**实验记录文档\_二值化ECG分类**

目录

[BNN\_ECG\_NET v1 2](#_Toc17212)

[超参数调整 3](#_Toc24276)

[EXPERIMENT 1 4](#_Toc25765)

[EXPERIMENT 2 5](#_Toc1241)

[EXPERIMENT 3 6](#_Toc7413)

[超参数及调参方式汇总 6](#_Toc359)

[最终运行的结果（部分） 6](#_Toc18672)

**设计思想：**

对MIT-BIH数据集中的心电信号进行17分类，压缩卷积神经网络。

**数据集介绍：**

MIT（Massachusetts Institute of Technology ）

BIH（Beth Israel Hospital ）

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。

其中，23个（“100系列”）是从超过4000个动态心电图带的集合中随机选择的，另外25个（“200系列”）是选择包括不常见但临床上重要的心律失常的例子。受试者包括25名年龄在32至89岁之间的男性和22名23岁到89岁的女性。大约60%的受试者都是住院患者。

本课题对心电信号进行17分类，采用的是Paweł Pławiak的二次分类数据集。

17classes：'NSR', 'APB', 'AFL', 'AFIB', 'SVTA', 'WPW','PVC', 'Bigeminy', 'Trigeminy', 'VT', 'IVR', 'VFL', 'Fusion', 'LBBBB', 'RBBBB', 'SDHB', 'PR'

ECG信号长度为10s，采样频率360Hz，即每个样本有3600个数据，总的样本数为1000，数据格式是.mat。

**网络部分：**

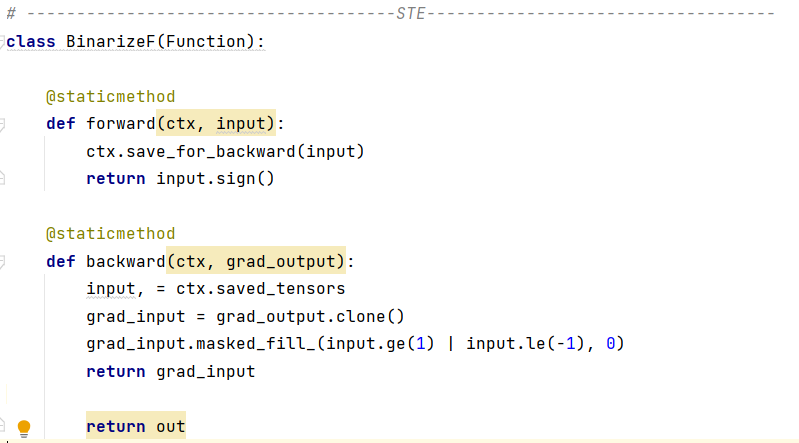
二值量化分为权重二值化和全二值化。权重二值化只将权重量化到1bit，而全二值化则将权重和激活都量化到1bit。对权重二值化会损失一部分精度，在权重二值化的基础上对激活二值化则会进一步降低精度，此文档是有关优化全二值化网络的实验记录。

**实验过程**

# BNN\_ECG\_NET v1：Naive BNN

## 二值化函数：STE

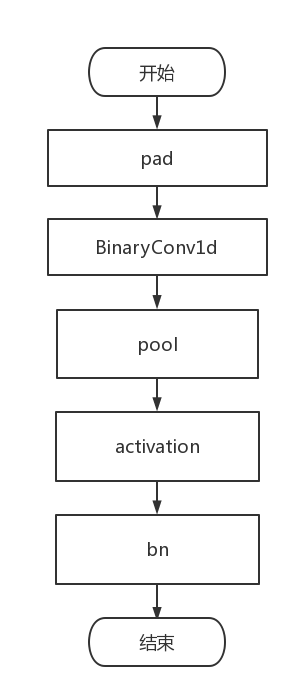
**函数定义如下**



## 网络基本模块

**Bn\_bin\_conv\_poo l= Conv + MaxPool + PRelu + BN**

**功能实现流程如下：**



首先，对输入张量 I 进行常数填充，以使得边缘像素也能够参与卷积运算；

self.pad = nn.ConstantPad1d(padding=padding, value=padding\_value)

使用 BinaryConv1d 进行一维二值化卷积操作（此操作在下文详细介绍），并将其输出作为下一层的输入；

self.conv = BinaryConv1d(in\_channels, out\_channels, kernel\_size=kernel\_size, stride=stride, padding=0, bias=**False**)

对卷积结果进行最大池化操作，将特征图的尺寸进行压缩；

self.pool = nn.MaxPool1d(kernel\_size=pool\_size, stride=pool\_stride)

通过 PReLU 函数进行激活处理，增加非线性特征；

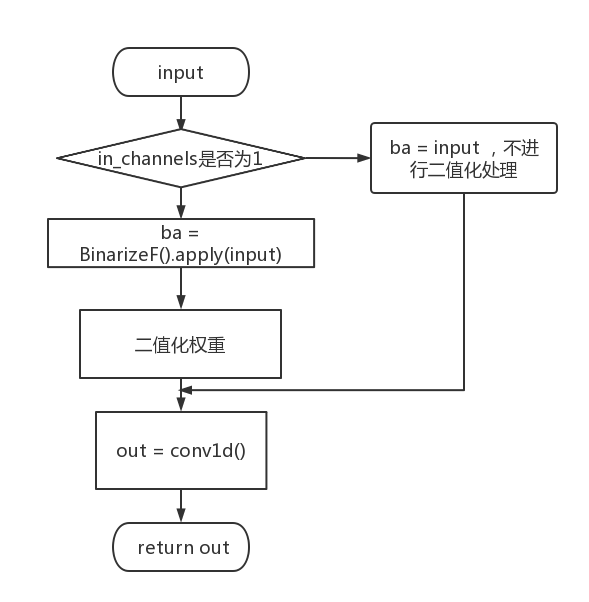
self.prelu = nn.PReLU()

对激活后的结果进行批标准化，加速网络收敛。

**网络基本块各层作用**

* 其中BN层的作用是归一化，二值化网络由于权重均为正一或者负一，会导致中间值的绝对值过大，分布偏离正态分布，不利于收敛。不添加BN层的情况下，网络的精度较低。BN层的位置还极大地影响了二值神经网络训练的稳定性，因此要将数据输入的均值和方差保持在合理范围内，并且训练过程变得更加平滑。
* 一维无偏置卷积，填充值为1，填充大小为5
* 采用池化层代替全连接层

其中BinaryConv1d是自定义的二值化无偏置卷积计算，在前向计算过程中，该类首先对输入和权重进行二值化处理，即将它们的元素值变为+1 或-1的形式。这里使用了上述提到的BinarizeF函数实现二值化操作。二值化卷积流程图如下：



## 网络结构

**五个基本块输入输出通道大小如下表所示**：

|  |  |  |  |
| --- | --- | --- | --- |
| layer\_num | layer\_name | in\_chanel | out\_chanel |
| 1 | Bn\_bin\_  conv\_pool1 | 1 | 8 |
| 2 | Bn\_bin\_  conv\_pool2 | 8 | 16 |
| 3 | Bn\_bin\_  conv\_pool3 | 16 | 32 |
| 4 | Bn\_bin\_  conv\_pool4 | 32 | 32 |
| 5 | Bn\_bin\_  conv\_pool5 | 32 | 17 |

**网络各层参数设置如下表所示：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| layer\_num | layer\_name | kernel\_size | stride | padding | pad\_value |
| 1 | conv1\_pool1 | 9 | 3 | 5 | 1 |
| 2 | MaxPool1 | 5 | 2 |  |  |
| 3 | Act1=PRelu() | | | | |
| 4 | BN1 | | | | |
| 5 | conv2\_pool2 | 9 | 1 | 5 | 1 |
| 6 | MaxPool1 | 5 | 2 |  |  |
| 7 | Act2=PRelu() | | | | |
| 8 | BN2 | | | | |
| 9 | conv3\_pool3 | 9 | 1 | 5 | 1 |
| 10 | MaxPool1 | 5 | 2 |  |  |
| 11 | Act3=PRelu() | | | | |
| 12 | BN3 | | | | |
| 13 | conv4\_pool4 | 9 | 1 | 5 | 1 |
| 14 | MaxPool1 | 5 | 2 |  |  |
| 15 | Act4=PRelu() | | | | |
| 16 | BN4 | | | | |
| 17 | conv5\_pool5 | 9 | 1 | 5 | 1 |
| 18 | MaxPool1 | 5 | 2 |  |  |
| 19 | Act5=PRelu() | | | | |
| 20 | BN5 | | | | |

