对固定时间长度ECG信号进行心律失常检测的神经网络算法调研

|  |  |
| --- | --- |
| 学 院： | 电子科学与工程学院 |
| 专 业： | 物联网专业 |
| 学 号： | 06117113 |
| 姓 名： | 吴中行 |
| 指导老师： |  |
| 日 期： |  |
| 成 绩： |  |

目录

[对固定时间长度ECG信号进行心律失常检测的神经网络算法调研 1](#_Toc54176514)

[第 1 章 背景 4](#_Toc54176515)

[1. 1 生理信号检测 4](#_Toc54176516)

[1.1.1 ECG信号产生原理 4](#_Toc54176517)

[1.1.2 ECG信号的特征 6](#_Toc54176518)

[1.1.3 应用场景 6](#_Toc54176519)

[1. 2 数据集 7](#_Toc54176520)

[1. 3 发展趋势 8](#_Toc54176521)

[第 2 章 神经网络算法整理 10](#_Toc54176522)

[2. 1 传统机器学习算法（SVM） 10](#_Toc54176523)

[2.1.1 数据预处理 10](#_Toc54176524)

[2.1.2 SVM算法模型 11](#_Toc54176525)

[2.1.3 性能指标 12](#_Toc54176526)

[2. 2 卷积神经网络CNN 14](#_Toc54176527)

[2.2.1 数据预处理 14](#_Toc54176528)

[2.2.2 卷积神经网络结构 15](#_Toc54176529)

[2.2.3 性能指标 15](#_Toc54176530)

[2. 3 长短期记忆网络LSTM 17](#_Toc54176531)

[2.3.1 数据预处理 18](#_Toc54176532)

[2.3.2 CNN-LSTM网络结构 18](#_Toc54176533)

[2.3.3 性能指标 20](#_Toc54176534)

[2. 4 总结 22](#_Toc54176535)

[第 3 章 神经网络1.0版本分析 24](#_Toc54176536)

[3. 1 网络结构 24](#_Toc54176537)

[3. 2 性能指标分析 26](#_Toc54176538)

[第 4 章 优化思路 27](#_Toc54176539)

[4. 1 网络结构优化 27](#_Toc54176540)

[4.1.1 修改1——GAP代替FC 27](#_Toc54176541)

[4.1.2 修改2——修改填充方式 29](#_Toc54176542)

[4.1.3 修改3——删除bias 30](#_Toc54176543)

[4.1.4 修改4——微调 32](#_Toc54176544)

[4. 2 量化 34](#_Toc54176545)

[4.2.1 背景 34](#_Toc54176546)

[4.2.2 量化概念 34](#_Toc54176547)

[4.2.3 量化目的 34](#_Toc54176548)

[4.2.4 量化方法 35](#_Toc54176549)

[4.2.5 总结 36](#_Toc54176550)

[4. 3 分类器优化LSTM 37](#_Toc54176551)

[附录 40](#_Toc54176552)

[参考文献 40](#_Toc54176553)

[神经网络1.0版本代码 40](#_Toc54176554)

[① config.json 41](#_Toc54176555)

[② data.build.py 42](#_Toc54176556)

[③ dev\_32.json 和 train\_32.json 42](#_Toc54176557)

[④ load.py 42](#_Toc54176558)

[⑤ network.py（核心） 42](#_Toc54176559)

[⑥ train.py（核心） 43](#_Toc54176560)

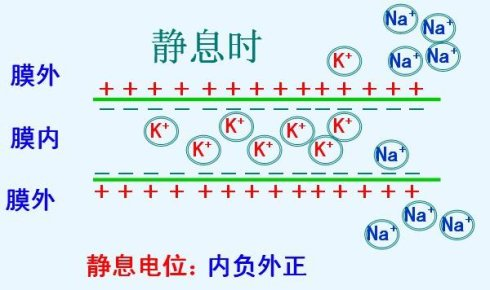
[⑦ util.py 44](#_Toc54176561)

# 背景

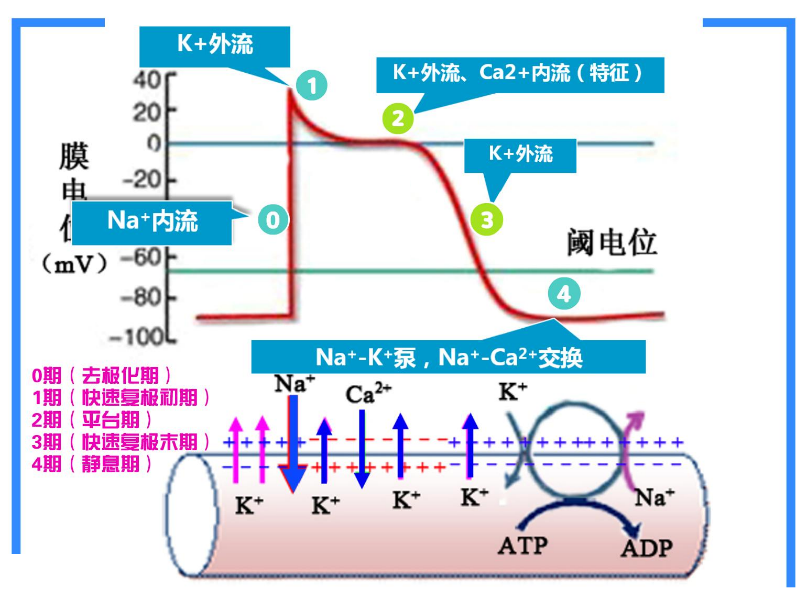
## 生理信号检测

### ECG信号产生原理

ECG信号又称心电信号，是人体心脏的心脏细胞的细胞膜产生的电势差，心房和心室肌在静止的间歇中，由于细胞内外离子(包括K＋，Na＋，Ca2＋，cl-等)浓度差别很大，处于“极化状态”。一旦受到自搏细胞传来的激动，这极化状态便暂时瓦解，在心电图上称为“除极”，由此产生心电活动。心房肌的除极在心电图上表现为P波，心室肌的除极表现为QRS波群。

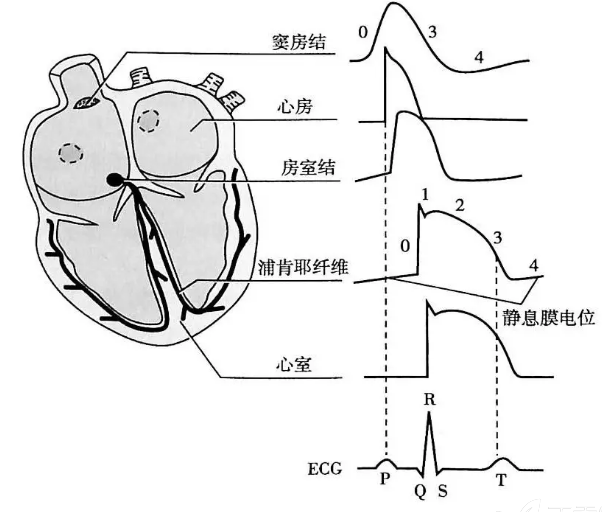
当然在一次除极后，心肌又会恢复原来的极化状态，此过程称为“复极”。复极过程远较除极缓慢，电活动所产生的振幅也较低。心房的复极在P—R段上，一般不明显。心室肌复极则表现为心电图上的ST段及T波。细胞膜电位变化如下图所示。

图表 1 膜内外离子分布

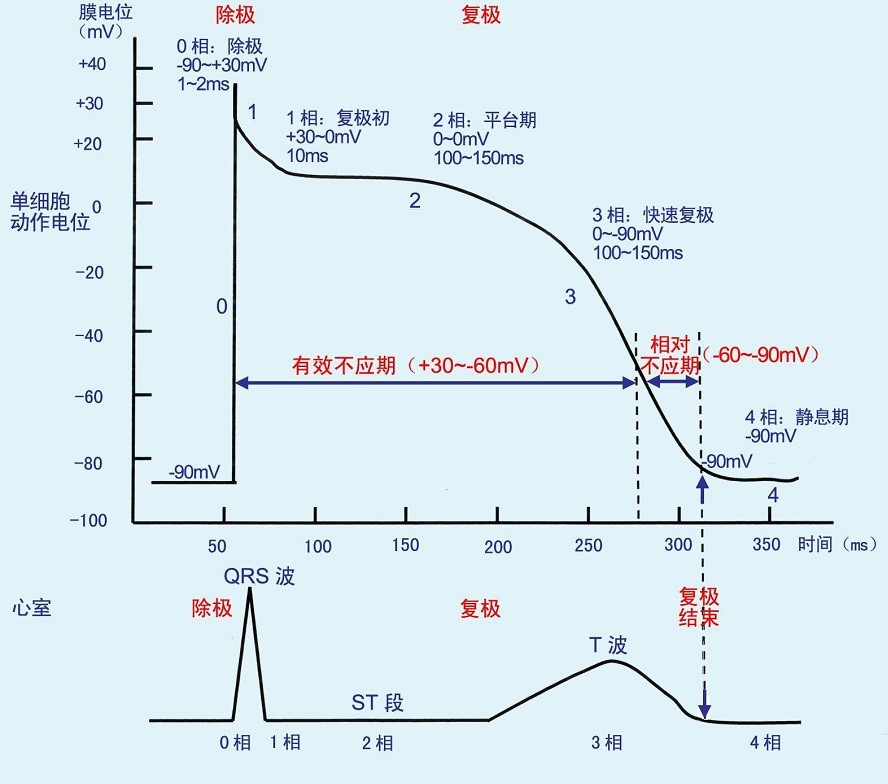
细胞膜激动时产生的电位如下图所示。

图表 2 细胞膜激动电位变化

在除极过程中，Na+内流，K+外流，形成反向电势差，电位突变产生脉冲信号。同时会传导下去，形成电流信号。传导完成后细胞膜复极，缓慢恢复到初始期。

从人体体表采集到各心肌细胞的动作电位叠加后形成如下图所示的心电信号。

图表 3 各心肌细胞的动作电位

**叠加后的信号就是看到做QRS波**，如下图所示。

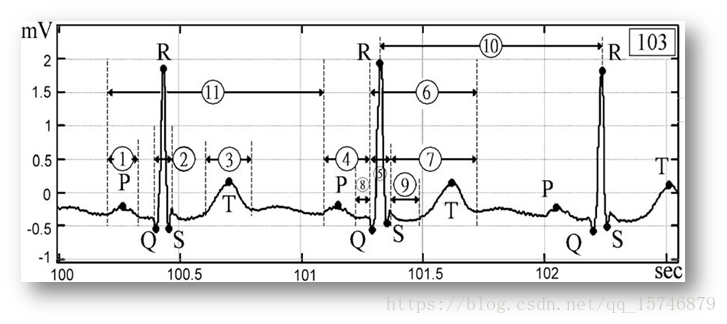
图表 4 叠加后的信号（QRS波）

### ECG信号的特征

生理信号检测是测量心脏电活动的电压与时间的关系（ECG信号）。正常的心电图模式变化发生在许多心脏异常中，包括心脏节律紊乱（例如心房颤动和室性心动过速），冠状动脉血流量不足（例如心肌缺血和心肌梗塞）和电解质紊乱（例如低钾血症和高钾血症）。通过生理信号，可以检测出这些心脏症状。

医生在通过ECG诊断心血管疾病时，其实关注的是各个波形的变化情况，例如，当QRS波变大变宽时，可能发生了室性早搏；ST段抬高时，可能发生了心肌梗死。这样，通过最直观的波形变化，结合医生的经验，可以进行疾病的诊断。

