基于深度学习的心律失常诊断和管理分析系统

------韩卓琪

使用WFDB读取数据

读取.hea文件

In [5]:

```
from IPython.display import display
import wfdb
record = wfdb.rdheader('../mit-bih-arrhythmia-database-1.0.0/100')
# display(record.__dict__)
```

executed in 13ms, finished 17:19:59 2021-01-23

读取record数据

使用rdrecord函数，该函数的返回值为一个wfdb中定义的record对象。

```
def rdrecord(record_name, sampfrom=0, sampto=None, channels=None,
             physical=True, pb_dir=None, m2s=True, smooth_frames=True,
            ignore_skew=False, return_res=64, force_channels=True,
          channel_names=None, warn_empty=False):
```

常用的重要参数:

- record_name : 储存心电信号的路径;
- sampfrom : 起始位置;
- sampto : 终止位置;
- channels :optional，选择读取某个通道的数据，默认读取全部通道;

In [7]:

```
from IPython.display import display
import wfdb
record=wfdb.rdrecord('../mit-bih-arrhythmia-database-1.0.0/100')
# display(record.__dict__)
```

executed in 54ms, finished 17:21:57 2021-01-23

几个经常使用的属性值:

1. fs：采样频率;
2. n_sig：信号通道数;
3. sig_len：信号长度;
4. p_signal：模拟信号值，储存形式为ndarray或者是list;
5. d_signal：数字信号值，储存形式为ndarray或者是list。

这些属性都能直接进行访问（如：使用record.fs可以直接读取到采样频率)。

读取.art文件

In [8]:

```
import wfdb
annotation=wfdb.rdann('../mit-bih-arrhythmia-database-1.0.0/100', 'atr')
# display(annotation.__dict__)
```

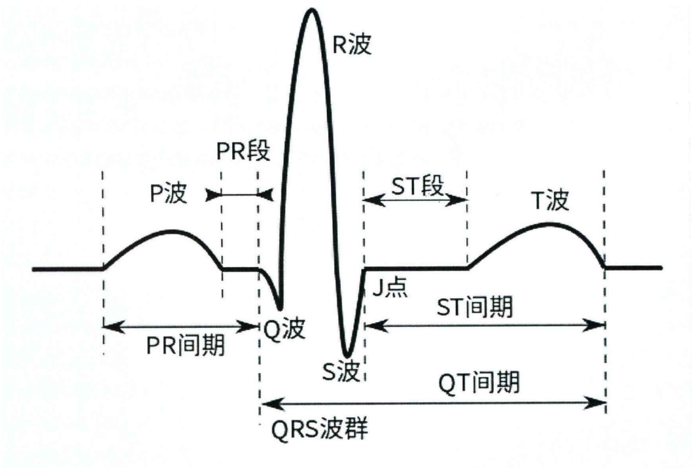
executed in 50ms, finished 17:22:02 2021-01-23

其中的symbol为心拍注释（包括了正常类型N和各种异常类型）

常见的心拍注释

- 正常心拍 | Normal beat | N | N
- 左束支传导阻滞 | Left bundle branch block beat | LBBB | L
- 右束传导支阻滞 | Right bundle branch block beat | RBBB | R
- 房性早搏 | Atrial premature beat | APB | A
- 室性早搏 | Premature ventricular contraction | PVC | V

数据预处理和模型数据集构建



数据分布统计

In [221]:

```
import os
type=[]
rootdir = '../mit-bih-arrhythmia-database-1.0.0'          # 设置根路径
files = os.listdir(rootdir) #列出文件夹下所有的目录与文件
name_list=[]
last_name_list=[]             # last_name_list=[100,101,...234]
MLII=[]                        # 用MLII型导联采集的人
type={}                        # 标记及其数量
for file in files:
    if file[0:3] in name_list:    # 根据数据库实际情况调整熟知，这里判断的是每个文件的前三个字符
        continue
    else:
        name_list.append(file[0:3])
for name in name_list:         # 遍历每一个数据文件
    if name[0] not in ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']:          # 跳过无用的文件
        continue
    last_name_list.append(name)
    record = wfdb.rdrecord(rootdir+'/'+name)    # 读取一条记录（100），不用加扩展名
```

executed in 1.95s, finished 09:04:13 2021-01-24

对每一条数据的MLII导联通道的心拍类型做一个统计

In [253]:

```
import pandas as pd

for name in last_name_list:         # 遍历每一个人
    if 'MLII' in record.sig_name:          # 选取一种导联方式（这里以MLII导联为例）
        MLII.append(name)                  # 记录下这个文件
    annotation = wfdb.rdann(rootdir+'/'+name, 'atr')    # 然后读取一条记录的atr文件，扩展名atr
    for symbol in annotation.symbol:                   # 同时记录下这个文件对应的标记类型
        if symbol in list(type.keys()):
            type[symbol]+=1
        else:
            type[symbol]=1
print('sympbol_name',type)
type_frampe = pd.DataFrame(list(type.values()))
type_pie_data = list(type_frampe.values.reshape(1,-1)[0])
```

executed in 1.80s, finished 09:21:33 2021-01-24

```
sympbol_name {'+': 33566, 'N': 1951352, 'A': 66196, 'V': 185380, '~': 16016, '|': 3432, 'Q': 858, '/': 182728, 'f': 25532, 'x': 5018, 'F': 20878, 'j': 5954, 'L': 209950, 'a': 3900, 'J': 2158, 'R': 188734, '[': 156, '!': 12272, ']': 156, 'E': 2756, 'S': 52, '"': 11362, 'e': 416}
```