## 超参数调整

* **epochs：1000**

在梯度下降的模型训练的过程中，神经网络逐渐从不拟合状态到优化拟合状态，达到最优状态之后会进入过拟合状态。因此epoch并非越大越好，一般是指在50到200之间。数据越多样，相应epoch就越大。可以根据loss的收敛情况来选择epoch大小。

* **Batch\_Size ：64**

**一般选择2的指数倍。为了提高处理数据的速度选择了64。**

 Batch\_Size太小，算法不收敛。

 随着 Batch\_Size 增大，处理相同数据量的速度越快。

 随着 Batch\_Size 增大，达到相同精度所需要的 epoch 数量越来越多。

当 Batch\_Size 增大到某个时候，达到时间上的最优。

 由于最终收敛精度会陷入不同的局部极值，因此 Batch\_Size 增大到某些时候，达到最终收敛精度上的最优。

* **Learning-Rate：**

学习率设置过小的时候，每步太小，下降速度太慢，可能要花很长的时间才会找到最小值。

学习率过大过大的时候，每步太大，虽然收敛得速度很快，但可能会跨过或忽略了最小值，导致一直来回震荡而无法收敛。

刚开始训练时：学习率以 0.01 ~ 0.001 为宜；一定轮数过后：逐渐减缓；接近训练结束：学习速率的衰减应该在100倍以上。可以通过观察loss来选择学习率对收敛的影响。

* **优化器+lr\_scheduler：**

SGD，Stochastic Gradient Descent，随机梯度下降。

Adma，daptive Moment Estimation，自适应矩估计。梯度下降速度快，使用简单，但是容易在最优值附近震荡。

* **激活函数：**

激活函数 (Activation functions) 对于人工神经网络模型去学习、理解非常复杂和非线性的函数来说具有十分重要的作用。引入激活函数可以将非线性特性引入到神经网络中。

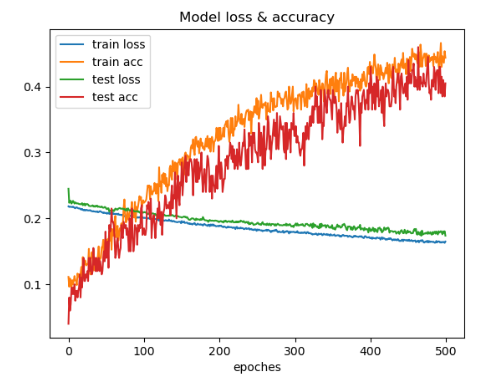
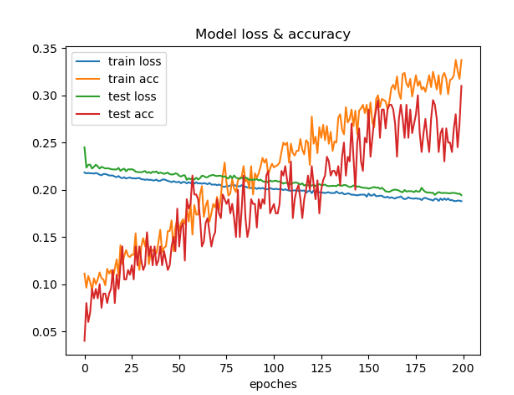
激活函数分为两类，饱和激活函数和非饱和激活函数。饱和激活函数有Sigmod、tanh，非饱和激活函数有ReLu、PReLu、Hardtanh等等。

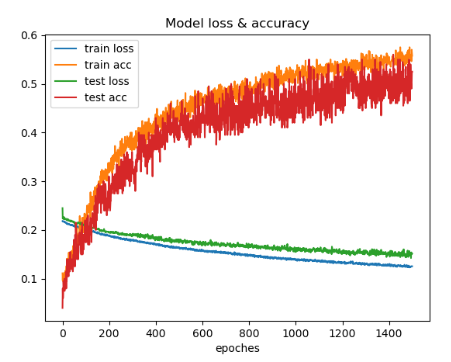
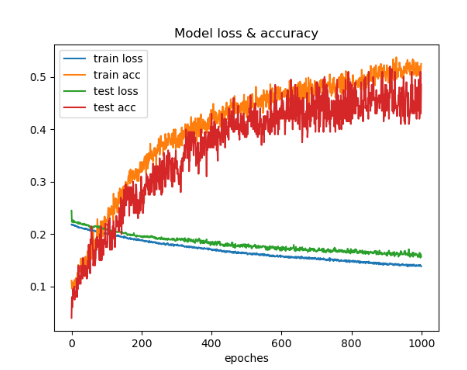
层数比较多的神经网络模型在训练的时候会出现梯度消失(gradient vanishing problem)和梯度爆炸(gradient exploding problem)问题。梯度消失问题和梯度爆炸问题一般会随着网络层数的增加变得越来越明显。当激活函数为饱和函数时，容易发生梯度消失，此时可以用非饱和激活函数来代替。

### EXPERIMENT 1

ps：淡黄色底纹的表格代表此实验正在修改的部分

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | batchsize | Learning-Rate | 优化器+lr\_scheduler： |
| * 200 * 500 * 1000 * 1500 | 64 | 0.001 | optimizer = torch.optim.SGD(model.parameters(), lr=lr)  torch.optim.lr\_scheduler.ExponentialLR(optimizer, gamma= 0.5) |

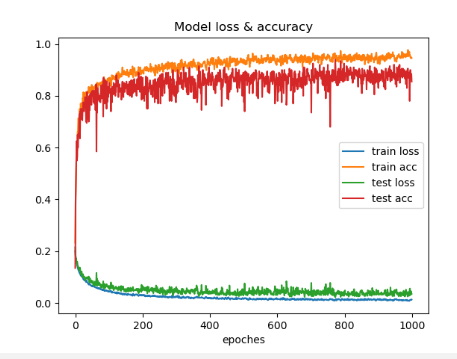




**epochs=1500跟epochs=1000相比已经没有太大的提升了，所以epochs减少为1000.但此时的准确率还未达到60%，所以进一步优化参数。**

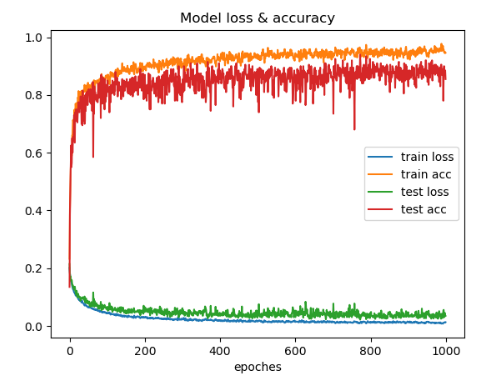
### EXPERIMENT 2

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | batchsize | Learning-Rate | 优化器+lr\_scheduler： |
| 1000 | 64 | 0.001 | optimizer = optim.Adam(model.parameters(),lr=lr)  torch.optim.lr\_scheduler.ExponentialLR(optimizer, gamma= 0.5) |



当把优化器换成Adam时，准确率大大提高，且loss也能较好的收敛。可能是因为网络模型较复杂不太适合使用SGD，而且SGD优化器在模型中学习的速度很慢，不如所以毕设里采用Adma优化器。

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | batchsize | Learning-Rate | 优化器+lr\_scheduler： |
| 1000 | 64 | 0.001 | optimizer = optim.Adam(model.parameters(),lr=lr)  lr\_scheduler=torch.optim.lr\_scheduler.CosineAnnealingWarmRestarts(optimizer,epochs,1,0.02) |



学习率优化方式改为带预热的余弦退火，准确率没有提高。

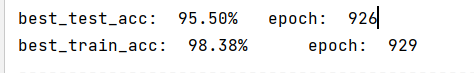
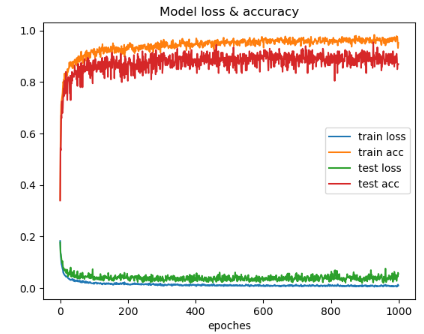
学习率优化方式改为余弦退火，准确率仍没提高。

改为StepLR的时候最高的准确率下降至88%。

认为更改lr\_scheduler对结果不会有太大影响了，所以进行实验3.