所以，这就引出来了第一类常用的特征：**形态特征**。早期生理信号检测也是手动提取这些形态特征后进行检测。常用的形态特征有：

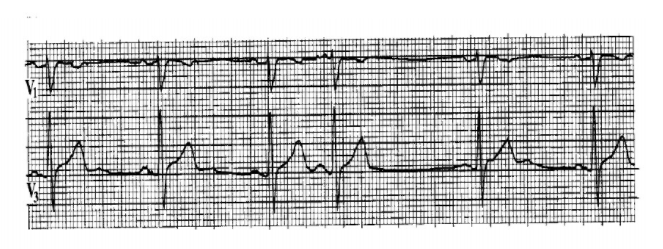
P 波振幅 ②QRS波振幅 ③T 波振幅 ④PR间期 ⑤QRS间期 ⑥QT间期 ⑦ST间期 ⑧PR段水平 ⑨ST段水平 ⑩RR间期，如下图所示：

图表 5 ECG信号形态特征

QRS复合波是心电图上最显眼的地方，也包含大量心脏生理信息。ECG有三个主要成分：P波，代表心房的去极化；QRS复合波，它代表心室的去极化; 和T波，它代表心室的复极化。

### 应用场景

ECG信号是诊断心律失常（或心律失常）最基本、最易获得的方法，因为它是一种无创、易于使用的方法，可以提供有关心脏健康和病理学的有用信息。主要应用于三种场合：

ECG信号可以**精准检测心律失常**。心律失常疾病包括窦性心动过速，房性心动过速，室性心动过速，室上性心动过速，心房扑动，心房颤动，预激综合征，房室传导阻滞等。心律失常是心血管疾病的重要表现。后者是一个严重的社会问题。因为1）心律失常具有很高的发病率和死亡率，往往从发病到死亡时间很短。2） 针对心脏的慢性病治疗费用很高。3）随着全球人口的逐步老龄化，上述问题将加剧，从而可能使死亡人数从2030年的1700万增加到2030年的2400万。

除此之外，ECG信号还用于各种人群体检，普查。专业医师可以通过ECG信号，了解有无严重的先心病，有无心脏结构异常。

图表 6 房性早搏动ECG特征

另外急性心肌缺血，心肌梗死，也可以通过ECG信号来快速识别。

## 数据集

**数据集*MIT-BIH***

数据集获取：

1 https://archive.physionet.org/physiobank/database/html/mitdbdir/records.htm

图片包含 屏幕截图

描述已自动生成2 https://archive.physionet.org/cgi-bin/atm/ATM

**数据集介绍：**

MIT —— Massachusetts Institute of Technology

BIH —— Beth Israel Hospital

自1975年以来，Beth Israel Deaconess和MIT一直支持心律失常分析和相关课题的研究。研究取得的最大的成果之一是MIT-BIH心律失常数据库，我们在1980年完成数据库创建并开始发布。

MIT-BIH 心律失常数据库包含48个30分钟长度双通道动态心电图记录片段，这些片段是从 1975 - 1979年间BIH心律失常实验室的47名研究对象那里获得的。

数据集中包含了17个不同的类别，内容完善，数据量丰富。分别为以下17种类别：'NSR', 'APB', 'AFL', 'AFIB', 'SVTA', 'WPW','PVC', 'Bigeminy', 'Trigeminy', 'VT', 'IVR', 'VFL', 'Fusion', 'LBBBB', 'RBBBB', 'SDHB', 'PR'。

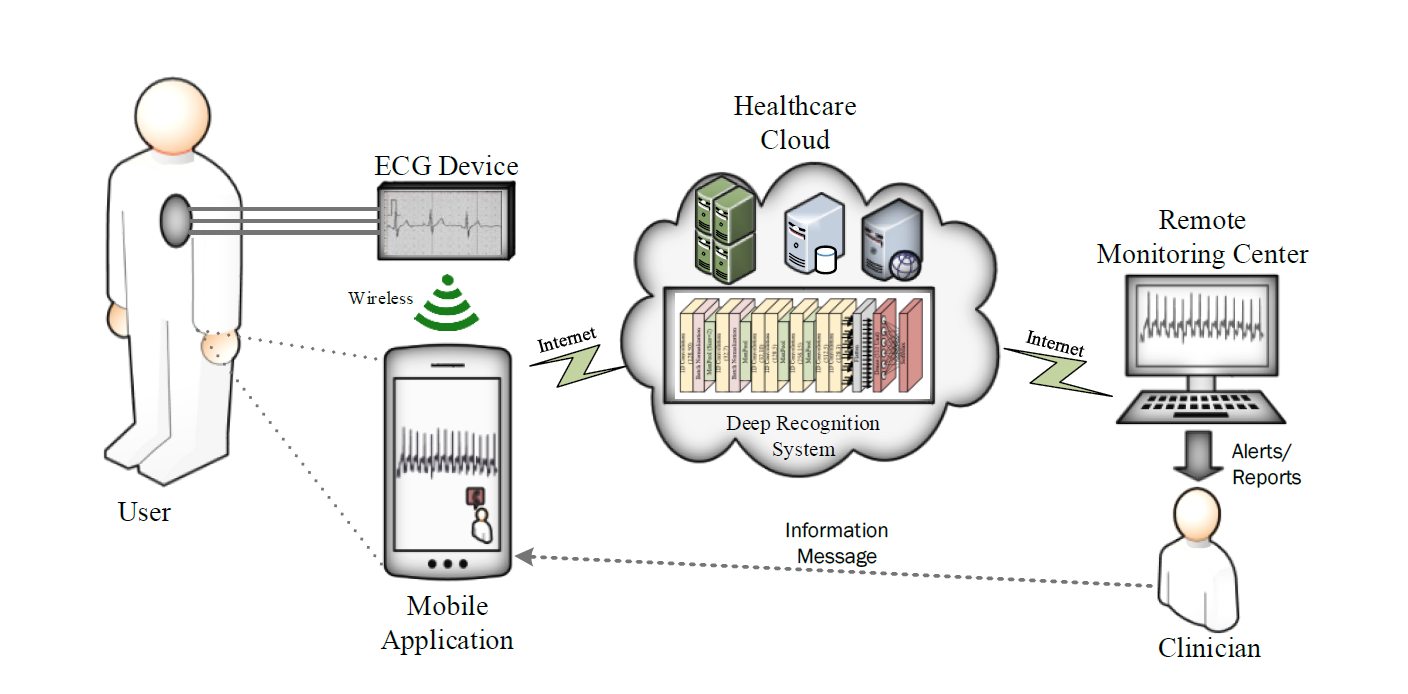
MIT-BIH心率失常数据库也是**绝大多数生理信号检测算法所采用的的数据集**。可以在数据库中采集不同时间长度，例如可采用不同采用频率截取10s时间长度的ECG信号，也可单独截取QRS波部分作为训练样本。

## 发展趋势

近几年，随着人工智能的兴起，“智能医疗”也迅速发展。深度学习算法在生理信号检测领域的优秀表现得到广泛关注。

ECG信号检测的传统方法是QRS波群检测。过去，在心电信号的分析中最为首要和关键的问题是QRS波群的检测，不仅是诊断心律失常的重要依据，而且只有在确定QRS波群后，才有可能计算心律、心律变异性，检测ST段的参数，并进一步地检测和分析心电的其他细节信息，才能对其进行分类等操作。对QRS波群检测方法的研究已有不少学者做了大量工作但他们至今在某方面还不完善，特别是在干扰严重或非典型R波等情况下检测错误率较大。

QRS波群检测是通过手工精心制作一个特征提取加选择的模型。一般是基于一个或者几个QRS波。通过上述分析的几个ECG信号形态特征，编写规则进行检测。最终还是需要结合医生的人工判断才能得到结果，人工成本和错误率也无法降低。由于采集信号时位置、大小的不同和心脏的解剖性质，让心电图即使对同一个正常人也具有高度的可变性；同时，心电信号向体表传导的不同方式也起着重要的影响作用，临床效果往往差。由于ECG表现出来的这些巨大的变化，使得运用这个工具进行诊断具有很强的直觉性和主观性，同时手工编写的规则复杂度高，不利于实时检测和云端部署。因此对ECG模式的解释变得非常困难。

目前深度学习方法在ECG信号检测方面的优势已经很显著，这也是未来ECG信号检测的一个主要发展方向。利用深度学习算法**可以做一个端到端（end to end）的检测模型，信号特征提取，不同信号分类一步到位**。

图表 7 云端部署的深度学习算法模型

基于深度学习的ECG信号检测往往是基于固定时间长度的信号的分析（例如10s）。这**比单个QRS波蕴含了更多的信息**，虽然人工无法提取这些隐藏信息，也不能确定这些信息代表了什么心率特征，但是通过深度学习的方法，可以完整的利用这些信息，并且自动生成一套检测“规则”。从而提升临床诊断的准确率。同时，深度学习算法只用到了简单的MAC操作，减少计算复杂度，使之能**在移动设备和云端上应用**，并具有一定的实时性。

# 神经网络算法整理

## 传统机器学习算法（SVM）

传统的机器学习算法主要有决策树（Decision Trees）、贝叶斯分类（Bayesian classification）、支持向量机（Support Vector Machine，SVM）、逻辑回归(Logistic Regression)等。这些算法都曾运用于ECG信号检测之中。

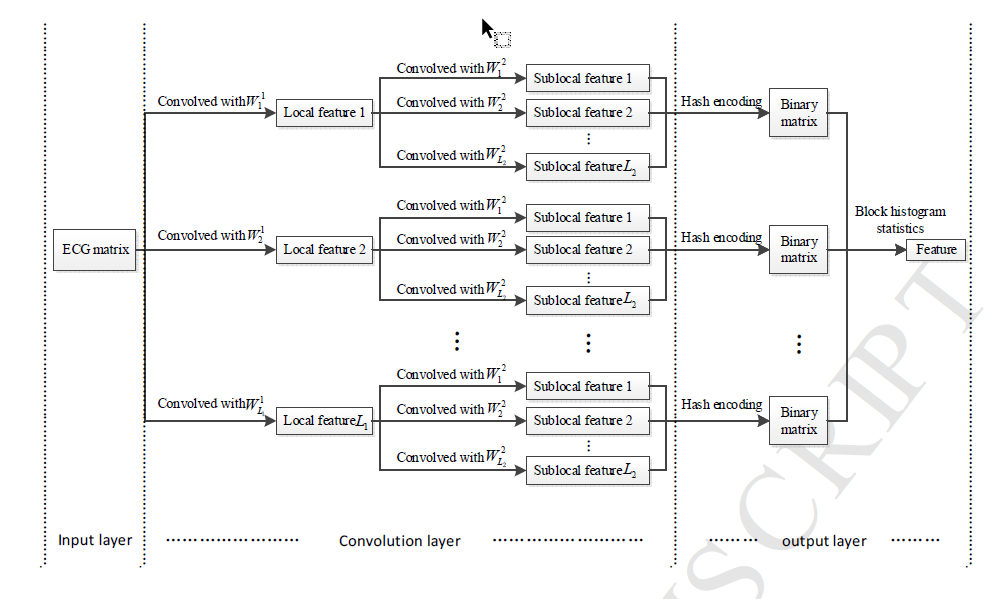
目前，SWM方法在模式识别、回归估计、概率密度函数估计等方面得到了广泛的应用，例如，在模式识别方面，手写字体识别，语音识别，面部检测，文本识别等应用。相比其他机器学习算法，**如今SVM算法在二分类ECG信号分类检测中，准确率依然可靠，可以达到97%，仍有一定应用意义**。