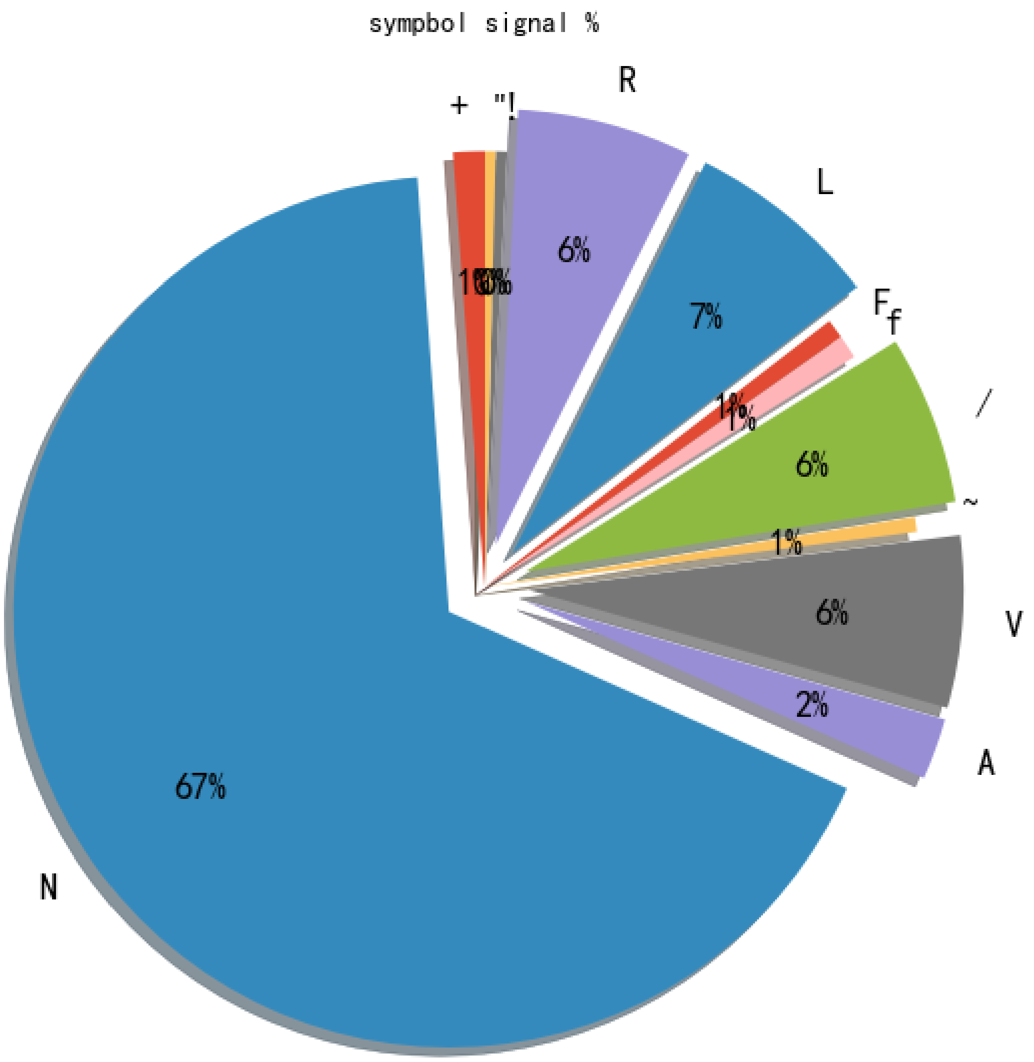
In [296]:

```
plt.figure(figsize=(10,10))
plt.style.use('ggplot')

label=['+', 'N', 'A', 'V', '~', '/', 'f', 'F', 'L', 'R', '!', '"']
explode = (0, 0.1, 0.1, 0.1, 0, 0.1, 0, 0, 0.1, 0.1, 0, 0)   # only "explode" the 2nd slice (i.e.
'Hogs')

plt.pie([33566, 1951352, 66196, 185380, 16016, 182728, 25532, 20878, 209950, 188734, 12272,
11362],labels=label, autopct='%.0f%%', shadow=True, startangle= 90, textprops={'fontsize': 18},
pctdistance = 0.7, labeldistance = 1.1, explode=explode)
plt.title('sympbol signal %')
```

executed in 296ms, finished 09:40:50 2021-01-24

Out[296]:

Text(0.5, 1.0, 'sympbol signal %')



数据裁剪转换成EXCEL TODO

In [14]:

```
import pywt
import matplotlib.pyplot as plt

# 测试集在数据集中所占的比例
RATIO = 0.3
```

executed in 7ms, finished 17:23:18 2021-01-23

小波变换去噪算法

小波变换有两个变量：尺度a（scale）和平移量 b（translation）

尺度a控制小波函数的伸缩， 平移量b控制小波函数的平移。尺度就对应于频率(反比），平移量b就对应于时间。

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)$$

函数x在标度a的子空间上的投影形式

$$x_a(t) = \int_{\mathbb{R}} WT_\psi\{x\}(a,b) \cdot \psi_{a,b}(t)db$$

小波系数公式：

$$WT_\psi\{x\}(a,b) = \langle x, \psi_{a,b}\rangle = \int_{\mathbb{R}} x(t)\psi_{a,b}(t)dt$$