### EXPERIMENT 3

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | batchsize | Learning-Rate | 优化器+lr\_scheduler： |
| 1000 | 64 | 0.01 | optimizer = optim.Adam(model.parameters(),lr=lr)  lr\_scheduler = torch.optim.lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=epochs, eta\_min=0.05,last\_epoch=-1) |



当把初始的lr调大至0.01时，最高准确率可以提高到95.5%，loss曲线已经可以收敛的很好了。如果初始lr设置为0.001，数值较小，再降低很容易到达极值点。所以可以把lr改为0.01.

### EXPERIMENT 4

固定上述参数之后更换激活函数测试模型的性能

|  |  |  |  |
| --- | --- | --- | --- |
| 激活函数 | image | ACC |  |
| ReLu() |  | best\_test\_acc: 94.00%  best\_train\_acc: 97.62% |  |
| PReLu() |  | best\_test\_acc: 95.50%  best\_train\_acc: 98.38% |  |
| tanh() |  | best\_test\_acc: 30.00%  best\_train\_acc: 31.00% |  |
| Htanh() |  | best\_test\_acc: 28.50%  best\_train\_acc: 29.12% | 准确率极低，loss在几个epoch后便不再下降 |

从上表可以看出PReLu()激活的效果最好。

## 超参数及调参方式汇总

|  |  |
| --- | --- |
| epochs | 1000 |
| batch\_size | 64 |
| 损失函数 | 交叉熵损失 |
| learning\_rate | 0.01 |
| 优化器 | optimizer = optim.Adam(model.parameters(), lr=lr) |
| 自适应学习率 | torch.optim.lr\_scheduler.CosineAnnealingWarmRestarts(optimizer,epochs,1,0.02) |
| 激活函数 | PReLu() |
| seed | 110 |

## 最终运行的结果（部分）

best\_test\_acc: 95.50% epoch: 926

best\_train\_acc: 98.38% epoch: 929

the model accuracy is 0.87

+-----------+-------+-------+-------+-------+-------+

| | ACC | SEN | SPE | PPR | F1 |

+-----------+-------+-------+-------+-------+-------+

| NSR | 0.91 | 0.911 | 0.91 | 0.797 | 0.85 |

| APB | 0.955 | 0.625 | 0.984 | 0.769 | 0.69 |

| AFL | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| AFIB | 0.99 | 0.929 | 1.0 | 1.0 | 0.963 |

| SVTA | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| WPW | 0.995 | 0.8 | 1.0 | 1.0 | 0.889 |

| PVC | 0.935 | 0.778 | 0.96 | 0.75 | 0.764 |

| Bigeminy | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| Trigeminy | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| VT | 0.995 | 1.0 | 0.995 | 0.667 | 0.8 |

| IVR | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| VFL | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| Fusion | 0.985 | 0.0 | 1.0 | 0.0 | 0.0 |

| LBBBB | 0.985 | 0.929 | 0.989 | 0.867 | 0.897 |

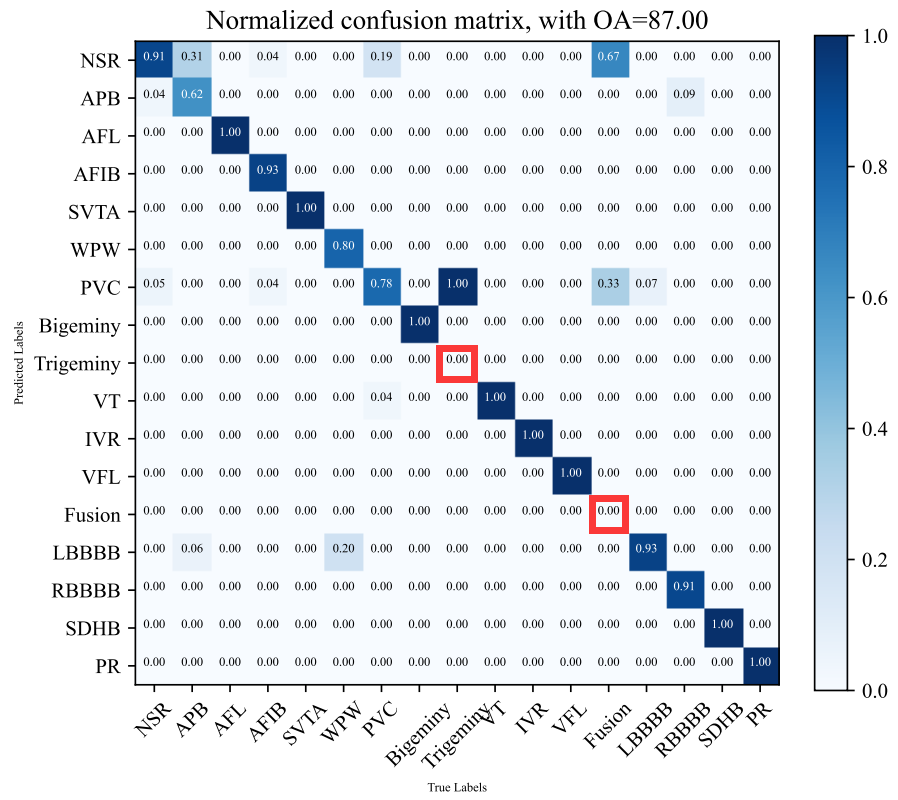
| RBBBB | 0.995 | 0.909 | 1.0 | 1.0 | 0.952 |

| SDHB | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| PR | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| SUM | 0.985 | 0.87 | 0.992 | 0.87 | 0.87 |

+-----------+-------+-------+-------+-------+-------+



# BNN\_ECG\_NET v2：XNOR Net

## 修改1：BNN+XNOR

此节修改代码在train\_XNOR\_NET0

### 改进二值化权重的方式

引入比例因子α进行权重二值化，仍然对输入不二值化，网络的层数、结构不做改变，具体参数详见v1中网络结构表格。

**运行结果：**

best\_test\_acc: 51.00% epoch: 879

best\_train\_acc: 53.00% epoch: 643

the model accuracy is 0.46

+-----------+-------+-------+-------+-------+-------+

| | ACC | SEN | SPE | PPR | F1 |

+-----------+-------+-------+-------+-------+-------+

| NSR | 0.62 | 0.804 | 0.549 | 0.409 | 0.542 |

| APB | 0.92 | 0.0 | 1.0 | 0.0 | 0.0 |

| AFL | 0.97 | 0.0 | 1.0 | 0.0 | 0.0 |

| AFIB | 0.865 | 0.821 | 0.872 | 0.511 | 0.63 |

| SVTA | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| WPW | 0.975 | 0.0 | 1.0 | 0.0 | 0.0 |

| PVC | 0.87 | 0.037 | 1.0 | 1.0 | 0.071 |

| Bigeminy | 0.94 | 0.0 | 1.0 | 0.0 | 0.0 |

| Trigeminy | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| VT | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| IVR | 0.99 | 1.0 | 0.99 | 0.5 | 0.667 |

| VFL | 0.98 | 0.25 | 0.995 | 0.5 | 0.333 |

| Fusion | 0.985 | 0.0 | 1.0 | 0.0 | 0.0 |

| LBBBB | 0.945 | 0.786 | 0.957 | 0.579 | 0.667 |

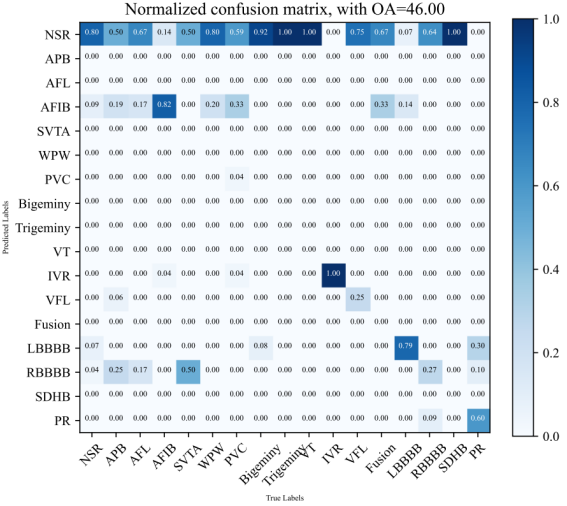
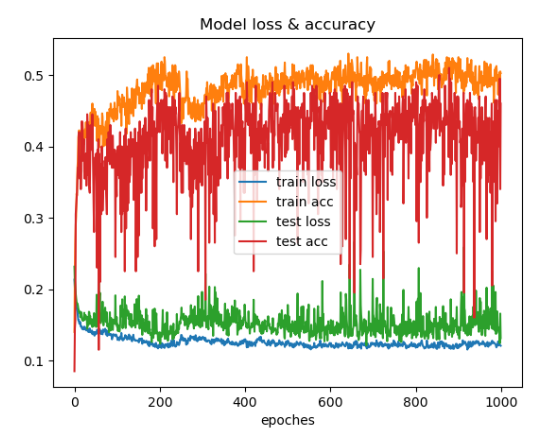
| RBBBB | 0.915 | 0.273 | 0.952 | 0.25 | 0.261 |