这里主要以论文***《Automatic recognition of arrhythmia based on principal component analysis network and linear support vector machine》***为例，介绍SVM算法，即支持向量机在ECG信号检测中的应用。

论文中提出了一种新的心跳识别方法，目的是应用主成分分析网络（PCANet）对含噪心电信号进行特征提取。为了提高分类速度，将线性支持向量机（SVM）应用于实验中，在MIT-BIH心律失常数据库中识别出五种不平衡原始和无噪声的心电图，以验证我们的算法的有效性，其正确率分别达到97.77%和97.08%，我们的分类方法对心搏倾斜和倾斜的分类结果分别显示了较高的准确率，说明该方法是一种实用的心电信号识别方法，具有一定的噪声鲁棒性和偏态数据适用性。

### 数据预处理

主成分分析网络（PCANet）作为一种深度学习框架，主要用于从样本中提取高维特征向量。

从结构上看，**该网络可以看作是卷积神经网络的一个简化版本**，具有基本的层次和级联结构。已经开发了一些与CNN类似的算法，如PCANet。然而，由于提取的PCA滤波器直接用作PCANet的卷积核，因此不需要像CNN那样通过复杂的迭代过程来计算卷积核。同时，与传统的神经网络模型相比，PCANet只需要调整少量的超参数。这使得PCANet能够更快地处理数据。此外，由于PCA滤波器的去噪功能，PCANet具有显著的噪声鲁棒性和对数据清洁度的要求较低。

图表 8 PCANET

在输入原始ECG信号后，经过PCANet提取高维信息，再输入到线性SVM分类器之中。

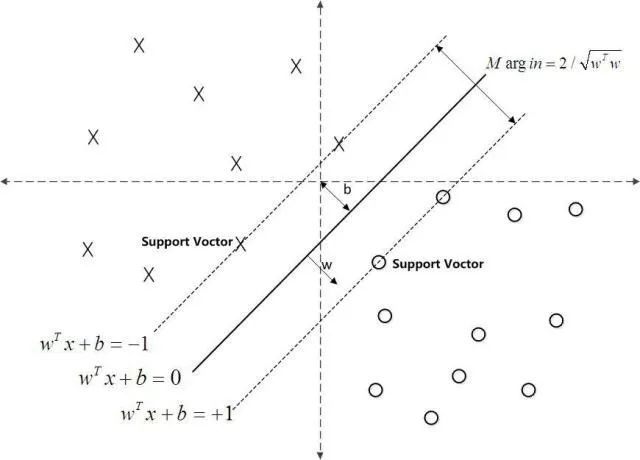
目前，PCANet在深度学习方面显示出了巨大的价值，并在视觉图像识别方面取得了良好的效果。

### SVM算法模型

数据经过PCANet之后，论文使用了四种不同的分类器来根据PCANet特征对这两种类型的心跳进行分类组：线性支持向量机（SVM）、KNN、BP神经网络（BP-NN）和RF方法。这里主要介绍SVM算法。

线性支持向量机分类器是专门为海量数据和特征而实现的。计算出的决策超平面用于将样本分类为不同的类别。误差惩罚因子的选取对线性支持向量机的精度有着重要的影响。在我们的实验中，我们使用线性核函数的支持向量机，参数C设为1。

SVM的模型是让所有点到超平面的距离大于一定的距离，也就是所有的分类点要在各自类别的支持向量两边。

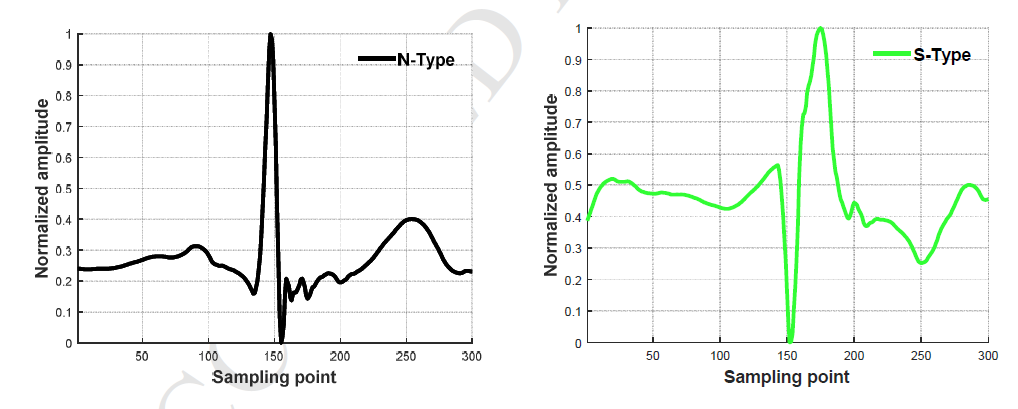
由此得到优化函数：

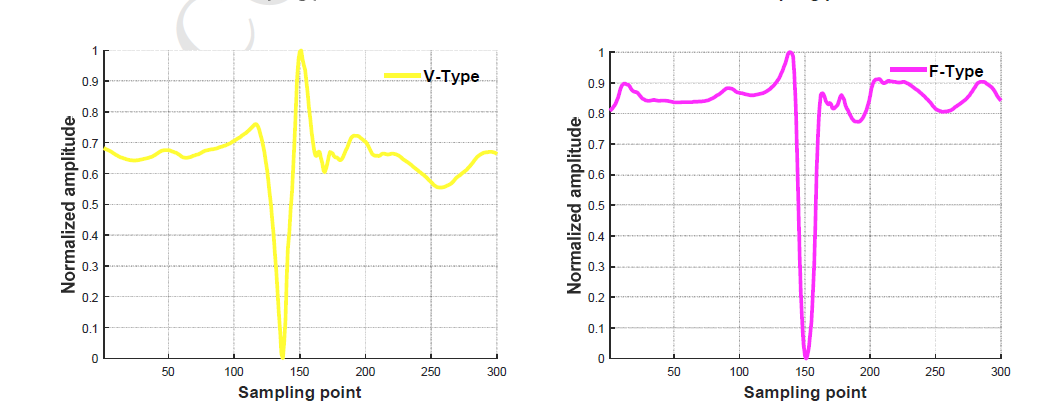
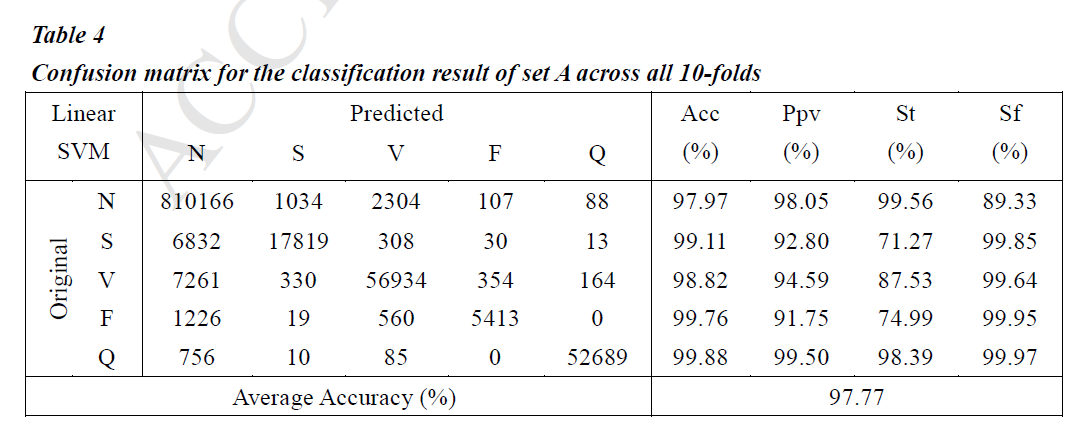
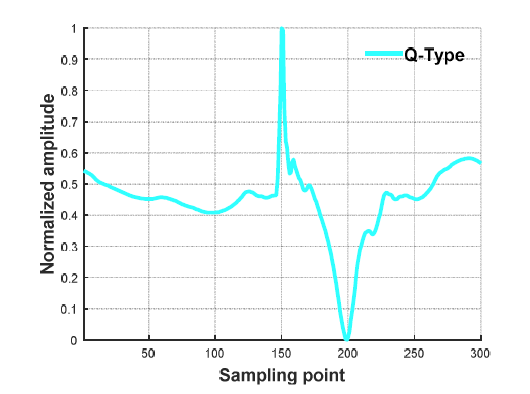
算法流程为：

1. 构造约束优化问题
2. 利用SMO算法求出上式最小时，对应的向量
3. 计算每一个支持向量
4. 得到**分类决策函数**

### 性能指标

论文中使用的所有心跳都来自麻省理工学院波黑心律失常数据库（MIT-BIH）。我们采用了300个采样点，去除了每段ECG记录两端波形的不完整心跳，时长约0.83s的QRS波信号。

分类标准为QRS波的五种形式。N、S、V、F、Q。

为了抑制随机因素对结果的影响，使用10倍交叉验证。表4中的记录显示了10倍采样的总体结果（其准确度的标准偏差分别约为0.0005和0.0022），无论数据量或验证方法如何，我们的实验都是相对可靠的，可以重复进行。

可见，准确率可以达到**97~99%**

## 卷积神经网络CNN

**目前基于CNN的ECG信号检测模型是主流**，这里以论文***《Arrhythmia Detection Using Deep Convolutional Neural Network With Long Duration ECG Signals》***为例，分析这一类算法。

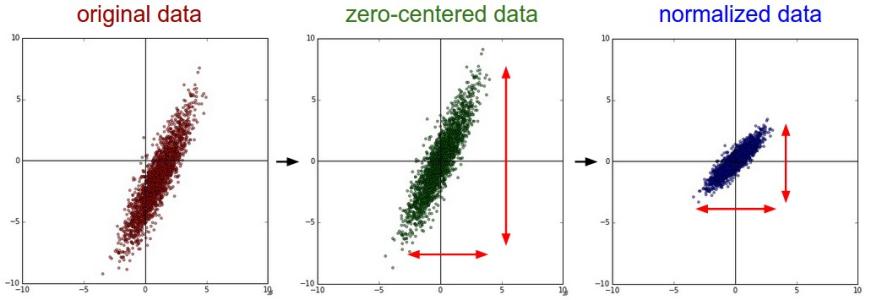
本文提出了一种基于长期心电图（ECG）信号分析的心律失常（17类）检测新的深度学习方法。我们研究的目的是设计一种基于深度学习的新方法，以高效，快速地对心律不齐进行分类。

研究基于MIT-BIH心律失常数据库中的1000条ECG信号片段。应用基于10 s ECG信号片段分析的方法（与SVM方法不同，不是单个QRS复合信号）（平均而言，分类/分析少13倍）。设计了完整的end to end结构，而不是传统方法中使用的手工特征提取和选择。

我们的主要贡献是设计一种新的1D卷积神经网络模型（1D-CNN）。提出的方法是1）高效，2）快速（实时分类）3）非复杂和4）使用简单（组合特征提取和选择以及在一个阶段进行分类）。

Deep 1D-CNN在17个心律不齐症疾病（分类）中的总体识别准确度为91.33％，每个样本的分类时间为0.015秒。与当前的研究相比，我们的结果是迄今为止最好的结果之一，我们的解决方案可以在移动设备和云计算中实现。