阈值公式：

$$\lambda = \frac{median|w|\sqrt{2lnN}}{0.6745}$$

In [131]:

```
# 小波去噪预处理
def denoise(data):
    # 小波变换
    coeffs = pywt.wavedec(data=data, wavelet='db5', level=9)
    cA9, cD9, cD8, cD7, cD6, cD5, cD4, cD3, cD2, cD1 = coeffs

    # 阈值去噪
    threshold = (np.median(np.abs(cD1)) / 0.6745) * (np.sqrt(2 * np.log(len(cD1))))
    cD1.fill(0)
    cD2.fill(0)
    for i in range(1, len(coeffs) - 2):
        coeffs[i] = pywt.threshold(coeffs[i], threshold)

    # 小波反变换,获取去噪后的信号
    rdata = pywt.waverec(coeffs=coeffs, wavelet='db5')
    return rdata
```

executed in 19ms, finished 20:36:57 2021-01-23

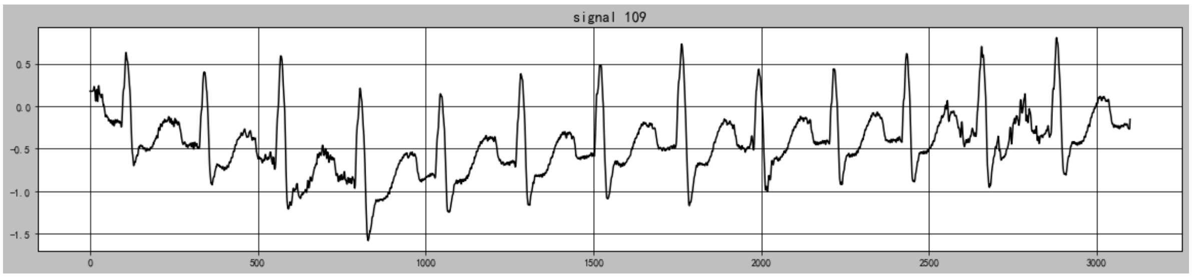
In [104]:

```
# 109 噪声数据-基线漂移数据
record_109 = wfdb.rdrecord('../mit-bih-arrhythmia-database-1.0.0/109',sampfrom=0, sampto=3100,
channel_names=['MLII'])
data_109 = record_109.p_signal.flatten()
plt.figure(figsize=(20, 4))
plt.style.use('grayscale')
plt.plot(list(data_109))
plt.title('signal 109')
```

executed in 242ms, finished 20:20:03 2021-01-23

Out[104]:

```
Text(0.5, 1.0, 'signal 109')
```



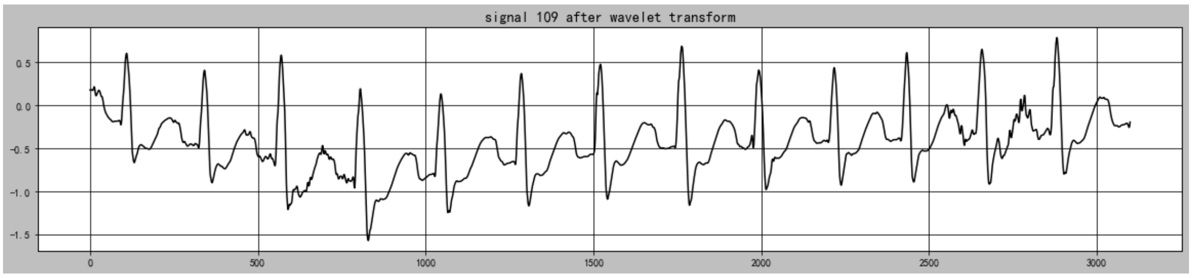
In [132]:

```
rdata_109 = denoise(data_109)
plt.figure(figsize=(20, 4))
plt.plot(list(rdata_109))
plt.title('signal 109 after wavelet transform')
```

executed in 170ms, finished 20:37:00 2021-01-23

Out[132]:

Text(0.5, 1.0, 'signal 109 after wavelet transform')