| SDHB | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| PR | 0.975 | 0.6 | 0.995 | 0.857 | 0.706 |

| SUM | 0.936 | 0.46 | 0.966 | 0.46 | 0.46 |

+-----------+-------+-------+-------+-------+-------+



模型表现大大下降，按理说引入可调整的比例因子不应该性能下降很多，检查了下代码发现是没有在训练时对权重的操作步骤顺序搞反了，改过来后重新运行。

**运行结果：**

best\_test\_acc: 79.50% epoch: 4829

best\_train\_acc: 80.62% epoch: 4794



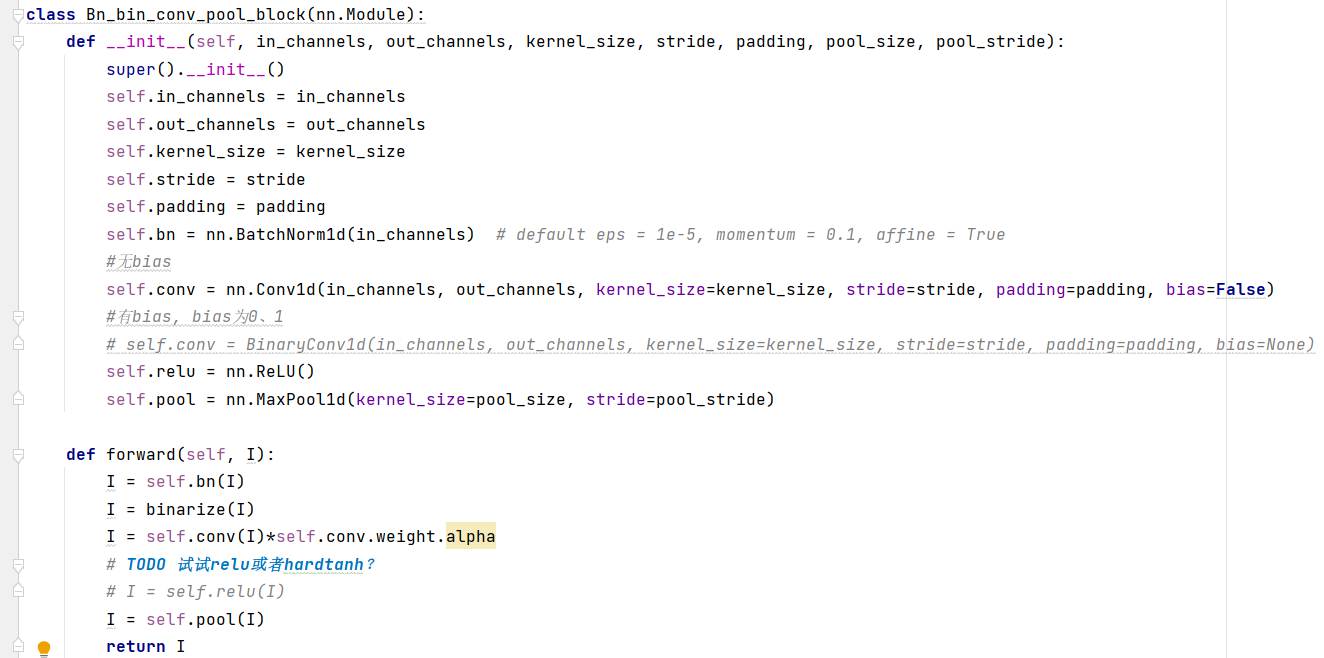
准确度有提升。但是不如用STE二值化的效果好，还没找到原因。

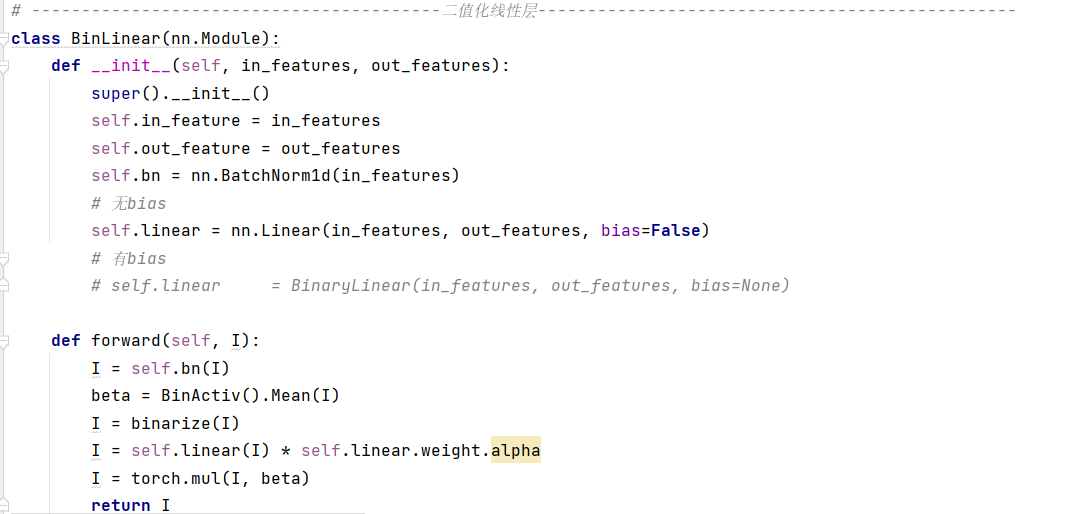
### 二值化输入

对输入进行二值化，权重二值化的方式仍采用用比例因子进行调整，并且引入比例因子对卷积进行近似处理，线性层同理。block结构如下所示，7个block，两个线性层

超参数设置如下：

|  |  |
| --- | --- |
| epochs | 1000 |
| batch\_size | 64 |
| 损失函数 | 交叉熵损失 |
| learning\_rate | 0.01 |
| 优化器 | optimizer = optim.Adam(model.parameters(), lr=lr) |
| 学习率调整 | lr\_scheduler=torch.optim.lr\_scheduler.CosineAnnealingWarmRestarts(optimizer,epochs,1,0.02) |
| 激活函数 | ReLu() |
| seed | 110 |





运行结果：

the model accuracy is 0.85

+-----------+-------+-------+-------+-------+-------+

| | ACC | SEN | SPE | PPR | F1 |

+-----------+-------+-------+-------+-------+-------+

| NSR | 0.915 | 0.911 | 0.917 | 0.81 | 0.857 |

| APB | 0.93 | 0.562 | 0.962 | 0.562 | 0.562 |

| AFL | 0.995 | 0.833 | 1.0 | 1.0 | 0.909 |

| AFIB | 0.995 | 0.964 | 1.0 | 1.0 | 0.982 |

| SVTA | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| WPW | 0.985 | 0.8 | 0.99 | 0.667 | 0.727 |

| PVC | 0.945 | 0.778 | 0.971 | 0.808 | 0.792 |

| Bigeminy | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| Trigeminy | 0.99 | 0.0 | 0.995 | 0.0 | 0.0 |

| VT | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| IVR | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| VFL | 0.995 | 0.75 | 1.0 | 1.0 | 0.857 |