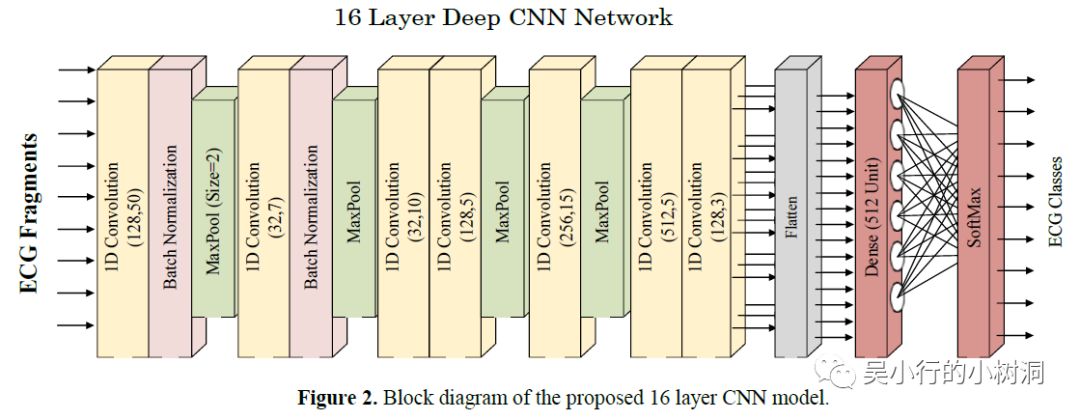
### 数据预处理

再输入模型前，对数据进行了一定的预处理。采用了恒定分量减少和增益降低，并测试了三种类型的归一化：（i）无归一化；（ii）信号重新缩放到[-1,1]（iii）信号**标准化**（信号标准偏差=l，信号平均值=0），实施了重新缩放以获得最佳结果。

图表 9 数据的标准化

### 卷积神经网络结构

根据论文神经网络框架，搭建如下网络：

由7个一维卷积层，4个最大化池，2个标准化层，1个平展层，1个全连接层以及最后的多分类层构成。卷积核填充模式padding=same。

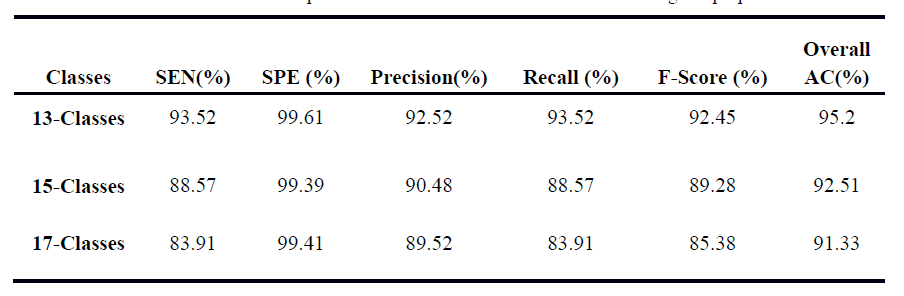
图片包含 文字, 收据

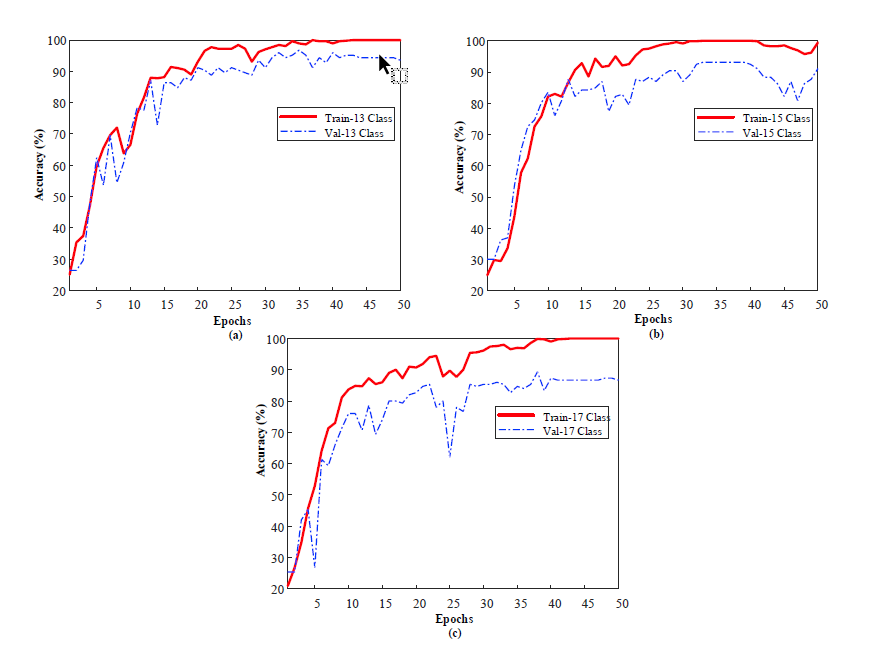
描述已自动生成网络具体参数如下：

### 性能指标

对于ecg片段的自动分类，包含1000个信号片段（每个片段包含3600个样本）的ecg数据集用于优化1-D CNN网络的性能评估，因为该数据集中的某些ecg类样本数很稀疏，除了最初的17类数据集外，还创建了另外两个子数据集，包括15个和13个类]。在所有实验研究中，每个子数据集中的70%、15%和15%分别用于训练、验证和测试阶段。

所提出的1D-CNN网络首先使用13-15和17个类别的训练和验证数据分别进行训练。验证阶段数据用于网络参数调整。然后将训练好的分类网络应用于分配给测试阶段的数据。测试阶段的数据是分类器系统在训练阶段从未见过的数据。图4显示了所提方法的训练和验证性能图

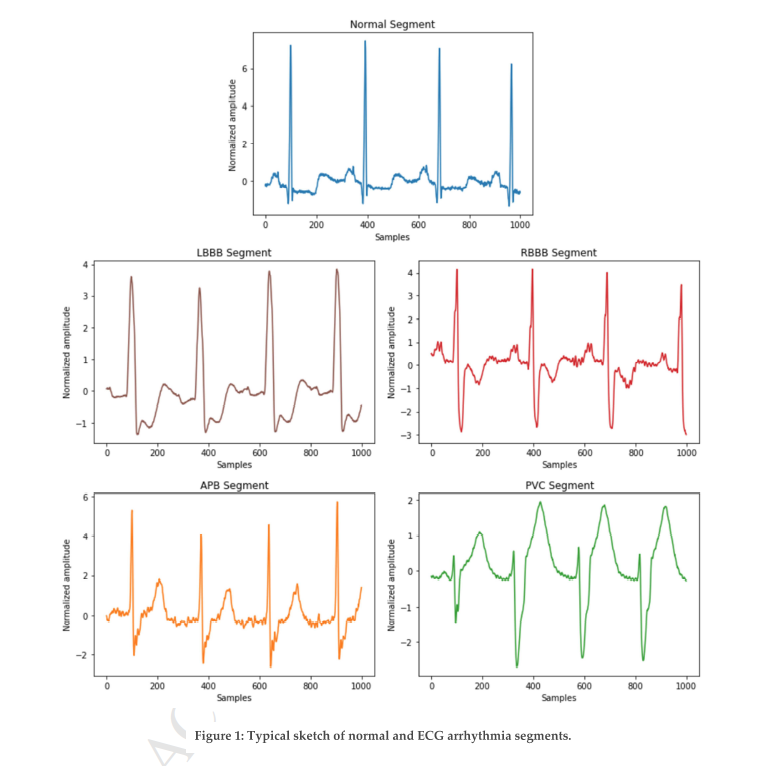
三个组都采用16层CNN网络，50轮。网络性能评价标准分别为：SEN、SPE、Precision、Recall、F-score、AC。

如上图所示，所有病例均获得了较高的准确率。在50轮后，13个类的训练和验证阶段的准确率分别为100%和93.96%，15个类的训练和验证准确率分别为99.41%和91.10%。最后，17个分类获得了100%的训练准确率和86.67%的验证准确率。

## 长短期记忆网络LSTM

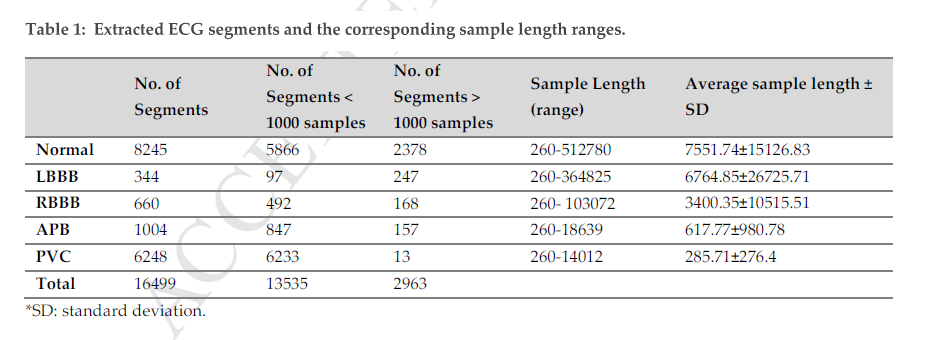
LSTM算法主要也是起分类器的作用，分类之前提取特征信息还是采用的CNN网络。这里主要以论文***《Automated diagnosis of arrhythmia using combination of CNN and LSTM techniques with variable length heart beats》***为例分析。

本文提出了一种结合卷积神经网络（CNN）和长短时记忆（LSTM）诊断正常窦性心律、左束支传导阻滞的自动化系统心电图（LBBB）、右束支传导阻滞（RBBB）、房性早搏（APB）、室性早搏（PVC）五种不同类别。

同时，这项工作的新颖之处在于使用了MIT-BIT心律失常生理数据库中的可变长度的ECG片段。该系统在处理可变长度数据时表现出良好的分类性能，使用十倍交叉验证策略，分类ACC为98.10%，SEN为97.50%，SPE为98.70%。建议临床医生作为常规心电筛查的辅助手段，能够准确地检测到心律失常。