模型训练集和测试集建立

In [19]:

```
numberSet = ['100', '101', '103', '105', '106', '107', '108', '109', '111', '112', '113', '114', '115',
                         '116', '117', '119', '121', '122', '123', '124', '200', '201', '202', '203', '205', '208',
                         '210', '212', '213', '214', '215', '217', '219', '220', '221', '222', '223', '228', '230',
                         '231', '232', '233', '234']
dataSet = []
lableSet = []
for number in numberSet:
      ecgClassSet = ['N', 'A', 'V', 'L', 'R']

      # 读取心电数据记录
      print("正在读取 " + number + " 号心电数据...")
      record = wfdb.rdrecord('../mit-bih-arrhythmia-database-1.0.0/' + number, channel_names=['MLII'])
      data = record.p_signal.flatten() # flatten是numpy.ndarray.flatten的一个函数，即返回一个一维数组。
      rdata = denoise(data=data) # 小波去噪预处理

      # 获取心电数据记录中R波的位置和对应的标签
      annotation = wfdb.rdann('../mit-bih-arrhythmia-database-1.0.0/' + number, 'atr')
      Rlocation = annotation.sample
      Rclass = annotation.symbol

      # 去掉前后的不稳定数据
      start = 10
      end = 5
      i = start
      j = len(annotation.symbol) - end

      # 因为只选择NAVLR五种心电类型,所以要选出该条记录中所需要的那些带有特定标签的数据,舍弃其余标签的点
      # dataSet在R波前后截取长度为300的数据点
      # lableSet将NAVLR按顺序转换为[0, 1, 2, 3, 4]
      while i < j:
            try:
                  lable = ecgClassSet.index(Rclass[i])
                  x_train = rdata[Rlocation[i] - 99:Rlocation[i] + 201]
                  dataSet.append(x_train)
                  lableSet.append(lable)
                  i += 1
            except ValueError:
                  i += 1
```

executed in 4.40s, finished 17:23:34 2021-01-23

...

In [20]:

```
# 转numpy数组,打乱顺序
dataSet = np.array(dataSet).reshape(-1, 300)   #不知道300个采样点数据一行的有几行 所以用reshape(-1, 300)
lableSet = np.array(lableSet).reshape(-1, 1)
train_ds = np.hstack((dataSet, lableSet))
np.random.shuffle(train_ds) #shuffle() 方法将序列的所有元素随机排序。
```

executed in 530ms, finished 17:23:42 2021-01-23

In [21]:

```
train_ds.shape
```

executed in 18ms, finished 17:23:44 2021-01-23

Out[21]:

```
(92192, 301)
```

In [22]:

```
# NAVLR类别个数统计
c0=c1=c2=c3=c4=0
for i in lableSet:
      if i == 0:
            c0+=1
      elif i == 1:
            c1+=1
      elif i == 2:
            c2+=1
      elif i == 3:
            c3+=1
      elif i == 4:
            c4+=1
print('N|0:',c0)
print('A|1:',c1)
print('V|2:',c2)
print('L|3:',c3)
print('R|4:',c4)
```

executed in 251ms, finished 17:23:48 2021-01-23

```
N|0: 71723
A|1: 1950
V|2: 6974
L|3: 6578
R|4: 4967
```

In [23]:

```
# 数据集及其标签集
X = train_ds[:, :300].reshape(-1, 300, 1) # shape:(92192, 300, 1)
Y = train_ds[:, 300] # shape:(92192,)
```

executed in 110ms, finished 17:23:51 2021-01-23

In [24]:

```
# 测试集及其标签集
shuffle_index = np.random.permutation(len(X))
test_length = int(RATIO * len(shuffle_index))
test_index = shuffle_index[:test_length]
train_index = shuffle_index[test_length:]
X_test, Y_test = X[test_index], Y[test_index]
X_train, Y_train = X[train_index], Y[train_index]
```

executed in 121ms, finished 17:23:53 2021-01-23

In [25]:

```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

executed in 8ms, finished 17:23:55 2021-01-23

```
(64535, 300, 1)
(64535,)
(27657, 300, 1)
(27657,)
```

In [26]:

```
from pylab import *
mpl.rcParams['font.sans-serif'] = ['SimHei'] # 中文显示
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
plt.style.use('ggplot')
```

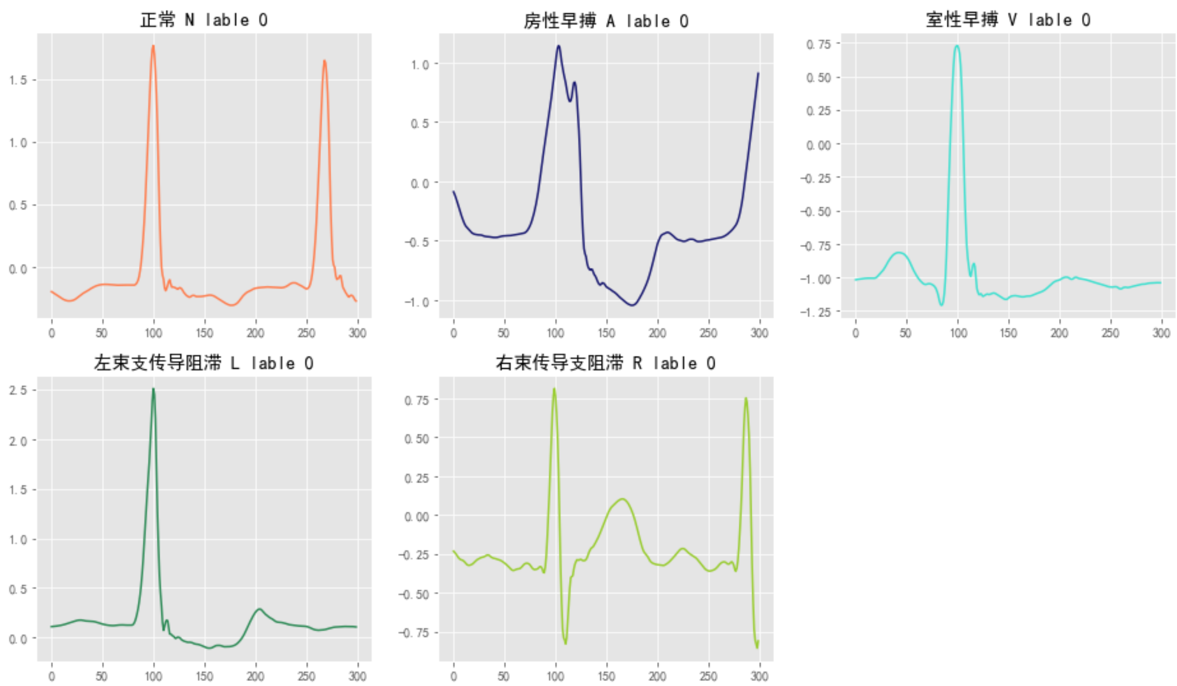
executed in 127ms, finished 17:23:57 2021-01-23

五种心律类别可视化

In [27]:

```
plt.figure(figsize=(16, 14))

plt.subplot(3, 3, 1)
list_X0 = list(X_train[1].flatten())
plt.title('正常 N lable %d'%Y_train[1])
plt.plot(list_X0, color='coral')

plt.subplot(3, 3, 2)
list_X1 = list(X_train[130].flatten())
plt.title('房性早搏 A lable %d'%Y_train[303])
plt.plot(list_X1, color='midnightblue')

plt.subplot(3, 3, 3)
list_X2 = list(X_train[0].flatten())
plt.title('室性早搏 V lable %d'%Y_train[0])
plt.plot(list_X2, color='turquoise')

plt.subplot(3, 3, 4)
list_X3 = list(X_train[27065].flatten())
plt.title('左束支传导阻滞 L lable %d'%Y_train[27065])
plt.plot(list_X3, color='seagreen')

plt.subplot(3, 3, 5)
list_X4 = list(X_train[8].flatten())
plt.title('右束传导支阻滞 R lable %d'%Y_train[8])
plt.plot(list_X4, color='yellowgreen')
plt.show()
```

executed in 691ms, finished 17:23:59 2021-01-23



深度学习Model Build

构建CNN模型

In [157]:

```
import numpy as np
import seaborn as sns
import tensorflow as tf
from sklearn.metrics import confusion_matrix
```

executed in 999ms, finished 21:59:16 2021-01-23

In [139]:

```
# 构建CNN模型
def CNN():
      newModel = tf.keras.models.Sequential([
            tf.keras.layers.InputLayer(input_shape=(300, 1)),
            # 第一个卷积层, 4 个 21x1 卷积核
            tf.keras.layers.Conv1D(filters=4, kernel_size=21, strides=1, padding='SAME',
activation='relu'),
            # 第一个池化层, 最大池化,4 个 3x1 卷积核, 步长为 2
            tf.keras.layers.MaxPool1D(pool_size=3, strides=2, padding='SAME'),
            # 第二个卷积层, 16 个 23x1 卷积核
            tf.keras.layers.Conv1D(filters=16, kernel_size=23, strides=1, padding='SAME',
activation='relu'),
            # 第二个池化层, 最大池化,4 个 3x1 卷积核, 步长为 2
            tf.keras.layers.MaxPool1D(pool_size=3, strides=2, padding='SAME'),
            # 第三个卷积层, 32 个 25x1 卷积核
            tf.keras.layers.Conv1D(filters=32, kernel_size=25, strides=1, padding='SAME',
activation='relu'),
            # 第三个池化层, 平均池化,4 个 3x1 卷积核, 步长为 2
            tf.keras.layers.AvgPool1D(pool_size=3, strides=2, padding='SAME'),
            # 第四个卷积层, 64 个 27x1 卷积核
            tf.keras.layers.Conv1D(filters=64, kernel_size=27, strides=1, padding='SAME',
activation='relu'),
            # 打平层,方便全连接层处理
            tf.keras.layers.Flatten(),
            # 全连接层,128 个节点
            tf.keras.layers.Dense(128, activation='relu'),
            # Dropout层,dropout = 0.2
            tf.keras.layers.Dropout(rate=0.2),
            # 全连接层,5 个节点
            tf.keras.layers.Dense(5, activation='softmax')
     ])
      return newModel
```

executed in 22ms, finished 21:37:48 2021-01-23

In [140]:

```
# 项目目录
project_path = "./"
# 日志目录
log_dir = project_path + "CNN_logs\\" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
# 模型目录
model_path = project_path + "ECG_CNN.h5"

# 构建CNN模型
model = CNN()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy', # 多类的对数损失
              metrics=['accuracy'])
model.summary()
# 定义TensorBoard对象
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
# 训练与验证
model.fit(X_train, Y_train, epochs=30,
          batch_size=128,
          validation_split=RATIO,
          callbacks=[tensorboard_callback])
model.save(filepath=model_path)
```

executed in 12m 19s, finished 21:50:11 2021-01-23

...

In [142]:

```
# 预测
Y_pred = model.predict_classes(X_test)
```

executed in 3.76s, finished 21:50:47 2021-01-23

机器学习分类模型常用评价指标有Accuracy, Precision, Recall和F1-score

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F1 - \text{score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Macro-average方法

- 该方法最简单，直接将不同类别的评估指标（Precision/ Recall/ F1-score）加起来求平均，给所有类别相同的权重。该方法能够平等看待每个类别，但是它的值会受稀有类别影响。

Weighted-average方法

- 该方法给不同类别不同权重（权重根据该类别的真实分布比例确定），每个类别乘权重后再进行相加。该方法考虑了类别不平衡情况，它的值更容易受到常见类（majority class）的影响。

Micro-average方法

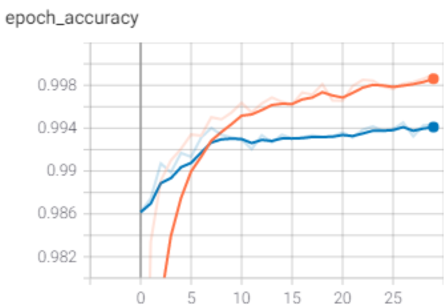
- 该方法把每个类别的TP, FP, FN先相加之后，在根据二分类的公式进行计算。

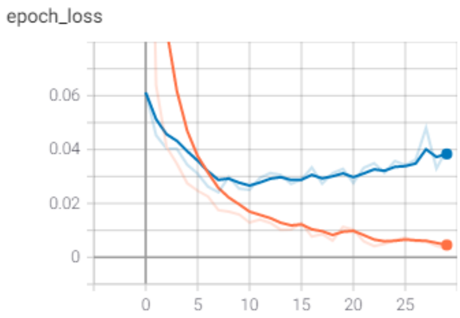
In [309]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

Y_pred.shape
Y_test.shape
# 计算精确度
# correct_prediction = np.equal(Y_pred, Y_test)
# print(np.mean(correct_prediction))    <==> accuracy_score()

print('Accuracy:', accuracy_score(list(Y_pred), list(Y_test)))
# print('Precision:', precision_score(list(Y_pred), list(Y_test), average='weighted'))
# print('Recall:', recall_score(list(Y_pred), list(Y_test), average='weighted'))
# print('F1_score:', f1_score(list(Y_pred), list(Y_test), average='weighted'))

print('------Weighted------')
print('Weighted precision', precision_score(list(Y_pred), list(Y_test), average='weighted'))
print('Weighted recall', recall_score(list(Y_pred), list(Y_test), average='weighted'))
print('Weighted f1-score', f1_score(list(Y_pred), list(Y_test), average='weighted'))
print('------Macro------')
print('Macro precision', precision_score(list(Y_pred), list(Y_test), average='macro'))
print('Macro recall', recall_score(list(Y_pred), list(Y_test), average='macro'))
print('Macro f1-score', f1_score(list(Y_pred), list(Y_test), average='macro'))
print('------Micro------')
print('Micro precision', precision_score(list(Y_pred), list(Y_test), average='micro'))
print('Micro recall', recall_score(list(Y_pred), list(Y_test), average='micro'))
print('Micro f1-score', f1_score(list(Y_pred), list(Y_test), average='micro'))
```

executed in 482ms, finished 17:17:05 2021-01-24

```
Accuracy: 0.9934193874968362
------Weighted------
Weighted precision 0.993660434452819
Weighted recall 0.9934193874968362
Weighted f1-score 0.9934932765602341
------Macro------
Macro precision 0.9692940889562662
Macro recall 0.9891487280946443
Macro f1-score 0.9789097544826614
------Micro------
Micro precision 0.9934193874968362
Micro recall 0.9934193874968362
Micro f1-score 0.9934193874968362
```