| Fusion | 0.99 | 0.333 | 1.0 | 1.0 | 0.5 |

| LBBBB | 0.985 | 0.857 | 0.995 | 0.923 | 0.889 |

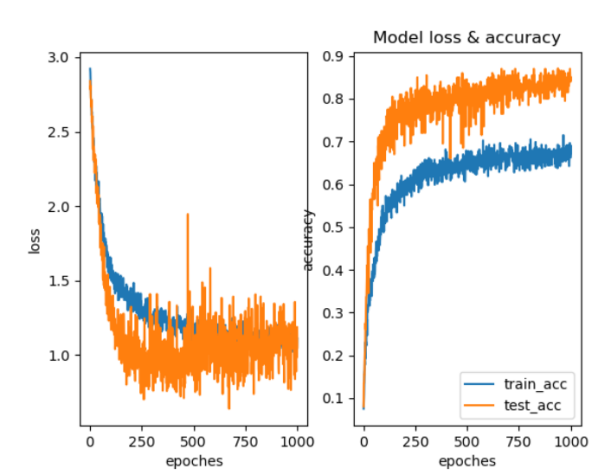
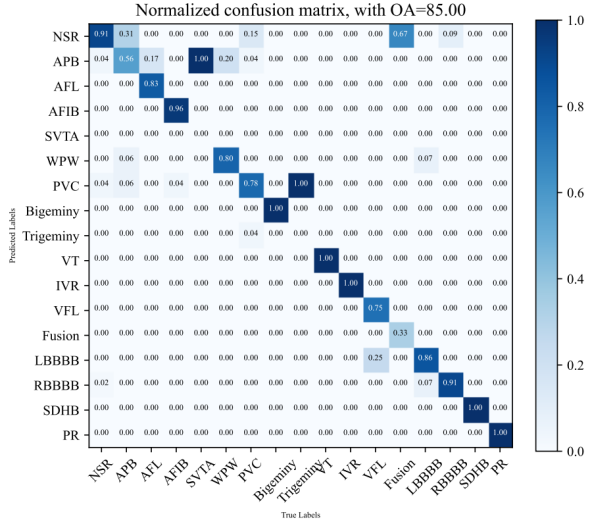
| RBBBB | 0.985 | 0.909 | 0.989 | 0.833 | 0.87 |

| SDHB | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| PR | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| SUM | 0.982 | 0.85 | 0.991 | 0.85 | 0.85 |

+-----------+-------+-------+-------+-------+-------+



与Naive BNN的模型精度差不多，test的损失波动较大。

## 修改2：XNOR NET

此节修改代码在train\_XNOR\_NET1

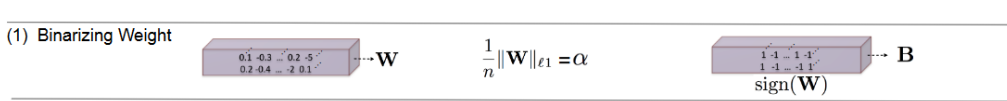
### EXPERIMENT 1：改变网络结构

去掉了两个线性层，用池化层代替，一共5个block，block前向传播大致如下：

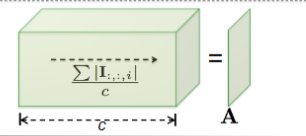


二值化输入和二值化权重的方式介绍如下：

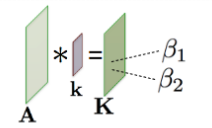
XNOR-Net与之前的Naive BNN有较大不同，一是将输入input二值化，而Naive BNN第1层输入是浮点数与二值化的weight卷积。二是改进了新的二值化方法，简单来说就是算出X、W的L1范数的均值，并以此为比例因子，它的的计算方式为：对于每一个通道（channel），将每个参数的绝对值求和，再除以元素总数n，得到该通道上的平均l1范数，最后在各个通道之间进行广播。然后再乘以sign(X)、sign(W)来近似表示全精度的X、W。二值化权重操作如下图：



二值化input时，由于计算L1正则化时存在很多冗余的计算，所以采用下列操作方式，即将输入在channel维度（c表示通道数）计算均值得到A。



用*k*（*k*为*w×h*大小的卷积核，其值为*1/w×h*）对A进行卷积得到K，K就是卷积后得到的β。



卷积后K会变小，但是之后把I与W卷积，I的维度就和K一样了，这样就可以进行点积。

一个典型的 CNN 具有卷积、批规范化、激活、池化这样的四层结构，其中，池化层可以对输入运用任何种类的池化方式。但在二值化的输入(-1,1)进入到池化过程时，会产生大量的信息丢失。例如，对二值化输入进行 max-pooling 时，会导致大部分输入只剩+1，使得消息消减，精度降低。为了解决这个问题，改善了网络结构，改变这几层的顺序，首先实行批规范化，保证 0 均值，然后进行二值化激活，使数据都是+1 和-1，再做二值化卷积，此时由于比例因子的作用输出的不再是-1 和+1，这会相对减少信息丢失。在这里，建议在二值化卷积后加一个非二值化激活步骤(如 ReLU)，这可以帮助训练比较复杂的网络。

二值化权重方面，定义了WeightOperation类，里面包括保存权重WeightSave、权重二值化WeightBinarize、恢复权重WeightRestore以及参数更新WeightGradient等一系列操作，通过实例化对象weightOP来完成这些操作。

在训练过程中，首先使用使用 weightOP.WeightBinarize()方法将权重进行二值化处理，然后进行：前向传播——>计算损失并累积——>梯度清零——>反向传播——>使用优化器更新权重。在下一次迭代中，又会恢复全精度权重进行训练，如此交替进行。

### EXPERIMENT 2：学习率调整

超参数设置如下：

|  |  |
| --- | --- |
| epochs | 1000 |
| batch\_size | 32 |
| 损失函数 | 交叉熵损失 |
| learning\_rate | 0.01 |
| 优化器 | optimizer = optim.Adam(model.parameters(), lr=lr) |
| 学习率调整 | 每15个epoch下降1个数量级 |
| 激活函数 | PReLu() |
| seed | 110 |

**学习率优化方式**

每15个epoch将学习率降低一个数量级

**运行结果**

best\_test\_acc: 43.50% epoch: 962

best\_train\_acc: 50.12% epoch: 994

--------------------------------------------------

the model accuracy is 0.43

+-----------+-------+-------+-------+-------+-------+

| | ACC | SEN | SPE | PPR | F1 |

+-----------+-------+-------+-------+-------+-------+

| NSR | 0.685 | 0.696 | 0.681 | 0.459 | 0.553 |

| APB | 0.93 | 0.188 | 0.995 | 0.75 | 0.3 |

| AFL | 0.97 | 0.0 | 1.0 | 0.0 | 0.0 |

| AFIB | 0.845 | 0.714 | 0.866 | 0.465 | 0.563 |

| SVTA | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| WPW | 0.975 | 0.0 | 1.0 | 0.0 | 0.0 |

| PVC | 0.815 | 0.222 | 0.908 | 0.273 | 0.245 |

| Bigeminy | 0.935 | 0.25 | 0.979 | 0.429 | 0.316 |

| Trigeminy | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| VT | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| IVR | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| VFL | 0.98 | 0.0 | 1.0 | 0.0 | 0.0 |

| Fusion | 0.985 | 0.0 | 1.0 | 0.0 | 0.0 |

| LBBBB | 0.925 | 0.786 | 0.935 | 0.478 | 0.595 |

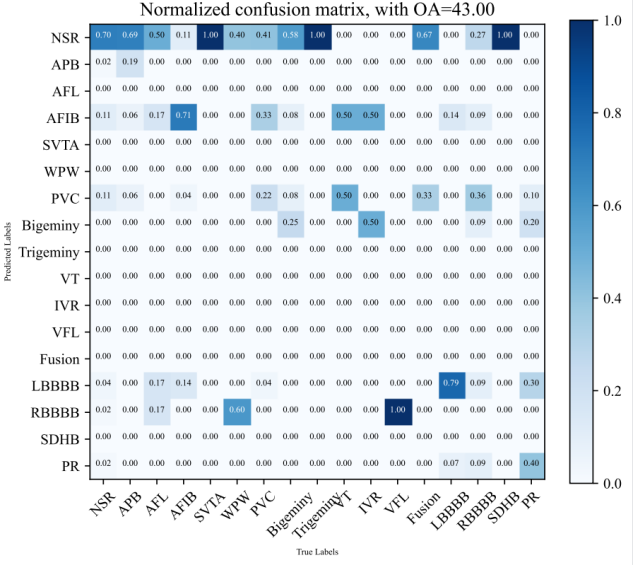
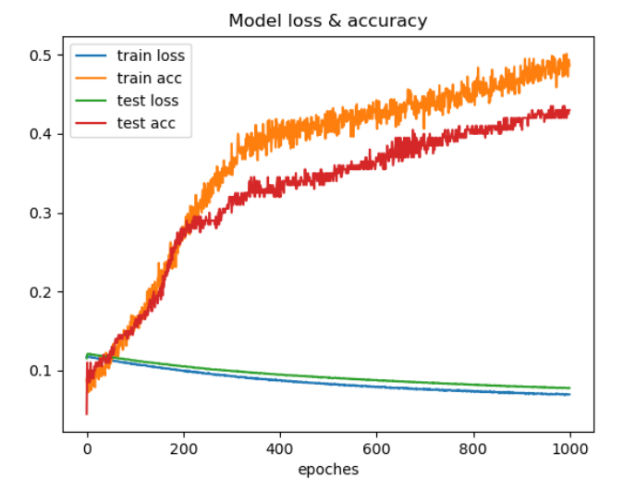
| RBBBB | 0.9 | 0.0 | 0.952 | 0.0 | 0.0 |

| SDHB | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| PR | 0.955 | 0.4 | 0.984 | 0.571 | 0.471 |

| SUM | 0.933 | 0.43 | 0.964 | 0.43 | 0.43 |

+-----------+-------+-------+-------+-------+-------+



感觉loss还有要下降的趋势就设置了3000个epoch，结果acc和loss跑飞了

自适应调整学习率

lr\_scheduler=torch.optim.lr\_scheduler.CosineAnnealingWarmRestarts(optimizer,epochs,1,0.02)