图表 10 五种分类

### 数据预处理

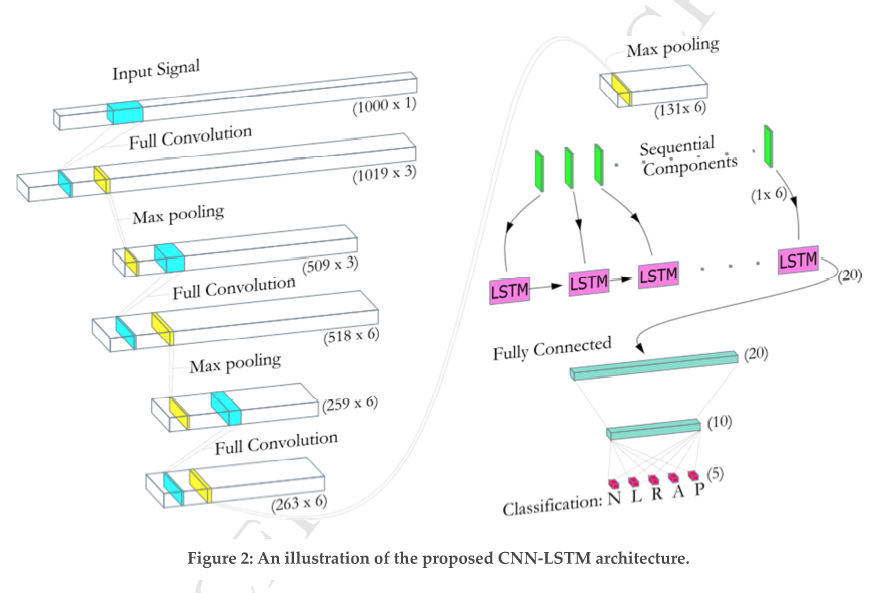
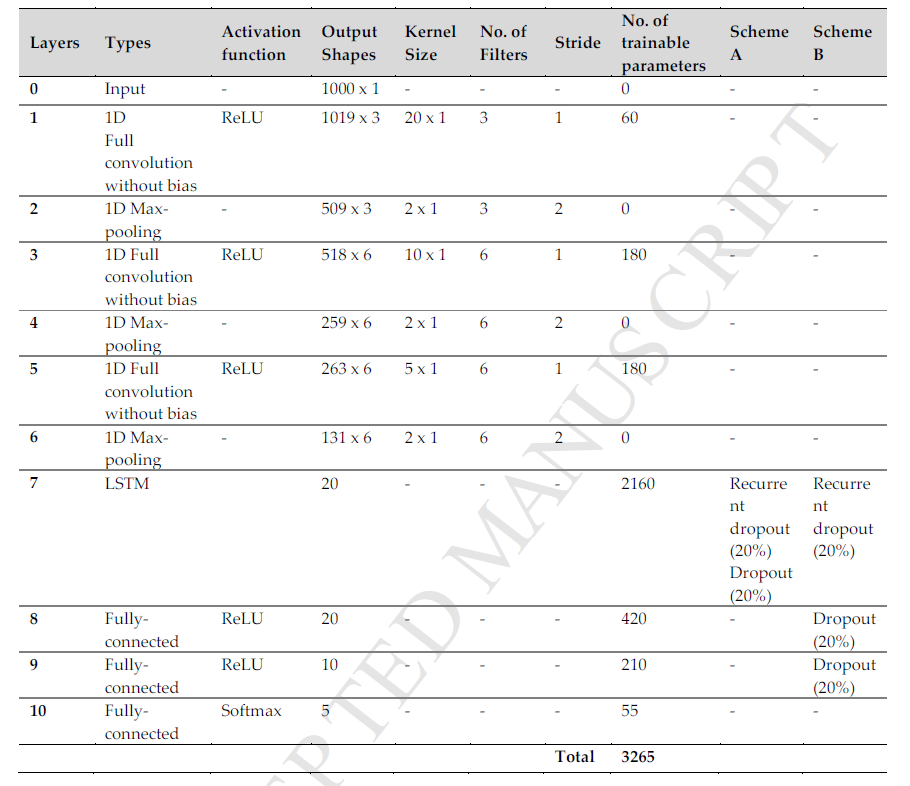
**分割过程是通过将不间断的心律失常节拍序列分配到相应的心律失常组来完成的**。每个节段仅包含一个ECG节拍类型，第一个r峰值左侧有99个采样点，最后一个不间断r峰值右侧有160个采样点。表1显示了从MIT-BIH心律失常数据库获得的ECG段总数，以及相应的采样长度范围。

从表1可以看出，分段之间样本长度的变化有时非常大。为了使模型的输入长度标准化，减少不必要的训练时间，将较长的数据段从末尾任意缩短到1000个样本。少于1000个样本的片段用0填充。这使得后续层的LSTM单元能够以更快的速度进行训练，而不必为较短长度的信号迭代经过填充零的长段。共2963个。

为了进一步加快训练过程，对13535个节段进行了截断和补零，并对输入数据进行了**z-score归一化处理**，消除了偏移效应，使心电信号幅值标准化。原始数据中的值范围各不相同。规范化技术通过将原始数据值压缩到更小的范围来帮助标准化值。这改善了梯度下降时网络中的梯度流，从而有助于加快收敛速度。正常、LBBB、RBBB、APB和PVC信号的典型ECG片段如图10所示。

### CNN-LSTM网络结构

本文提出了一种利用心电信号自动检测五种不同类型心律失常的新技术。心律失常是不稳定和不规则的，可能是单次或多次搏动。因此，建议的网络应能处理长度可变的此类信号。**该网络是CNN与LSTM的混合式深度学习模型**。图2说明了建议的体系结构，表2给出了结构的详细概述。

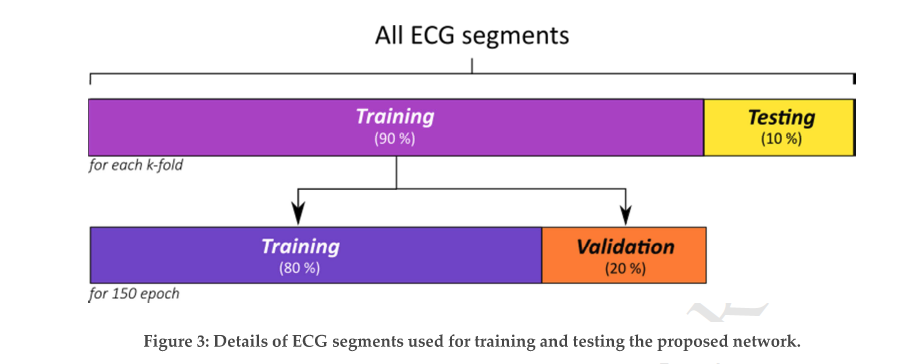
该结构的第1层至第6层是与最大池层耦合的卷积层，而第7层是LSTM层。在网络的最后阶段是一系列完全连接的层，用于预测输出。卷积层可以很好地提取空间特征地图，随后的LSTM层有助于模型捕捉这些特征地图中的特征信息。根据通过完全连接层的最终LSTM元的输出对ECG片段进行分类。

该模型1D-CNN部分由三个具有步长1的卷积层组成。每个卷积运算都是通过在MAC操作来执行的，每次一个样本，在这个过程中，叠加的矩阵被相乘和求和。在训练过程中，网络不断调整卷积核的权值，以提取数据中有意义的空间信息。

本文采用全卷积代替有效卷积。这是因为较短的长度段已经被额外地填充了零，为了保持零填充的完整性，在卷积操作期间不添加任何偏差。因此，来自卷积层的零填充序列的输出仍将被视为零。

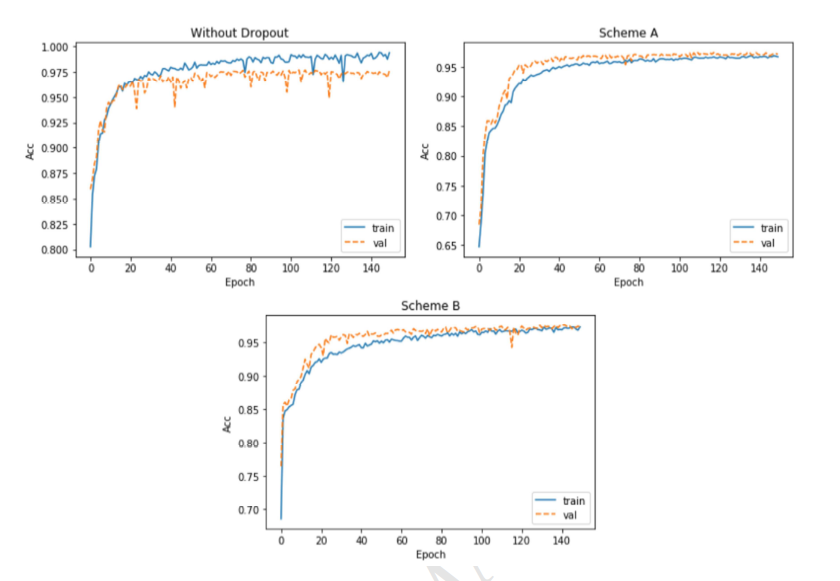
为了将输入表示的大小减半，在每一个卷积层之后对特征图应用大小为2的非重叠步长的最大池化滤波器，然后使用LSTM层从这些特征图中提取时间信息。从卷积和池化过程中提取的特征被分解为顺序分量，并输入到重复的LSTM单元进行时间分析。只有来自LSTM最后一步的输出然后被输入到完全连接层中，用于心律失常预测。最终利用softmax进行多分类。

### 性能指标

本文采用十倍交叉验证策略来评估所提出模型的稳健性。通过将数据分成10等份，对心电图片段进行分层随机抽样。每90个ECG片段用于训练CNN-LSTM网络，其余10个片段用于测试我们所提出系统的性能。该过程迭代10次，每次新初始化的模型都将使用一组新的训练和测试数据进行训练。为了监控训练过程，防止模型过度拟合，在每个训练阶段结束时用20%的k-fold训练数据来验证训练效果。图3显示了每个折叠的数据分布。

在每一次迭代中，使用Xavier算法重新初始化来自网络的权重，然后使用批大小为10的反向传播算法对模型进行端到端的150个时代的训练。将模型的学习速率（n）设为0.001，并与Adam优化器一起使用，以加速网络的学习过程。在更新过程中还应用了加权损失来解决类不平衡问题。每轮性能的评估由ACC、SEN、SPE、PPV组成。

本文中的深度网络是用python语言用Keras开发和评估的以Tensorflow为后端。使用的工作站由两个Intel Xeon 2.40 GHz组成（E5620）处理器和24GB RAM，用于训练网络。

每个训练阶段大约需要138次。12秒完成。共测试了三种不同的方案。第一种方案是一个训练过的没有任何丢失的网络，并作为参考介绍，以检验常规网络和丢失网络之间的影响。另外两个是不同dropout率的方案，其中方案A中LSTM层的循环连接和输入连接中有10%被丢弃，LSTM循环连接的10%被丢弃，随后两个完全连接层的连接丢失了10%。方案B中LSTM层的循环连接和输入连接中有20%被丢弃，LSTM循环连接的20%被丢弃，随后两个完全连接层的连接丢失了20%。每个方案的训练和验证精度曲线如图11所示。

图表 11 训练曲线

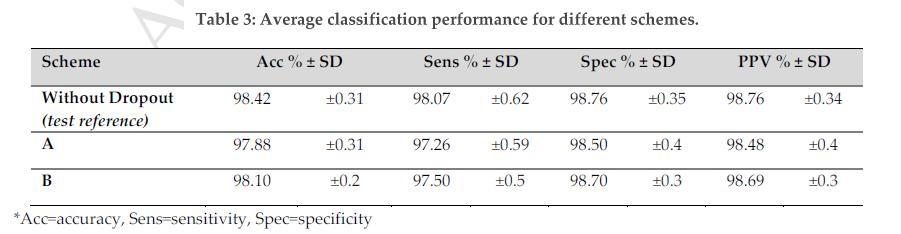
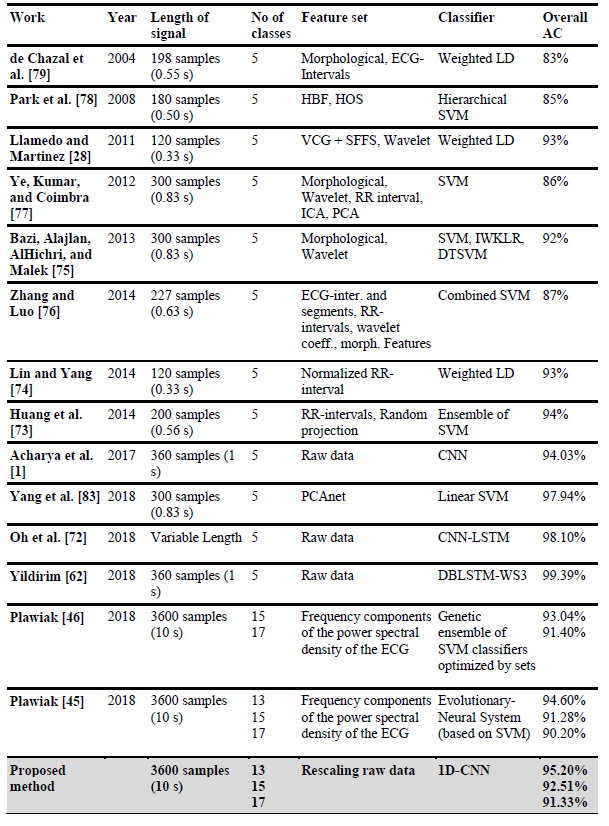
可以观察到，当验证ACC开始趋于平稳，而训练精度继续提高时，没有丢失的方案在epoch 25之后明显过拟合。另一方面，带有dropout网络的准确性相当稳定。验证曲线和训练曲线均在稳定状态下增加，并最终在150个时期结束时达到最高准确率。表3总结了所有三个方案的总体交叉验证性能。

