padding="same"))
        model.add(MaxPooling1D(2))

        model.add(GlobalAveragePooling1D())
```

```
        model.add(Dropout(0.3))
        model.add(Dense(num_classes, activation='softmax'))
        return(model)
    #将结果绘图
    def show_train_history(train_history, train, validation):
        plt.plot(train_history.history[train])
        plt.plot(train_history.history[validation])
        plt.ylabel('Train History')
        plt.ylabel(train)
        plt.xlabel('Epoch')
        plt.legend(['train', 'validation'], loc='upper left')
        plt.show()

    if __name__ == "__main__":
        train_iter = xs_gen()
        val_iter = xs_gen(train=False)

        ckpt = ModelCheckpoint(
            filepath='D:\Desktop\data\\best_model.{epoch:02d}-{val_loss:.4f}.h5',
            monitor='val_loss', save_best_only=True, verbose=1
        )

        model = build_model()
        opt = keras.optimizers.Adam(0.0002)
        model.compile(loss='categorical_crossentropy',
                      optimizer=opt, metrics=['accuracy'])
        print(model.summary())

        train_history = model.fit_generator(
            generator=train_iter,
            steps_per_epoch=Lens // Batch_size,
            epochs=25,
            initial_epoch=0,
            validation_data=val_iter,
            validation_steps=(Long - Lens) // Batch_size,
            callbacks=[ckpt],
        )
        model.save("D:\Desktop\data\\finishModel.h5")

        #将输出结果可视化
        show_train_history(train_history, 'acc', 'val_acc')
        show_train_history(train_history, 'loss', 'val_loss')
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.figure(figsize=(6, 4))
```

```
122.        plt.plot(train_history.history['acc'], "g--", label="训练集准确率")
123.        plt.plot(train_history.history['val_acc'], "g", label="验证集准确率")
124.        plt.plot(train_history.history['loss'], "r--", label="训练集损失函数")
125.        plt.plot(train_history.history['val_loss'], "r", label="验证集损失函数")
126.        plt.title('模型的准确率和损失函数', fontsize=14)
127.        plt.ylabel('准确率和损失函数', fontsize=12)
128.        plt.xlabel('世代数', fontsize=12)
129.        plt.ylim(0)
130.        plt.legend()
131.        plt.show()
```