**运行结果：**

best\_test\_acc: 62.50% epoch: 680

best\_train\_acc: 65.75% epoch: 897

--------------------------------------------------

the model accuracy is 0.595

+-----------+-------+-------+-------+-------+-------+

| | ACC | SEN | SPE | PPR | F1 |

+-----------+-------+-------+-------+-------+-------+

| NSR | 0.74 | 0.875 | 0.688 | 0.521 | 0.653 |

| APB | 0.905 | 0.062 | 0.978 | 0.2 | 0.095 |

| AFL | 0.97 | 0.0 | 1.0 | 0.0 | 0.0 |

| AFIB | 0.945 | 0.821 | 0.965 | 0.793 | 0.807 |

| SVTA | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| WPW | 0.985 | 0.6 | 0.995 | 0.75 | 0.667 |

| PVC | 0.875 | 0.444 | 0.942 | 0.545 | 0.49 |

| Bigeminy | 0.935 | 0.25 | 0.979 | 0.429 | 0.316 |

| Trigeminy | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| VT | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| IVR | 0.99 | 0.0 | 1.0 | 0.0 | 0.0 |

| VFL | 0.98 | 0.0 | 1.0 | 0.0 | 0.0 |

| Fusion | 0.985 | 0.0 | 1.0 | 0.0 | 0.0 |

| LBBBB | 0.965 | 0.643 | 0.989 | 0.818 | 0.72 |

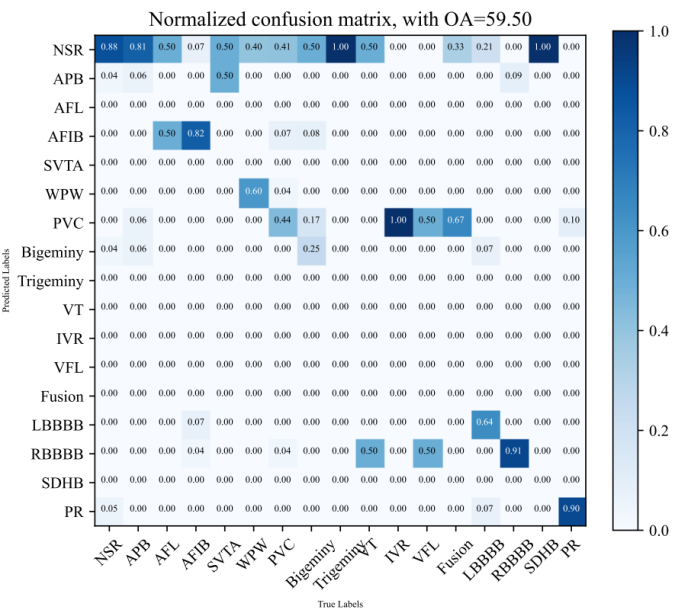
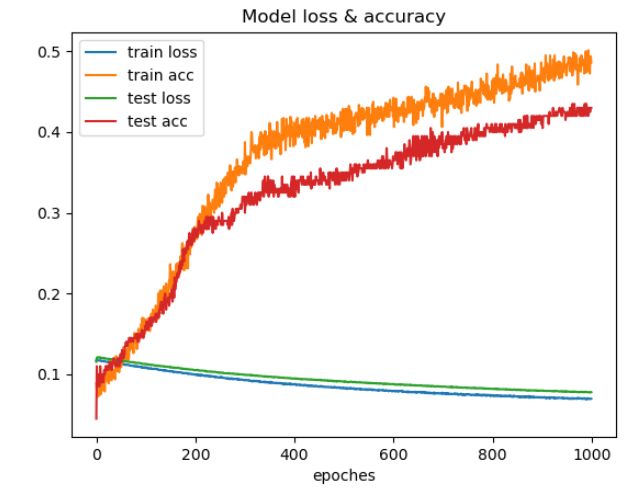
| RBBBB | 0.97 | 0.909 | 0.974 | 0.667 | 0.769 |

| SDHB | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| PR | 0.975 | 0.9 | 0.979 | 0.692 | 0.783 |

| SUM | 0.952 | 0.595 | 0.975 | 0.595 | 0.595 |

+-----------+-------+-------+-------+-------+-------+



准确率有了很大的提升，混淆矩阵对角线比上一次好很多，原本有几类分不出来的现在可以有较高的分类精度了。

### EXPERIMENT 2：增加block

增加一个block

**运行结果：**

best\_test\_acc: 71.00% epoch: 935

best\_train\_acc: 73.50% epoch: 596

--------------------------------------------------

the model accuracy is 0.705

+-----------+-------+-------+-------+-------+-------+

| | ACC | SEN | SPE | PPR | F1 |

+-----------+-------+-------+-------+-------+-------+

| NSR | 0.82 | 0.893 | 0.792 | 0.625 | 0.735 |

| APB | 0.945 | 0.5 | 0.984 | 0.727 | 0.593 |

| AFL | 0.97 | 0.0 | 1.0 | 0.0 | 0.0 |

| AFIB | 0.96 | 0.857 | 0.977 | 0.857 | 0.857 |

| SVTA | 0.985 | 0.0 | 0.995 | 0.0 | 0.0 |

| WPW | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| PVC | 0.91 | 0.519 | 0.971 | 0.737 | 0.609 |

| Bigeminy | 0.935 | 0.417 | 0.968 | 0.455 | 0.435 |

| Trigeminy | 0.995 | 0.0 | 1.0 | 0.0 | 0.0 |

| VT | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| IVR | 0.985 | 0.0 | 0.995 | 0.0 | 0.0 |

| VFL | 0.98 | 0.0 | 1.0 | 0.0 | 0.0 |

| Fusion | 0.98 | 0.0 | 0.995 | 0.0 | 0.0 |

| LBBBB | 0.97 | 0.929 | 0.973 | 0.722 | 0.812 |

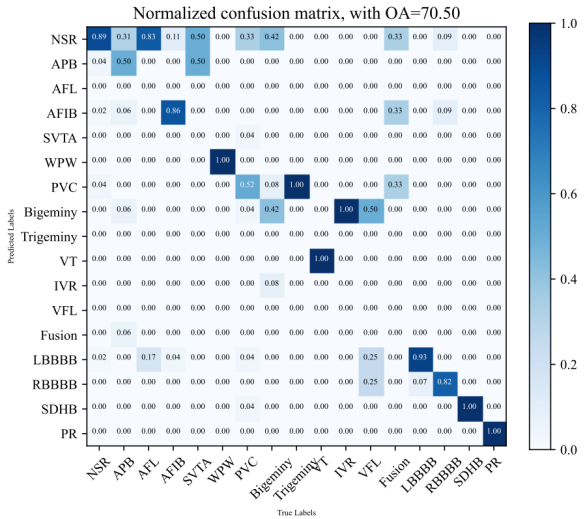
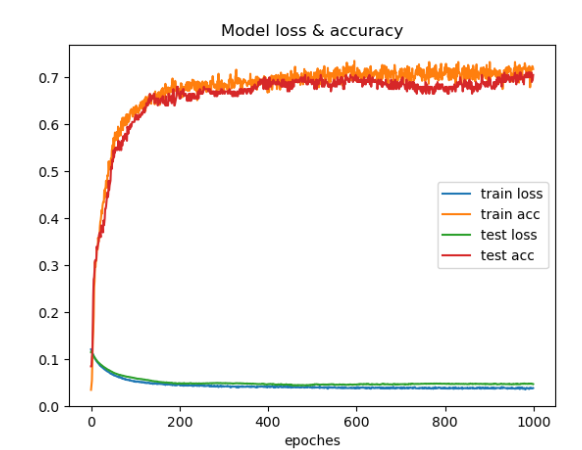
| RBBBB | 0.98 | 0.818 | 0.989 | 0.818 | 0.818 |

| SDHB | 0.995 | 1.0 | 0.995 | 0.5 | 0.667 |

| PR | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

| SUM | 0.965 | 0.705 | 0.982 | 0.705 | 0.705 |

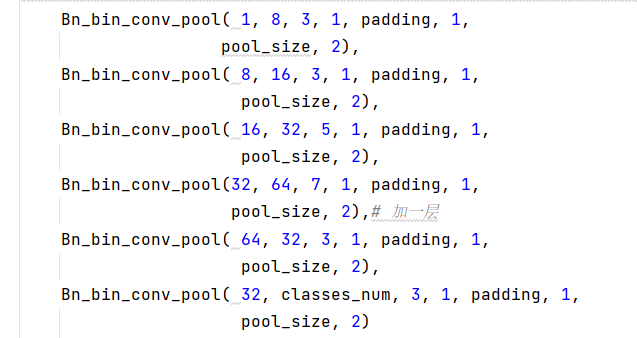
+-----------+-------+-------+-------+-------+-------+

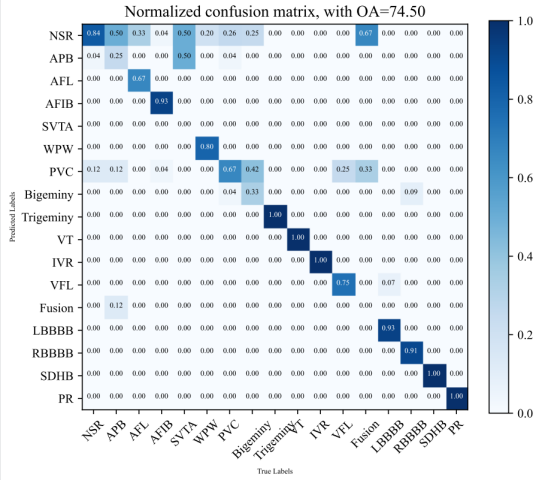
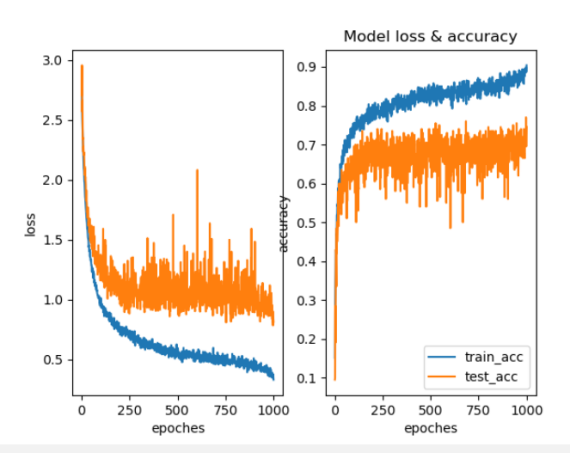


准确率大大提高了，而且混淆矩阵的对角线更加清晰。但是进行再优化batchsize或者学习率模型都没有什么太大的变化，所以决定再更改一下网络层数。

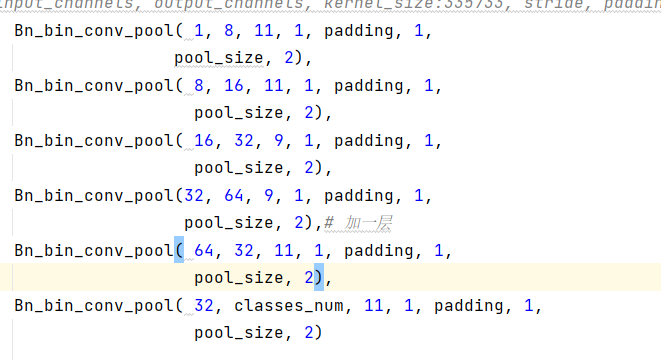
### EXPERIMENT 4：改变卷积核大小

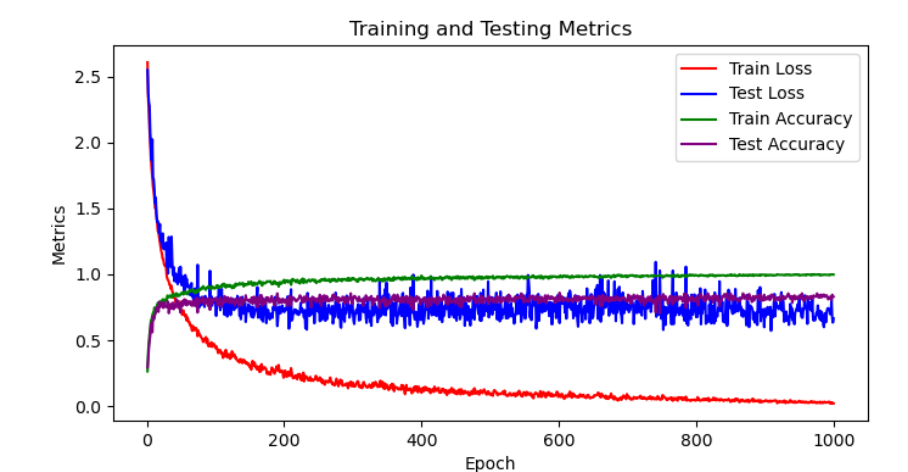
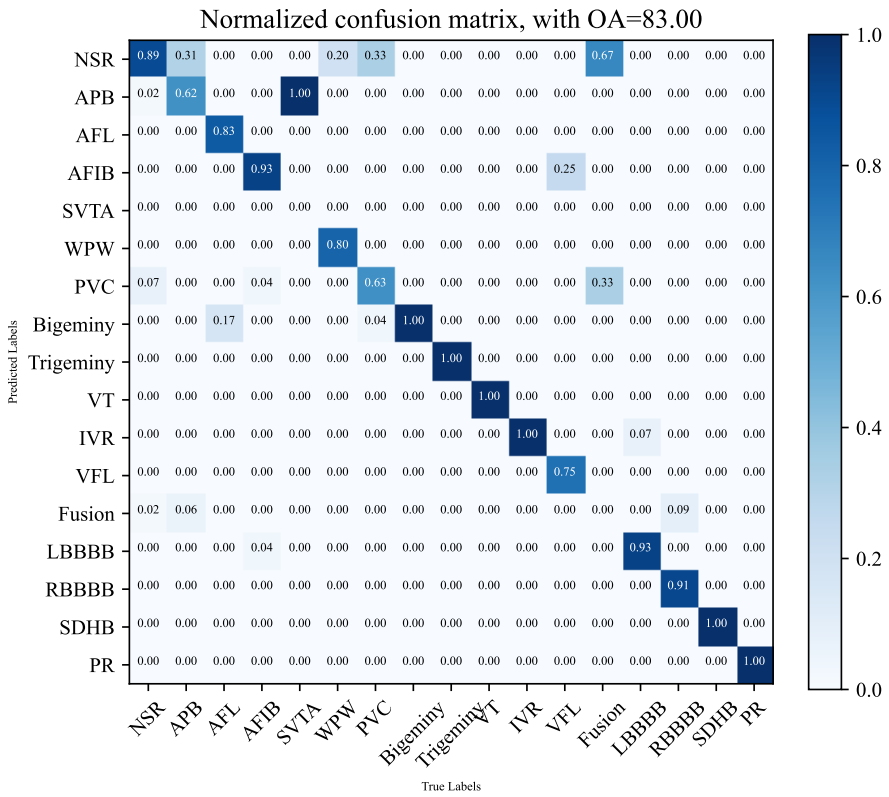
将第一层卷积核大小改成3×3，步长改为1，减小信息损失。第二层改成同样尺度，然后增加一层卷积层





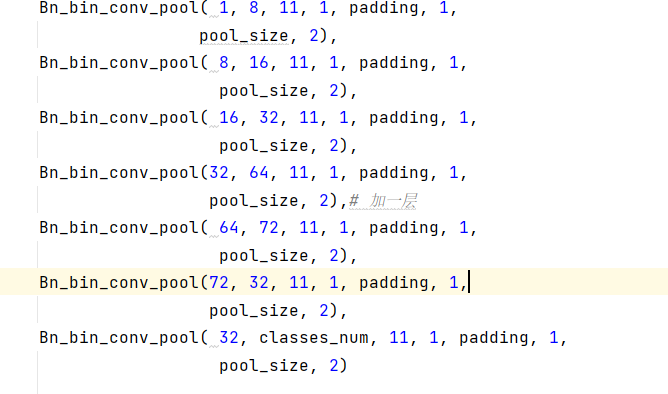
改变卷积核大小

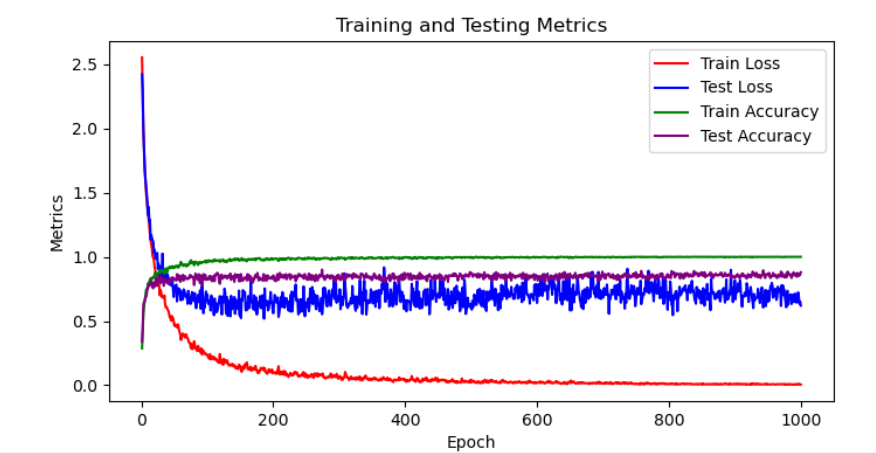


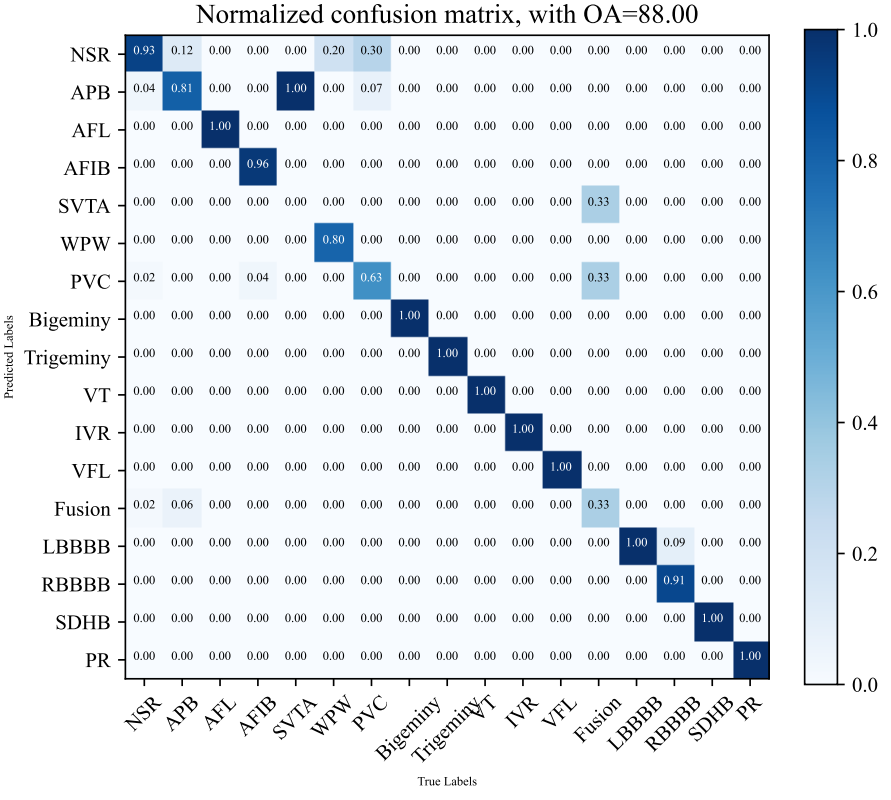


### EXPERIMENT 5

当卷积核大小为11，池化大小为5，padding为5时效果比之前都有很大提升，尝试采用上述尺度的基础上再增加一个block

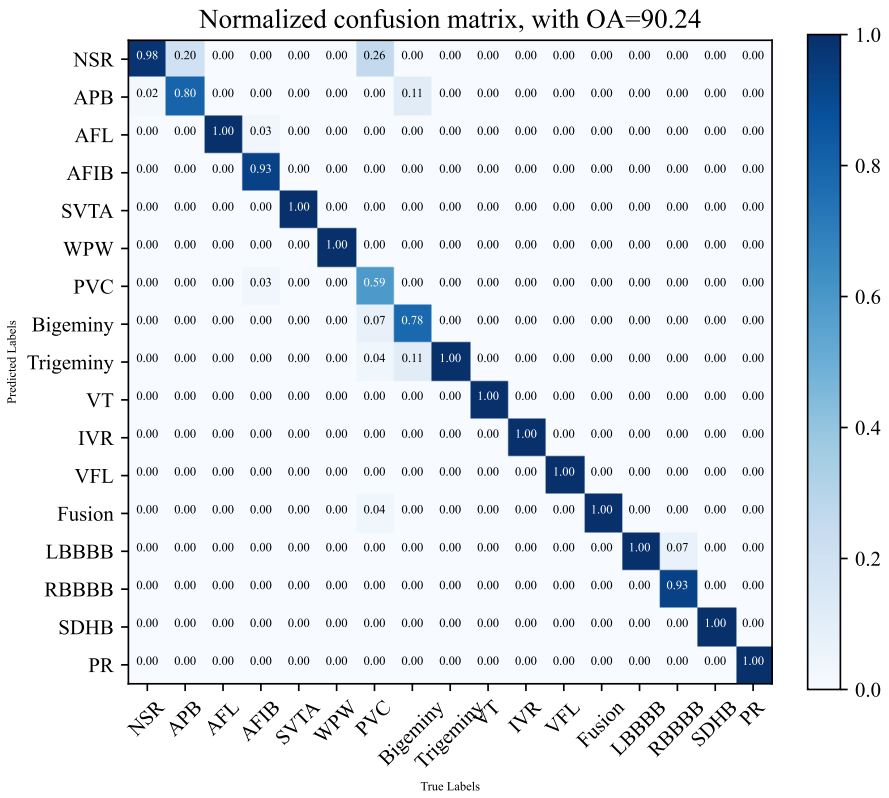
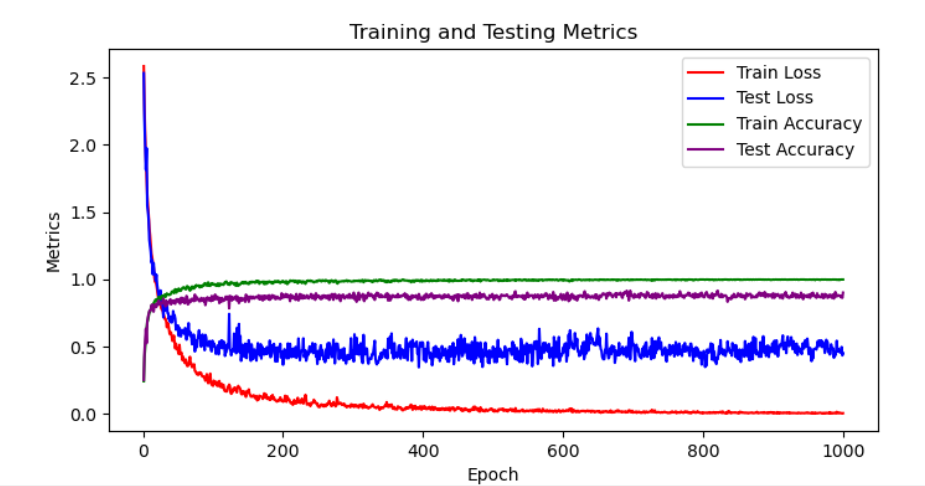






卷积层和池化层已经增加的够多了，但还是有两类分不出来，查看数据集发现这两类特别少，只能稍微增加一下那两类的数据

### EXPERIMENT 6：增加数据



增加了数据集的数量之后那两类可以分出来了，而且训练集loss收敛的比较好，但是测试集的loss收敛情况一般，整体ACC是可以提高到90+了。

# XONR NET++

# IR-NET