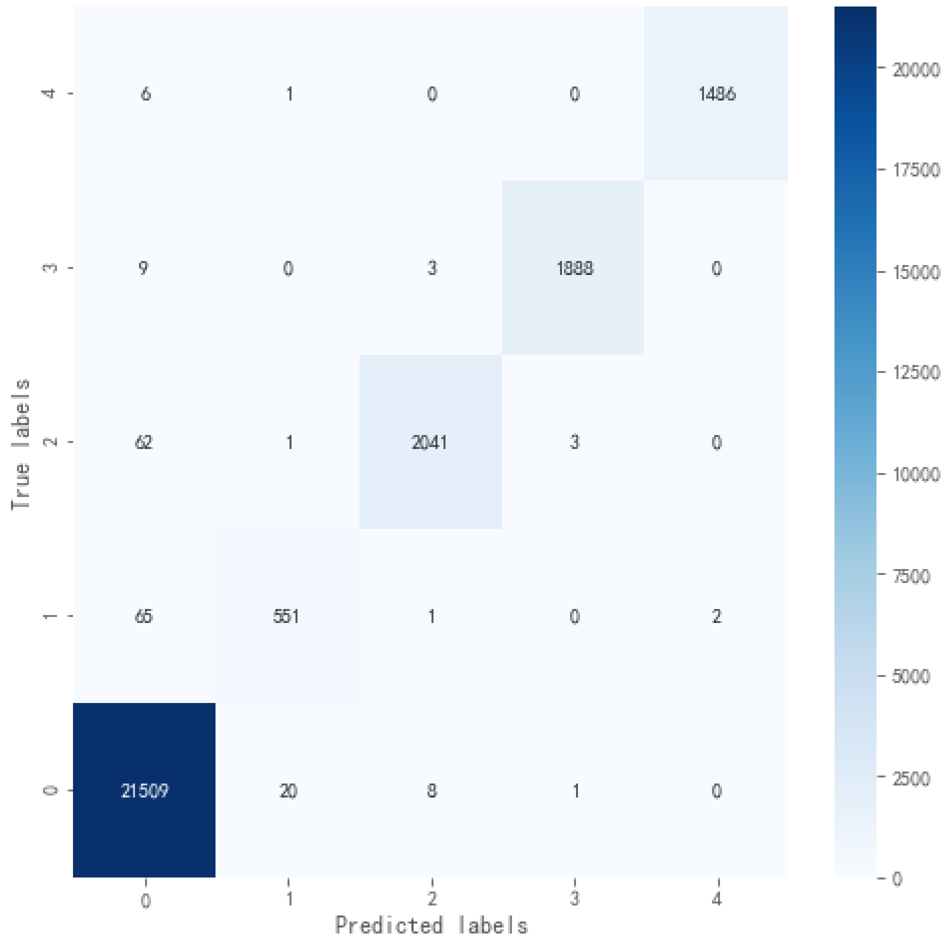



epoch_loss
0.06
0.04
0.02
0
0
5
10
15
20
25

In [301]:

```
# 混淆矩阵
con_mat = confusion_matrix(Y_test, Y_pred)

# 绘图
plt.figure(figsize=(8, 8))
sns.heatmap(con_mat, annot=True, fmt='.20g', cmap='Blues')
plt.ylim(0, 5)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
"""
N|0: 71723
A|1: 1950
V|2: 6974
L|3: 6578
R|4: 4967
"""
```

executed in 331ms, finished 10:03:56 2021-01-24



Out[301]:

```
'\nN|0: 71723\nA|1: 1950\nV|2: 6974\nL|3: 6578\nR|4: 4967\n'
```

构建CNN+LSTM模型

In [201]:

```
def CNN_LSTM():
      newmodel = tf.keras.models.Sequential([
            tf.keras.layers.Conv1D(filters=128, kernel_size=20, strides=3,
padding='same',activation=tf.nn.relu),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.MaxPool1D(pool_size=2, strides=3),
            tf.keras.layers.Conv1D(filters=32, kernel_size=7, strides=1, padding='same',
activation=tf.nn.relu),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.MaxPool1D(pool_size=2, strides=2),
            tf.keras.layers.Conv1D(filters=32, kernel_size=10, strides=1, padding='same',
activation=tf.nn.relu),
            # tf.keras.layers.Conv1D(filters=128, kernel_size=5, strides=2, padding='same',
activation=tf.nn.relu),
            tf.keras.layers.MaxPool1D(pool_size=2, strides=2),
            # tf.keras.layers.Conv1D(filters=512, kernel_size=5, strides=1, padding='same',
activation=tf.nn.relu),
            # tf.keras.layers.Conv1D(filters=128, kernel_size=3, strides=1, padding='same',
activation=tf.nn.relu),
            tf.keras.layers.LSTM(10),
            tf.keras.layers.Flatten(),
            # tf.keras.layers.Dense(units=512, activation=tf.nn.relu),
            tf.keras.layers.Dropout(rate=0.1),
            tf.keras.layers.Dense(units=20, activation=tf.nn.relu),
            tf.keras.layers.Dense(units=10, activation=tf.nn.relu),
            tf.keras.layers.Dense(units=7, activation=tf.nn.softmax)
     ])
      return newmodel
```

executed in 19ms, finished 22:29:28 2021-01-23

In [207]:

```
# 日志目录
log_dir = project_path + "CNN_LSTM_logs\\" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
# 模型目录
model_path = project_path + "ECG_CNN_LSTM.h5"

# 构建CNN_LSTM模型
CNN_LSTM_model = CNN_LSTM()

CNN_LSTM_model.compile(optimizer='adam',
                     loss='sparse_categorical_crossentropy', # 多类的对数损失
                     metrics=['accuracy'])

# CNN_LSTM_model.summary()
# 定义TensorBoard对象
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
# 训练与验证
CNN_LSTM_model.fit(X_train, Y_train, epochs=30,
               batch_size=128,
               validation_split=RATIO,
               callbacks=[tensorboard_callback])
CNN_LSTM_model.save(filepath=model_path)
```

executed in 17m 4s, finished 22:47:40 2021-01-23

...

In [209]:

```
# 预测
CNN_LSTM_Y_pred = CNN_LSTM_model.predict_classes(X_test)
```

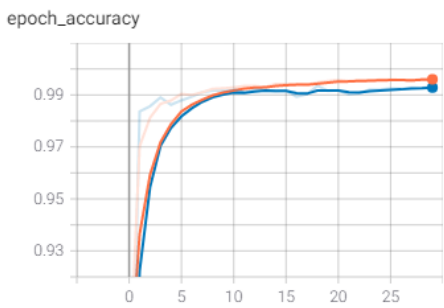
executed in 3.10s, finished 23:34:55 2021-01-23