图5和图6分别给出了方案a和方案b的混淆矩阵。结果表明，方案b模型对正常、LBBB、RBBB和APB类别的预测效果较好，而方案A对PVC类别的预测效果较好。这两种模型在APB段的分类中都表现得很差。**在方案A中，1.8%的数据被错误分类，而在方案B中，有1.3%的数据被错误分类。**

## 总结

图表 12 两种方案的混淆矩阵

包括DBN等其他不同方法的表现总结如下图所示：



# 神经网络1.0版本分析

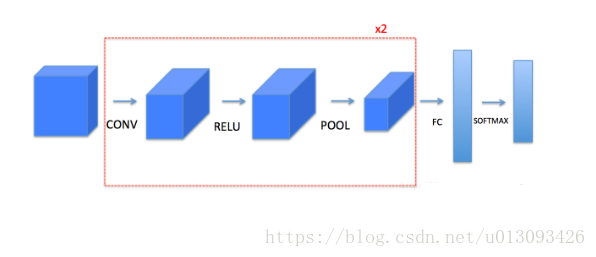
## 网络结构

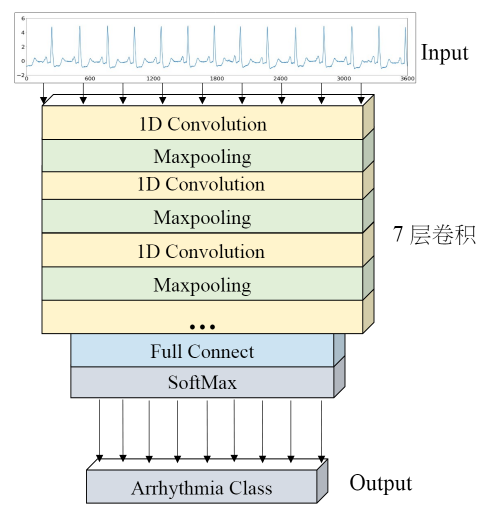
神经网络1.0版本网络结构如下。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num | Layer\_name | channel | size | Stride |
| 1 | Conv1 | 8 | 16 | 2 |
| 2 | Act1 | Rule(max\_value=2) | | |
| 3 | mp1 |  | 8 | 4 |
| 4 | Conv2 | 12 | 12 | 2 |
| 5 | Act2 | Rule(max\_value=2) | | |
| 6 | Mp2 |  | 4 | 2 |
| 7 | Conv3 | 32 | 9 | 1 |
| 8 | Act3 | Rule(max\_value=1) | | |
| 9 | Mp3 |  | 4 | 2 |
| 10 | Conv4 | 64 | 7 | 1 |
| 11 | Act4 | Rule(max\_value=1) | | |
| 12 | Mp4 |  | 4 | 2 |
| 13 | Conv5 | 64 | 5 | 1 |
| 14 | Act5 | Rule(max\_value=1) | | |
| 15 | Mp5 |  | 2 | 2 |
| 16 | Conv6 | 64 | 3 | 1 |
| 17 | Act6 | Rule(max\_value=1) | | |
| 18 | Mp6 |  | 2 | 2 |
| 19 | Conv7 | 72 | 3 | 1 |
| 20 | Act7 | Rule(max\_value=1) | | |
| 21 | Mp7 |  | 2 | 2 |
| 22 | Flatten |  | | |
| 23 | Dropout1 | 0.6 | | |
| 24 | Dense1 |  | | |
| 25 | Act8 | Rule(max\_value=2) | | |
| 26 | Dropout2 | 0.3 | | |
| 27 | Dense2 |  | | |

**主要由两部分组成：黄色的卷积神经网络和橙色的全连接层。前者主要作用是提取信息特征，后者则是作为分类器。**

在构建模型中，总的来说，卷积神经网络与神经网络的区别是增加了若干个卷积层，而卷积层又可细分为卷积（CONV）和池化（POOL）两部分操作，然后是全连接层（FC），可与神经网络的隐藏层相对应；最后是softmax层预测输出值。其中，模型的训练包括前向传播和反向传播等计算过程。

CNN结构如下：

项目中搭建的卷机神经网络共包括七层卷积层，七层池化层，后加全连接层，采用relu和softmax激活层。网络结构图如下。

## 性能指标分析

本项目训练条件为：

主要性能指标为特定数据集（MIT-BIH）中实现的识别准确率，网络规模，计算量等性能指标。如表 10所示。

训练所采用的的超参数如下：

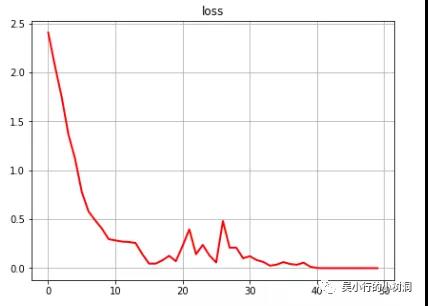
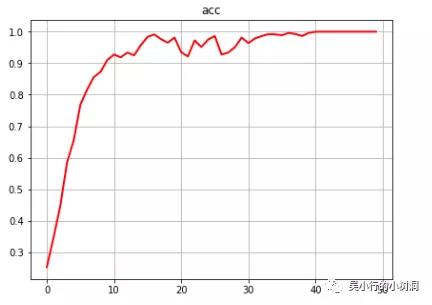
|  |  |
| --- | --- |
| lr | 0.001 |
| patience | 80 |
| batch\_size | 72 |
| epoches | 300 |
| max\_value | 2 |
| 位宽 | （1,2,5） |

图表 13 神经网络1.0版本性能参数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Classes** | **Train\_loss** | **Train\_acc** | **Val\_acc** | **绝对最大中间量**  **absmax** | **卷积层最大参数数量** | **识别时间(s)** |
| 13-Classes | - | - | - | - | - | - |
| 15-Classes | - | - | - | - | - | - |
| 17-Classes-1 | 9.55e-04 | 1 | 0.9374 | 20.71 | 655871 | 0.83 |
| 17-Classes-2 | 0.0085 | 0.9984 | 0.9239 | 19.19 | 655871 | 0.79 |
| 17-Classes-3 | 0.0035 | 0.9988 | 0.9341 | 14.58 | 655871 | 0.91 |
| 17-Classes-4 | 5.75e-04 | 1 | 0.9339 | 21.95 | 655871 | 0.88 |

注：13-Classes和15-Classes的表现需要进一步实验分析。目前只进行了17-Classes的性能评估。

训练过程中，loss以及Acc变化如图11所示。



图表 14 训练中acc & loss变化

# 优化思路

## 网络结构优化

### 修改1——GAP代替FC

**内容**：

将原网络的全连接层(Full Connect)改为池化层(Global average pooling 或者 Global max pooling)。

**原因**：

1. 提升网络性能。部分相关实验结果表明使用池化层代替全连接层对网络性能基本不产生影响甚至于提升识别准确率；
2. 降低功耗。心律失常检测网络v1.0中全连接层参数占总参数的18.5%，使用全局池化层能完全避免这些参数，显著降低加速器访存参数的能耗。
3. 降低加速器设计复杂度。脉动阵列不适合进行全连接层的计算，或需要使用外部cpu进行。

**可行性**：

可行。修改方式简单。

**必要性**：

必要。全连接层对加速器设计影响较大，且修改后对加速器能效比等性能提升较大。

**所做修改：**

修改Block1-6 卷积层的padding模式，same改为valid。改为边界不填充0.

由于用GAP代替FC，最后每一层的特征会被平均，过深的网络会导致最后输入GAP特征过少，我减小了block2卷积层的步长和删除了block7。

**实验结果对比：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 原模型 | 修改1-1 | 修改1-2 | 修改1-3 | 修改1-4 |
| lr | 0.001 | | | | |
| patience | 80 | | | | |
| 0batch\_size | 72 | | | | |
| epoches | 300 | | | | |
| max\_value | 2 | | | | |
| 整数位宽 |  | | | | |
| 是否GAP代替FC | 是 | | | | |
| 是否修改padding | 否 | | | | |
| 是否删除bias | 否 | | | | |
| 修改block2步长 | 否 | 否 | 否 | 是 | 是 |
| 是否删除block7 | 否 | 否 | 是 | 是 | 否 |
| absmax | 13.66 | 8.09 | 12.86 | 11.06 | 11.32 |
| loss | 0.0044 | 6.9454e-4 | 9.711e-4 | 0.0013 | 8.8742e-4 |
| train\_acc | 0.9984 | 1 | 1 | 1 | 1 |
| val\_acc | 0.9276 | 0.9383 | 0.9441 | 0.9243 | 0.9145 |

**结论：**

用GAP代替FC对网络性能影响不大，甚至还有提升。

删除block7，步长为2效果最好。

保存block7，网络深度过深会导致出入GAP特征过少，影响精度。步长为2偏向于学习整体特征。

### 修改2——修改填充方式

**内容**：

将卷积层中的padding从same模式改为valid模式。

**原因**：

降低加速器设计复杂度。Same模式对各层输入的两端添加新的数据，需要在控制模块和InOut Buffer或Input Regfile模块中添加相关控制信号及功能，并修改数据流配置。

**可行性**：

可行。修改方式简单。

**必要性**：

不完全必要。若修改后对网络性能影响较大，则不用进行相关修改。

**所做修改：**

修改Block1-6 卷积层的padding模式，same改为valid。改为边界不填充0.

由于padding=valid，每一层会减小输出形状大小，过深的网络会导致最后输入FC特征过少，我减小了block2卷积层的步长和删除了block7。

**实验结果对比：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 原模型 | 修改2-1 | 修改2-2 | 修改2-3 | 修改2-4 |
| lr | 0.001 | | | | |
| patience | 80 | | | | |
| 0batch\_size | 72 | | | | |
| epoches | 300 | | | | |
| max\_value | 2 | | | | |
| 位宽 | （1,2,5） | | | | |
| GAP or FC | FC | | | | |
| padding | valid | | | | |
| 是否删除bias | 否 | | | | |
| block2步长 | 2 | 2 | 2 | 1 | 1 |
| 是否删除block7 | 否 | 否 | 是 | 是 | 否 |
| absmax | 13.66 | 10.04 | 9.58 | 13.04 | 11.03 |
| loss | 0.0044 | 0.0011 | 8.7929e-4 | 0.0012 | 6.457e-4 |
| train\_acc | 0.9984 | 1 | 1 | 1 | 1 |
| val\_acc | 0.9276 | 0.8783 | 0.8257 | 0.7697 | 0.8421 |

**结论：**

修改padding模式会导致边缘的特征不能利用，对acc影响相对较大。

缩小步长来增加特征，val\_acc和train\_acc相差很多，原因需要进一步分析。

删除block7会导致acc下降，通道数从72到64对于FC层来说影响很大。

微调时，可以尝试将block6的通道数改为72

### 修改3——删除bias

**内容**：

删除卷积层中的偏置(bias)。

**原因**：

降低加速器设计复杂度。心律失常检测网络v1.0中偏置的数量为316，仅占总参数量0.3%；在硬件中设计中需要增加额外模块进行偏置计算并进行数据流配置。

**可行性**：

可行。修改方式简单。

**必要性**：

不完全必要。若修改后对网络性能影响较大，则不用进行相关修改。

**修改：**

根据修改1和修改2，发现不修改FC层时，block2步长为2，不删除block7结果最好。而且这里bias对最后特征的形状没影响。这里就不修改这两项。

**实验结果：**

|  |  |  |
| --- | --- | --- |
|  | 原模型 | 修改3 |
| lr | 0.001 | |
| patience | 80 | |
| 0batch\_size | 72 | |
| epoches | 300 | |
| max\_value | 2 | |
| 整数位宽 |  | |
| GAP or FC | FC | |
| padding | same | |
| 是否删除bias | 是 | |
| block2步长 | 2 | |
| 是否删除block7 | 否 | |
| absmax | 13.66 | 6.7 |
| loss | 0.0044 | 6.1499e-4 |
| train\_acc | 0.9984 | 1 |
| val\_acc | 0.9276 | 0.9212 |

**结论：**

几乎没有影响。

### 修改4——微调

**内容**：

经过三次修改的对比，基本确定GAP代替FC，删除bias，删除block7，block2步长为2。

padding模式分为valid和same做对比。

**实验结果：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 修改  4-1 | 修改  4-2 | 修改  4-3 | 修改  4-3 | 修改  4-5 | 修改  4-6 | 修改  4-7 | 修改  4-8 |
| Lr | 0.001 | | | | | | | |
| Patience | 80 | | | | | | | |
| batch\_size | 72 | 64 | 32 | 16 | 72 | 64 | 32 | 16 |
| epoches | 1000 | | | | | | | |
| max\_value | 2 | | | | | | | |
| 位宽 | （1,2,5） | | | | | | | |
| GAP or FC | GAP | | | | | | | |
| padding | Same | | | | Valid | | | |
| 卷积核大小 | 不改变 | | | | 改变 | | | |
| 是否删除bias | 是 | | | | | | | |
| block2步长 | 2 | | | | | | | |
| 是否删除block7 | 是 | | | | | | | |
| absmax | 19.41 | 13.13 | 18.1 | 26.19 | 22.73 | 17.01 | 20.25 | 30.27 |
| val\_acc | 0.938 | 0.951 | 0.954 | 0.951 | 0.931 | 0.944 | 0.934 | 0.934 |

#### 神经网络2.0网络结构

Padding=same

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num | Layer  name | channel | size | Stride |
| 1 | Conv1 | 8 | 16 | 2 |
| 2 | Act1 | Rule(max\_value=2) | | |
| 3 | mp1 |  | 8 | 4 |
| 4 | Conv2 | 12 | 12 | 2 |
| 5 | Act2 | Rule(max\_value=2) | | |
| 6 | Mp2 |  | 4 | 2 |
| 7 | Conv3 | 32 | 9 | 1 |
| 8 | Act3 | Rule(max\_value=1) | | |
| 9 | Mp3 |  | 4 | 2 |
| 10 | Conv4 | 64 | 7 | 1 |
| 11 | Act4 | Rule(max\_value=1) | | |
| 12 | Mp4 |  | 4 | 2 |
| 13 | Conv5 | 64 | 5 | 1 |
| 14 | Act5 | Rule(max\_value=1) | | |
| 15 | Mp5 |  | 2 | 2 |
| 16 | Conv6 | 72 | 3 | 1 |
| 17 | Act6 | Rule(max\_value=1) | | |
| 18 | Mp6 |  | 2 | 2 |
| 19 | Dropout1 | 0.6 | | |
| 20 | GAP |  | | |
| 21 | Dropout2 | 0.3 | | |
| 22 | Dense2 |  | | |

Padding=valid

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num | Layer  name | channel | size | Stride |
| 1 | Conv1 | 8 | 16 | 2 |
| 2 | Act1 | Rule(max\_value=2) | | |
| 3 | mp1 |  | 7 | 2 |
| 4 | Conv2 | 16 | 12 | 2 |
| 5 | Act2 | Rule(max\_value=2) | | |
| 6 | Mp2 |  | 4 | 2 |
| 7 | Conv3 | 32 | 8 | 1 |
| 8 | Act3 | Rule(max\_value=1) | | |
| 9 | Mp3 |  | 5 | 2 |
| 10 | Conv4 | 64 | 8 | 1 |
| 11 | Act4 | Rule(max\_value=1) | | |
| 12 | Mp4 |  | 4 | 2 |
| 13 | Conv5 | 64 | 4 | 1 |
| 14 | Act5 | Rule(max\_value=1) | | |
| 15 | Mp5 |  | 2 | 2 |
| 16 | Conv6 | 72 | 3 | 1 |
| 17 | Act6 | Rule(max\_value=1) | | |
| 18 | Mp6 |  | 2 | 2 |
| 19 | Dropout1 | 0.6 | | |
| 20 | GAP |  | | |
| 21 | Dropout2 | 0.3 | | |
| 22 | Dense2 |  | | |

**结论：**

实验结果说明padding=same时，batch\_size=32时，acc最高，可以达到0.954，padding=valid时，batch\_size=64时，acc最高0.944

基本可以达到预期性能。

## 量化

对神经网络的权值进行量化，使网络模型大小变小，运行速度更快，且准确率与原来相近。

### 背景

Neural Network模型一般都会占用很大的磁盘空间，比如AlexNet的模型文件就超过了200 MB。型包含了数百万的参数，绝大部分的空间都用来存储这些模型的参数了。这些参数是浮点数类型的，普通的压缩算法很难压缩它们的空间。

一般模型的内部的计算都采用了浮点数计算，浮点数的计算会消耗比较大的计算资源（空间和cpu/gpu时间）,如果在不影响模型准确率的情况下，模型内部可以采用其他简单数值类型进行计算的话，计算速度会提高很多，消耗的计算资源会大大减小，尤其是对于移动设备来说，这点尤其重要。

### 量化概念

简单来说，就是把网络权值从高精度转化成低精度（**32位浮点数 float32 转化成 8位定点数 int8** 或**二值化为 1 bit**），但模型准确率等指标与原来相近，模型大小变小，运行速度加快。

### 量化目的

量化可以看作是噪声的一种来源，所以量化后的模型效果与原来相近。

优点

* 模型变小，运行速度变快。
* int8 只需 float32 内存带宽的25％，可以更好使用缓存并避免 RAM 访问出现瓶颈。
* 每个时钟周期执行更多的 SIMD 操作。
* 如有加速8位计算的 DSP 芯片则更快。

缺点

* 模型准确率稍差。

### 量化方法

先训练模型，再进行量化，测试时使用量化后的模型。

• 训练

一般使用 float32 来训练模型效果较好（特别是反向传播和梯度需要浮点来表示）

• 量化

①加入量化和去量化操作（如一种量化操作为根据该层权值的最大值和最小值映射到 8位区间）

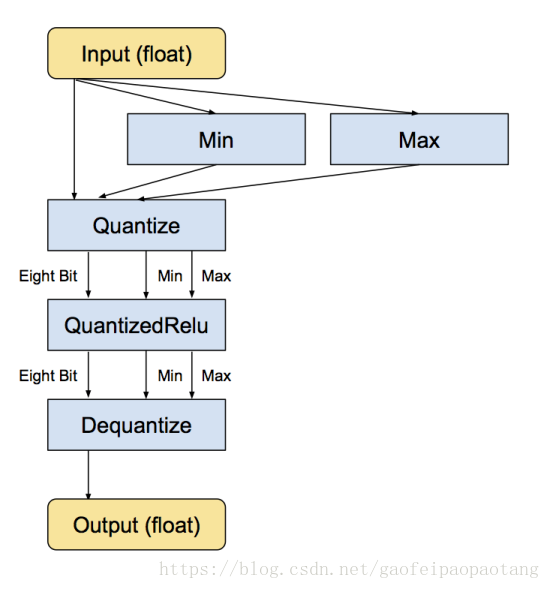
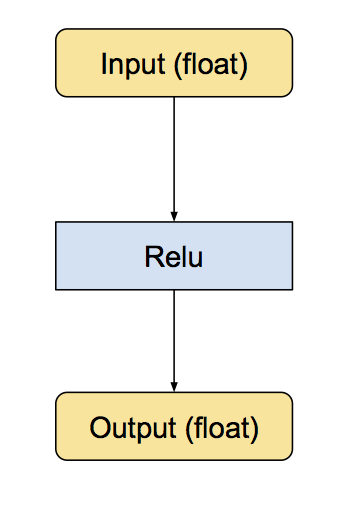
如下图 1 变成图 2

图 1

图 2

②把相应的运算转化为量化的运算（实现 8位版本的卷积、矩阵乘法等）

③删除相邻的 去量化-量化 操作

如下图3

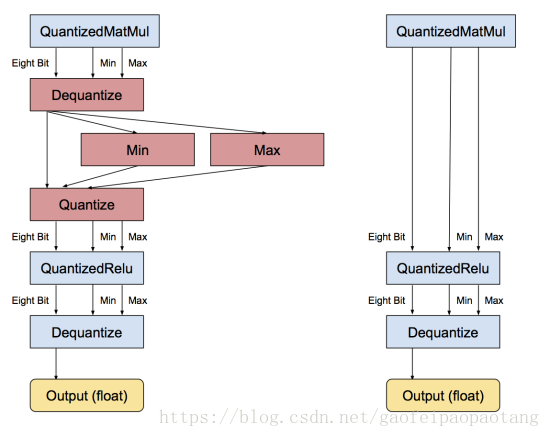
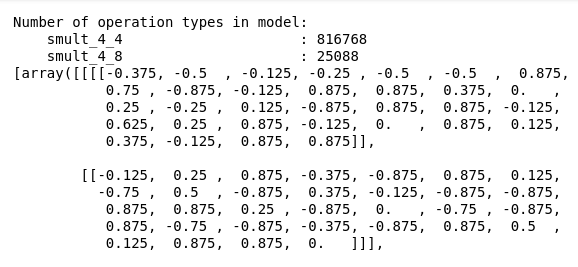
• 测试

图 3

使用量化后的模型来预测

### 总结

**量化理论有效的解释：**神经网络被过度参数化，进而包含足够的冗余信息，裁剪这些冗余信息不会导致明显的准确度下降。

**相关证据：**对于给定的量化方法，FP32 网络和 INT8 网络之间的准确度差距对于大型网络来说较小，因为大型网络过度参数化的程度更高。

图表 15 参数量化结果

量化训练所采用的的超参数如下：

|  |  |
| --- | --- |
| lr | 0.001 |
| patience | 80 |
| batch\_size | 72 |
| epoches | 300 |
| max\_value | 2 |
| 位宽 | （1,2,5） |

图表 13 量化对比结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Classes | Train\_loss | Train\_acc | Val\_acc | 绝对最大参数  absmax | 卷积层最大参数数量 | 识别时间(s) |
| 17-Classes-1 | 9.55e-04 | 1 | 0.9374 | 12.71 | 655871 | 0.83 |
| 17-Classes-2 | 5.75e-04 | 1 | 0.9239 | 15.19 | 655871 | 0.88 |
| Q-17-Classes-1 | 0.0015 | 0.9988 | 0.9141 | 19 | 655871 | 0.94 |
| Q-17-Classes-2 | 9.81e-04 | 1 | 0.8939 | 21 | 655871 | 1.03 |

## 分类器优化LSTM

**内容**：

将原网络的全连接层(Full Connect)与卷积层（CNN）之间插入LSTM单元，并减少卷积层层数。

**原因**：

1. 提升网络性能。部分相关实验结果表明功耗主要由参数访问和卷积计算产生，通过使用LSTM单元替换部分CNN网络，来达到降低功耗的效果；
2. 降低加速器设计复杂度。近似乘法单元设计较为复杂，MAC操作多，使用LSTM可以避免过多的MAC操作，降低设计复杂度。
3. 减少了模型深度，提升预测效率。

**可行性**：

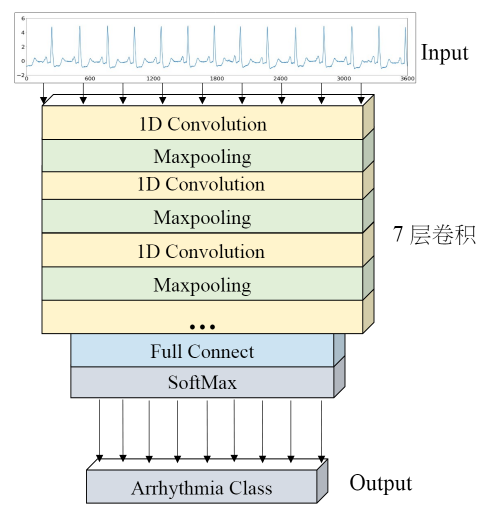
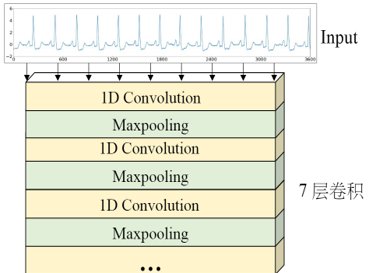
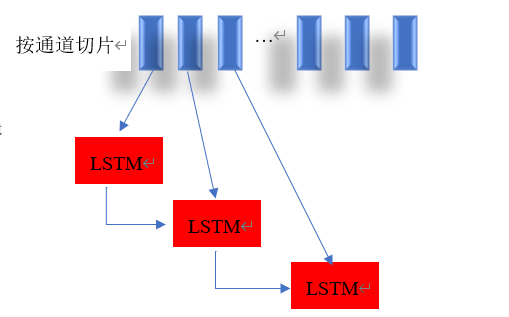
可行。修改方式简单。

**必要性**：

必要。卷积层对功耗影响较大。

**所做修改：**

去除了最后四层卷积层，插入了LSTM单元，进一步快速分类CNN网络所提取的特征，最后输入到全连接层进行多分类。



3层卷积



…

…

分类结果

全连接层

具体参数如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num | Layer\_name | channel | size | Stride |
| 1 | Conv1 | 8 | 16 | 2 |
| 2 | Act1 | Rule(max\_value=2) | | |
| 3 | mp1 |  | 8 | 4 |
| 4 | Conv2 | 12 | 12 | 2 |
| 5 | Act2 | Rule(max\_value=2) | | |
| 6 | Mp2 |  | 4 | 2 |
| 7 | Conv3 | 32 | 9 | 1 |
| 8 | Act3 | Rule(max\_value=1) | | |
| 9 | Mp3 |  | 4 | 2 |
| 10 | LSTM |  | | |
| 11 | Dropout1 | 0.5 | | |
| 12 | Dense1 |  | | |
| 13 | Act8 | Rule(max\_value=2) | | |
| 14 | Dropout2 | 0.3 | | |
| 15 | Dense2 |  | | |

**实验结果对比：**

|  |  |  |
| --- | --- | --- |
|  | 原模型 | 修改 |
| lr | 0.001 | |
| patience | 80 | |
| 0batch\_size | 72 | |
| epoches | 300 | |
| max\_value | 2 | |
| 整数位宽 | 2 | |
| GAP or FC | FC | |
| padding | same | |
| 是否删除bias | 是 | |
| absmax | 13.66 |  |
| loss | 0.0044 |  |
| train\_acc | 0.9984 |  |
| val\_acc | 0.9276 |  |

**结论：**

# 附录

### 参考文献

1. Paweł Pławiak et. ***Arrhythmia Detection Using Deep Convolutional Neural Network With Long Duration ECG Signals***
2. Yang, W., Si, Y., Wang, D., & Guo, B. (2018). ***Automatic recognition of arrhythmia based on principal component analysis network and linear support vector machine***. Computers in Biology and Medicine
3. S. L. Oh, E. Y. Ng, R. S. Tan, & U. R. Acharya, “***Automated diagnosis of arrhythmia using combination of CNN and LSTM techniques with variable length heart beats***”, Computers in Biology and Medicine, 2018.
4. ***Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network***
5. ***A novel application of deep learning for single-lead ECG classification***
6. Qiao Li and Cadathur Rajagopalan and Gari D. Clifford; ***A machine learning approach to multi-level ECG signal quality classification; Computer Methods and Programs in Biomedicine***; 2014; 117; 435 – 447

### 神经网络1.0版本代码

**程序代码如下：**

**def** build\_new\_network(inputs):  
  
 *# block 1* layer = Conv1D(8, 16, strides=2, padding=**'same'**, activation=**'relu'**, use\_bias=**True**)(inputs)  
 layer = MaxPooling1D(pool\_size=8, strides=4)(layer)  
  
 *# block 2\_plus* layer = Conv1D(12, 12, strides=2, padding=**'same'**, activation=**'relu'**)(layer)  
 layer = MaxPooling1D(pool\_size=4, strides=2)(layer)  
 *# block 2* layer = Conv1D(32, 9, strides=1, padding=**'same'**, activation=**'relu'**)(layer)  
 layer = MaxPooling1D(pool\_size=4, strides=2)(layer)  
 *# block 3* layer = Conv1D(64, 7, strides=1, padding=**'same'**, activation=**'relu'**)(layer)  
 layer = MaxPooling1D(pool\_size=4, strides=2)(layer)  
 *# block4* layer = Conv1D(64, 5, strides=1, padding=**'same'**, activation=**'relu'**)(layer)  
 layer = MaxPooling1D(pool\_size=2, strides=2)(layer)  
 *# block5* layer = Conv1D(64, 3, strides=1, padding=**'same'**, activation=**'relu'**)(layer)  
 layer = MaxPooling1D(pool\_size=2, strides=2)(layer)  
 *# block6* layer = Conv1D(72, 3, strides=1, padding=**'same'**, activation=**'relu'**)(layer)  
 layer = MaxPooling1D(pool\_size=2, strides=2)(layer)  
  
 **return** layer  
  
  
**def** add\_fc\_layers(layer, \*\*params):  
 layer = Flatten()(layer)  
 layer = Dropout(0.6)(layer)  
 layer = Dense(params[**"hidden\_node"**], activation=**'relu'**)(layer)  
 **if** params.get(**"dropout"**, 0.1):  
 layer = Dropout(params[**"dropout"**])(layer)  
 layer = Dense(params[**"num\_categories"**], activation=**'softmax'**)(layer)  
 **return** layer

### ① config.json

本项目文件下配置了网络的基本参数，包括隐藏层节点数、学习率、批尺寸、epoch、数据集和测试集文件路径等等初始参数。代码如下

{  
  
 **"hidden\_node"**:64,  
  
 **"learning\_rate"**: 0.001,  
 **"batch\_size"**: 72,  
 **"MAX\_EPOCHS"**:1000,  
  
 **"train"**: **"./train\_32.json"**,  
 **"dev"**: **"./dev\_32.json"**,  
 **"shuffle"**: **true**,  
 **"generator"**: **false**,  
 **"dropout"**: 0.3,  
 **"fc"**: **false**,  
  
 **"save\_dir"**: **"saved"**,  
 **"log\_dir"**: **"logs"**}

### ② data.build.py

本项目文件主要在于预处理数据并保存，搭建神经网络各个步骤中，个人认为最重要最关紧的步骤就在于数据集的选择以及数据预处理，它深刻影响到后续网络训练的结果。

### ③ dev\_32.json 和 train\_32.json

dev\_32.json为测试数据集，保存了测试数据的地址及标签。同理，train\_32.json为训练数据集，保存了训练数据的地址及标签。

### ④ load.py

本项目文件旨在导入训练数据集方法实现。

### ⑤ network.py（核心）

顾名思义，核心卷积网络模型搭建模块，具体实现方法网络结构处已说明，不再陈述。需要注意的是keras框架中模型的导入、优化器、损失函数的选择。

程序如下：

**from** keras.optimizers **import** Adam  
**from** keras.models **import** Model  
**from** keras.layers **import** Conv1D, MaxPooling1D, BatchNormalization  
**from** keras.layers **import** Dense, Flatten, Dropout, Input

上述代码将模型及网络层必要文件包导入。

**def** add\_compile(model, \*\*params):  
 optimizer = Adam(lr=params[**"learning\_rate"**])  
 model.compile(loss=**'categorical\_crossentropy'**,  
 optimizer=optimizer,  
 metrics=[**'accuracy'**])  
  
  
**def** build\_network(\*\*params):  
 inputs = Input(shape=params[**'input\_shape'**],  
 dtype=**'float32'**,  
 name=**'inputs'**)  
 layer = build\_new\_network(inputs)  
 output = add\_fc\_layers(layer, \*\*params)  
  
 model = Model(inputs=[inputs], outputs=[output])  
 add\_compile(model, \*\*params)  
 **return** model

以上代码注意keras中运用compile（）函数实现模型优化及损失函数的编写，包括在这里可以进行“accuracy”指标配置。

### ⑥ train.py（核心）

本项目文件是整个网络训练的主要切入口，主要实现训练集、验证集及测试集的导入，网络模型的保存，还有学习率的动态设置。这里需要注意的是数据集导入的方式以及网络模型训练后的保存，程序如下：

model.fit(  
 train\_x, train\_y,  
 batch\_size=batch\_size,  
 epochs=params[**"MAX\_EPOCHS"**],  
 validation\_data=(dev\_x, dev\_y),  
 callbacks=[reduce\_lr, stopping, logs, checkpointer])

主要利用fit（）关键函数实现训练集数据的导入，这里同时可以设置batchsize的大小，以及通过callback（）函数更新参数，保存模型。

### ⑦ util.py

自定义的一个函数，主要实现保存keras模型，供训练函数调用。

代码实现如下：

**import** os  
**import** pickle **as** pickle  
  
  
**def** load(dirname):  
 preproc\_f = os.path.join(dirname, **"preproc.bin"**)  
 **with** open(preproc\_f, **'r'**) **as** fid:  
 preproc = pickle.load(fid)  
 **return** preproc  
  
  
**def** save(preproc, dirname):  
 preproc\_f = os.path.join(dirname, **"preproc.bin"**)  
 **with** open(preproc\_f, **'w'**) **as** fid:  
 pickle.dump(preproc, fid)