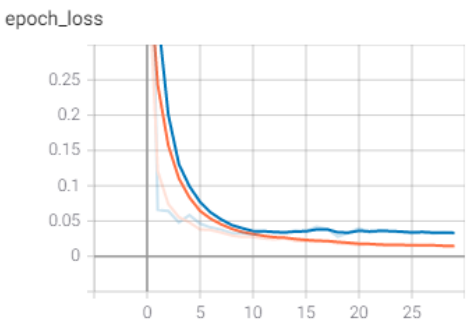
In [310]:

```
print('Accuracy:', accuracy_score(list(CNN_LSTM_Y_pred), list(Y_test)))

print('------Weighted------')
print('Weighted precision', precision_score(list(CNN_LSTM_Y_pred), list(Y_test),
average='weighted'))
print('Weighted recall', recall_score(list(CNN_LSTM_Y_pred), list(Y_test), average='weighted'))
print('Weighted f1-score', f1_score(list(CNN_LSTM_Y_pred), list(Y_test), average='weighted'))
print('------Macro------')
print('Macro precision', precision_score(list(CNN_LSTM_Y_pred), list(Y_test), average='macro'))
print('Macro recall', recall_score(list(CNN_LSTM_Y_pred), list(Y_test), average='macro'))
print('Macro f1-score', f1_score(list(CNN_LSTM_Y_pred), list(Y_test), average='macro'))
print('------Micro------')
print('Micro precision', precision_score(list(CNN_LSTM_Y_pred), list(Y_test), average='micro'))
print('Micro recall', recall_score(list(CNN_LSTM_Y_pred), list(Y_test), average='micro'))
print('Micro f1-score', f1_score(list(CNN_LSTM_Y_pred), list(Y_test), average='micro'))
```

executed in 481ms, finished 17:21:33 2021-01-24

```
Accuracy: 0.9915753697074882
------Weighted------
Weighted precision 0.9923193387653683
Weighted recall 0.9915753697074882
Weighted f1-score 0.9917894548525673
------Macro------
Macro precision 0.9549331419148714
Macro recall 0.989562018732947
Macro f1-score 0.9706355480044432
------Micro------
Micro precision 0.9915753697074882
Micro recall 0.9915753697074882
Micro f1-score 0.9915753697074882
```

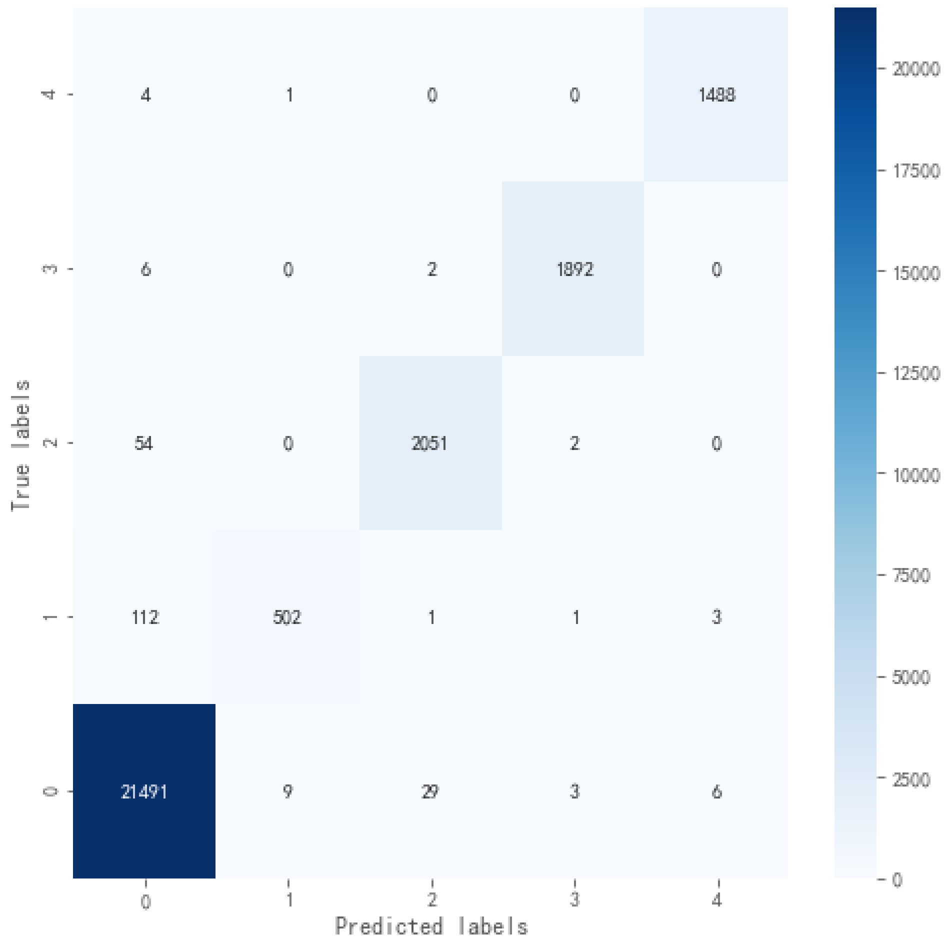




In [297]:

```
# 混淆矩阵
con_mat = confusion_matrix(Y_test, CNN_LSTM_Y_pred)

# 绘图
plt.figure(figsize=(8, 8))
sns.heatmap(con_mat, annot=True, fmt='.20g', cmap='Blues')
plt.ylim(0, 5)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

executed in 909ms, finished 10:03:02 2021-01-24



模型进一步优化比较 TODO

In [ ]: