# 1.试验系统设计

## 1.1 系统总体结构设计

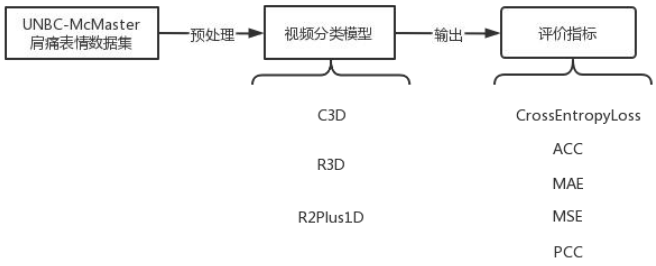
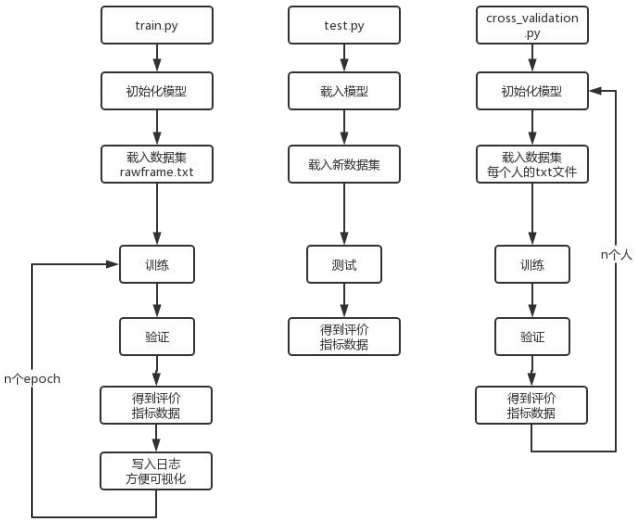



图 1 系统总体结构

## 1.2 代码设计




文件结构：

```
├─network
|  |  c3d-pretrained.pth
|  |  C3D_model.py
|  |  P3D_model.py
|  |  pytorch_i3d.py
|  |  R2Plus1D_model.py
|  └─ R3D_model.py
├─Pain_Images
|  ├─Images_new
|  └─Labels_new
├─run
```

```
|  ├─data
|  |  ├─C3D
|  |  ├─R2Plus1D
|  |  └─R3D
|  └─models
|  cross_validation.py
|  dataset.py
|  data_to_txt.py
|  test.py
|  train.py
```

**data_to_text.py:**

此 python 文件是为了把数据集主要信息打包成 txt 文件。txt 文件中包含了视频地址，视频帧数，视频标签。以其中一个视频为例，在 txt 文件中打印为：

***Pain_Images/Images_new/dr052t1afunaff 168 0***

其地址相对根目录为“Pain_Images/Images_new/dr052t1afunaff”，视频帧数为 168，视频标签为 0。

由于数据集每个类的数目极度不平衡，因此对于视频标签，我们做了减少分类数的处理，将原本的疼痛等级 0-15 减少为 0-5。其中原本的 4-5 疼痛等级归为一级，6-15 疼痛等级归为一级。

该代码主要生成 26 个 txt 文件。其中 1 个 txt 文件包含所有视频信息，另外 25 个 txt 文件分别包含了 25 个测试人员的视频信息。

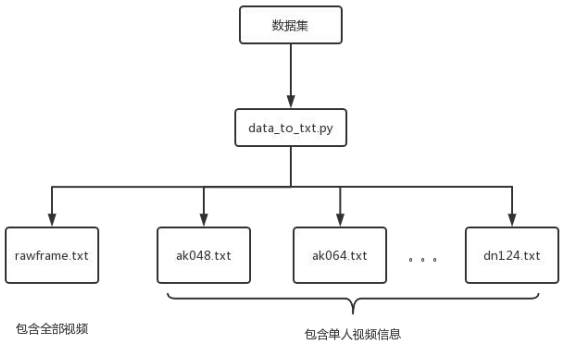



图 2 生成 txt 文件流程

**dataset.py:**

此 python 文件的任务是将数据集进行预处理并转化为可被 pytorch 使用的数据集。其中 Dataset_pain 数据集类继承自 torch.utils.data.Dataset。

```
class Dataset_pain(data.Dataset):
```

以下是类中各个函数的解释：

**__init__(self, txt_name, slip_len, step)**：初始化方法，接收三个参数：txt_name(文本文件名)，slip_len(滑动长度)和 step(步长)。在初始化时，会读取文本文件中的数据，并筛选出满足滑动长度条件的行。其中 self.height 和 self.width 为网络输入的图片高宽。

```
    def __init__(self, txt_name, slip_len, step):
```

```
        self.txt_name = txt_name
        self.slip_len = slip_len
        self.step = step
        self.height = 112
        self.width = 112
        self.data_txt = []
        self.sum_len = (slip_len - 1) * step + 1
        txt_path = os.path.join("data_txt", txt_name + ".txt")
        with open(txt_path, "r") as f:
            lines = f.readlines()
            for line in lines:
                line = line.strip()
                check_line = line.split()
                if int(check_line[1]) > self.sum_len:
                    self.data_txt.append(line)
```

**__len__(self)**：返回样本总数。通过计算 self.data_txt 列表的长度（即视频数）来实现。

**__getitem__(self,index)**：根据给定的索引生成一个数据样本。首先从 self.data_txt 中获取对应的行数据，然后对数据进行预处理，包括加载帧、裁剪、归一化和转换为张量格式。最后返回处理后的帧和标签。

**load_frames(self,file_dir)**：加载视频帧。参数 file_dir 为视频路径，包含视频帧的目录。该函数使用 sorted()函数对目录中所有文件的名称进行排序，并将它们的计数存储在变量 frame_count 中。接下来，创建一个名为 buffer 的空 NumPy 数组，其维度为(frame_count, height, width, 3)。数组的数据类型设置为 float32。然后使用 for 循环迭代目录中的每个文件名，并使用 OpenCV 的 imread()函数读取相应的图像。使用 astype()函数将图像转换为类型为 float64 的 NumPy 数组，并将其调整为具有维度(height, width, 3)的大小。最后，使用索引将调整后的帧分配给 buffer 数组的相应位置。在遍历目录中的所有文件后，返回 buffer 数组。

```
    def load_frames(self, file_dir):
        frames = sorted(os.listdir(file_dir))
        frame_count = len(frames)
        buffer = np.empty((frame_count, self.height, self.width, 3), np.dtype('float32'))
        for i, frame_name in enumerate(frames):
            frame = np.array(cv2.imread(os.path.join(file_dir, frame_name))).astype(np.flo
    at64)
            frame = np.resize(frame, [self.height, self.width, 3])
            buffer[i] = frame
        return buffer
```

**crop(self,buffer)**：帧采样。参数 buffer 是要裁剪的 NumPy 数组。首先生成一个随机整数索引 time_index,其范围在 0 和 buffer 数组长度减去所需裁剪帧长度(self.sum_len)之间。然后使用此索引以步长 self.step 对视频进行帧采样。

**to_tensor(self,buffer)**：转换数组维度。从(frame_count, height, width, 3)转换

为(3, frame_count, height, width)。

normalize(self,buffer)：标准化。

**C3D_model.py:**

此 python 文件实现了 C3D 网络。C3D 网络共有 8 个卷积层、4 个池化层、两个全连接层和一个 softmax 输出层。所有 3D 卷积核均为 3×3×3(d x k x k ,d 为时间深度)，步长为 1×1×1。为了在早期阶段保留更多的时间信息，设置 pool1 核大小为 1×2×2、步长 1×2×2。

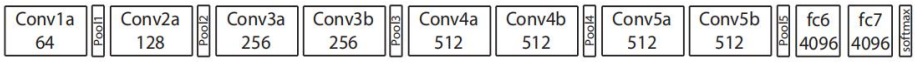



图 3 C3D 网络结构

```
1.          self.conv1 = nn.Conv3d(3, 64, kernel_size=(3, 3, 3), padding=(1, 1, 1))
2.          self.pool1 = nn.MaxPool3d(kernel_size=(1, 2, 2), stride=(1, 2, 2))
3.
4.          self.conv2 = nn.Conv3d(64, 128, kernel_size=(3, 3, 3), padding=(1, 1, 1))
5.          self.pool2 = nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2))
6.
7.          self.conv3a = nn.Conv3d(128, 256, kernel_size=(3, 3, 3), padding=(1, 1, 1))
8.          self.conv3b = nn.Conv3d(256, 256, kernel_size=(3, 3, 3), padding=(1, 1, 1))
9.          self.pool3 = nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2))
10.
11.         self.conv4a = nn.Conv3d(256, 512, kernel_size=(3, 3, 3), padding=(1, 1, 1))
12.         self.conv4b = nn.Conv3d(512, 512, kernel_size=(3, 3, 3), padding=(1, 1, 1))
13.         self.pool4 = nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2))
14.
15.         self.conv5a = nn.Conv3d(512, 512, kernel_size=(3, 3, 3), padding=(1, 1, 1))
16.         self.conv5b = nn.Conv3d(512, 512, kernel_size=(3, 3, 3), padding=(1, 1, 1))
17.         self.pool5 = nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2), padding=(0,
    1, 1))
18.
19.         self.fc6 = nn.Linear(8192, 4096)
20.         self.fc7 = nn.Linear(4096, 4096)
21.         self.fc8 = nn.Linear(4096, num_classes)
22.
23.         self.dropout = nn.Dropout(p=0.5)
24.
25.         self.relu = nn.ReLU()
```

**R2Plus1D_model.py:**

此 python 文件实现了 R2Plus1D 网络。其中 SpatioTemporalResLayer 类用于构建一个时空残差层，这个残差层由多个相同输出大小的重复块堆叠在一起组成。其中 SpatioTemporalConv 类，对由具有不同空间和时间轴的多个输入平面组成的输入信号应用分层 3D 卷积。通过在空间轴上执行 2D 卷积来生成中间子空间，然后在时间轴上执行 1D 卷积以生成最终输出。该类输入参数有是否为第一层卷积。如果这是第一层卷

积层，则分别使用两个 2D 卷积层处理空间和时间维度，然后进行批归一化和 ReLU 激活。如果这不是第一层卷积层，则仅使用一个 2D 卷积层处理空间维度，然后进行批归一化和 ReLU 激活。

然后通过 R2Plus1DNet 将这些层组装起来，形成 R2Plus1D 网络。

在 R2Plus1DNet 类中，self.conv1 是一个 3 通道、64 维的卷积层，使用 7x7 的卷积核进行卷积操作，步长为 2,填充为 3。self.conv2 是一个 64 维的卷积层，输出与 self.conv1 相同大小，不需要下采样。卷积核大小为 3x3x3。self.conv3 到 self.conv5 是三个残差层，每个残差层的通道数在前一层的基础上翻倍，同时在第一个块内进行下采样。

```
1.    self.conv1 = SpatioTemporalConv(3, 64, (1, 7, 7), stride=(1, 2, 2), padding=(0, 3, 3),
      first_conv=True)
2.    self.conv2 = SpatioTemporalResLayer(64, 64, 3, layer_sizes[0], block_type=block_type)
3.    self.conv3 = SpatioTemporalResLayer(64, 128, 3, layer_sizes[1], block_type=block_type,
      downsample=True)
4.    self.conv4 = SpatioTemporalResLayer(128, 256, 3, layer_sizes[2], block_type=block_typ
      e, downsample=True)
5.    self.conv5 = SpatioTemporalResLayer(256, 512, 3, layer_sizes[3], block_type=block_typ
      e, downsample=True)
6.    self.pool = nn.AdaptiveAvgPool3d(1)
```

在 R2Plus1DClassifier 中，将具有 512 维输入数据映射到具有 num_classes 维输出数据的空间，以完成分类。

```
1.    self.res2plus1d = R2Plus1DNet(layer_sizes, block_type)
2.    self.linear = nn.Linear(512, num_classes)
```

**R3D_model.py:**

此 python 实现了 R3D 网络。其中 SpatioTemporalConv 是一个空间-时间卷积层，用于对具有不同空间和时间轴的多个输入平面组成的输入信号进行卷积操作。SpatioTemporalResLayer 是一个空间-时间残差层，用于构建 ResNet 网络中的单个层。

通过 R3DNet 将这些层组装起来。

```
1.    self.conv1 = SpatioTemporalConv(3, 64, [3, 7, 7], stride=[1, 2, 2], padding=[1, 3, 3])
2.    self.conv2 = SpatioTemporalResLayer(64, 64, 3, layer_sizes[0], block_type=block_type)
3.    self.conv3 = SpatioTemporalResLayer(64, 128, 3, layer_sizes[1], block_type=block_type,
      downsample=True)
4.    self.conv4 = SpatioTemporalResLayer(128, 256, 3, layer_sizes[2], block_type=block_type,
      downsample=True)
5.    self.conv5 = SpatioTemporalResLayer(256, 512, 3, layer_sizes[3], block_type=block_type,
      downsample=True)
6.    self.pool = nn.AdaptiveAvgPool3d(1)
```

在 R3DClassifier 中，将具有 512 维输入数据映射到具有 num_classes 维输出数据的空间，以完成分类。

```
1.    self.res3d = R3DNet(layer_sizes, block_type)
2.    self.linear = nn.Linear(512, num_classes)
```

**train.py:**

该 python 文件将总数据随机按照 8:2 的比例分为训练集和测试集，进行 n 个周期的训练，得到相应的评价指标。使用 SummaryWriter 将每个周期的评价指标写入日志中，通过 tensorboard 进行可视化操作。在进行一定周期的训练后，保存模型。

该训练文件中使用了 SGD 优化器和 StepLR 学习率调度器。SGD 优化器使用随机梯度下降算法来最小化损失函数，其中 lr 参数设置学习率，momentum 参数设置动量大小，weight_decay 参数设置权重衰减系数。StepLR 学习率调度器用于在训练过程中动态调整学习率。它会在每个 step_size 时间间隔内将学习率乘以 gamma 因子，从而使模型在训练初期快速收敛，然后逐渐降低学习率，以便更好地适应数据分布的变化。

**test.py:**

该 python 文件使用 train.py 保存的模型对数据集进行测试，得到其评价指标。

**cross_validation.py:**

该 python 文件实现了 k 折交叉验证，并得到其评价指标。其中 k 为测试人数，在本数据集中为 25 人。同 train.py 文件，该程序使用了 SGD 优化器和 StepLR 学习率调度器。

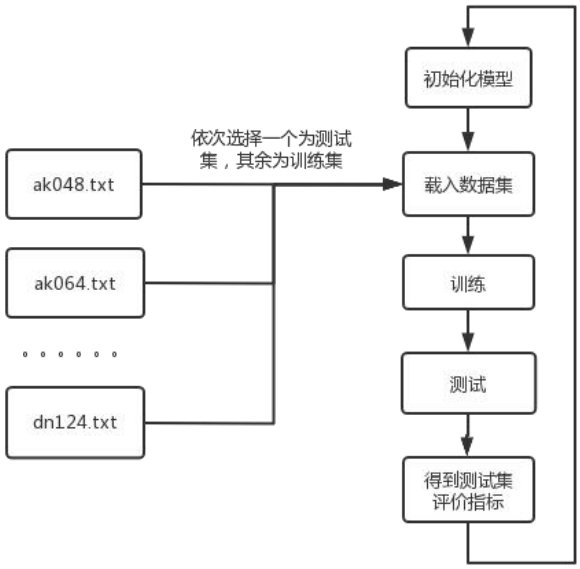



图 4 交叉验证代码实现流程

### 1.3 输入/输出设计

系统的输入为视频帧，系统的输出是图像的分类结果。根据分类结果与标签得到相应的评价指标。

# 2. 软件实施与实验运行

## 2.1 软件系统实施

**train.py 训练参数设置：**

| | C3D | R2Plus1D | R3D |
|---|---|---|---|

| num_epoch(周期数) | 50 | 50 | 50 |
|---|---|---|---|
| Lr(初始学习率) | 1e-4 | 1e-3 | 1e-3 |
| SGD momentum(动量因子) | 0.9 | 0.9 | 0.9 |
| weight_decay(L2 正则化系数) | 5e-4 | 5e-4 | 5e-4 |
| step_size(学习率调整周期) | 5 | 5 | 5 |
| gamma(降低因子) | 0.5 | 0.5 | 0.5 |

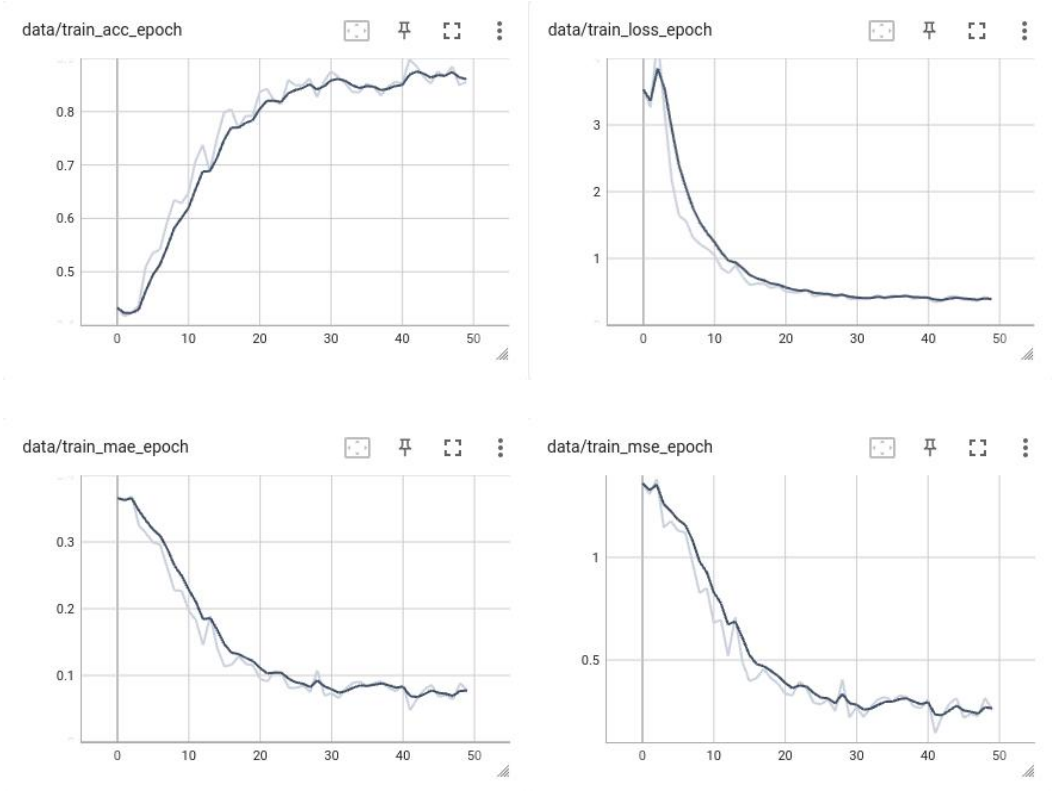
**cross_validation.py 参数设置：**

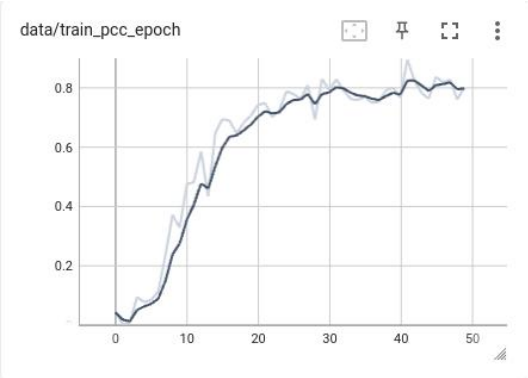
| | C3D | R2Plus1D | R3D |
|---|---|---|---|
| Lr(初始学习率) | 1e-4 | 1e-3 | 1e-3 |
| SGD momentum(动量因子) | 0.9 | 0.9 | 0.9 |
| weight_decay(L2 正则化系数) | 5e-4 | 5e-4 | 5e-4 |
| step_size(学习率调整周期) | 5 | 5 | 5 |
| gamma(降低因子) | 0.5 | 0.5 | 0.5 |

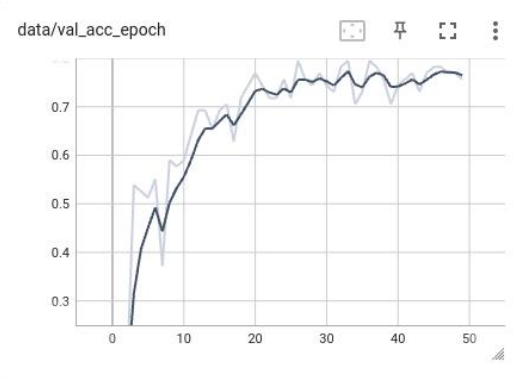
## 2.2 数据集训练结果分析

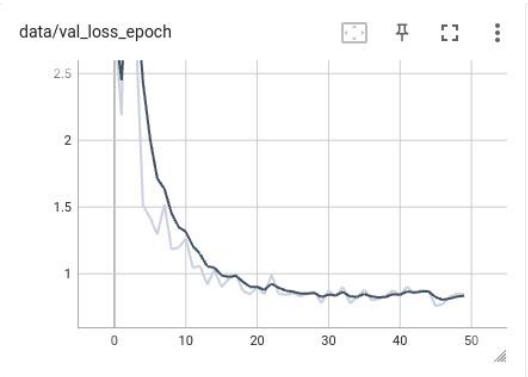
运行 train.py，通过 tensorboard 进行数据可视化。

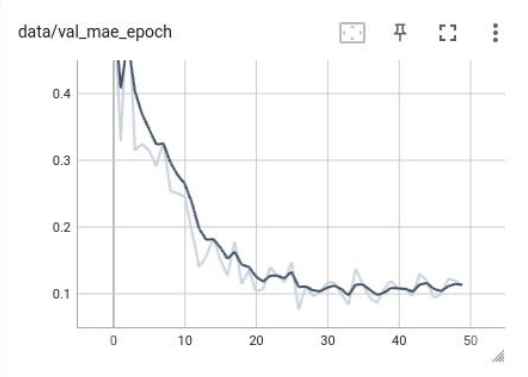
**C3D:**

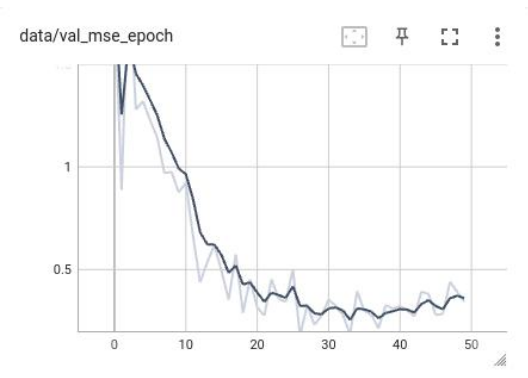


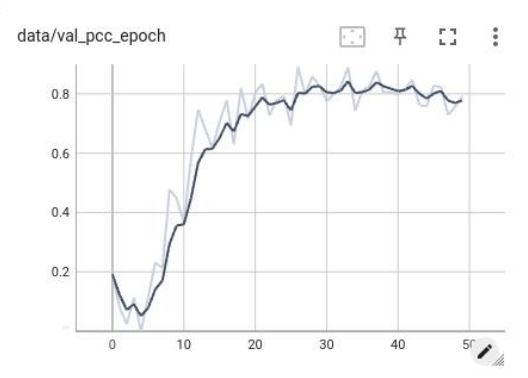

data/train_pcc_epoch
0.8
0.6
0.4
0.2
0
10
20
30
40
50


data/val_acc_epoch
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50

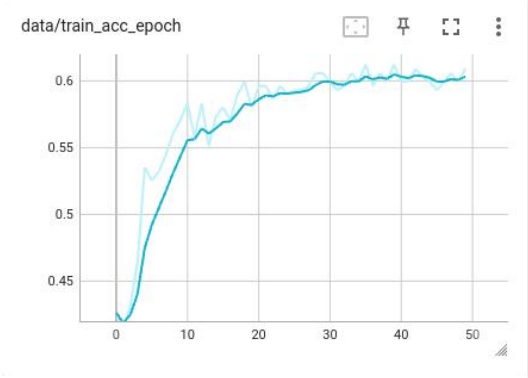

data/val_loss_epoch
2.5
2
1.5
1
0
10
20
30
40
50

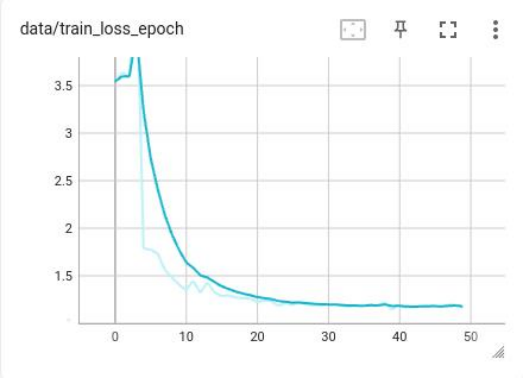

data/val_mae_epoch
0.4
0.3
0.2
0.1
0
10
20
30
40
50

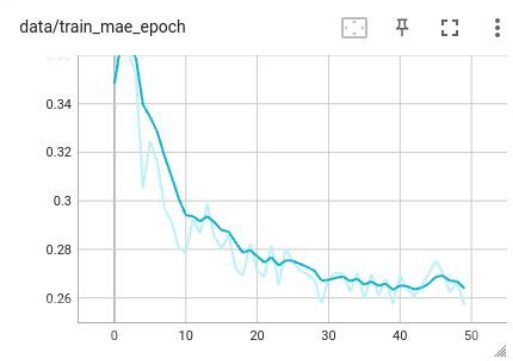

data/val_mse_epoch
1
0.5
0
10
20
30
40
50

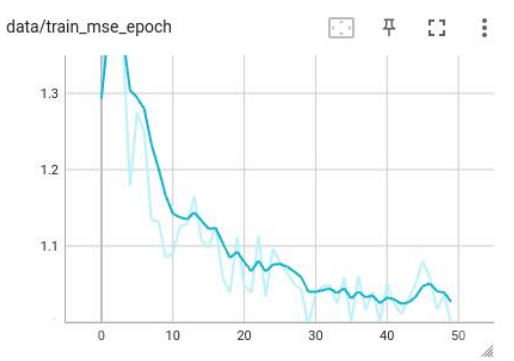

data/val_pcc_epoch
0.8
0.6
0.4
0.2
0
10
20
30
40
50

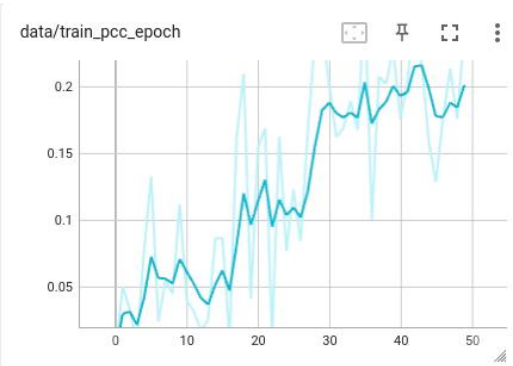
**R2Plus1D：**

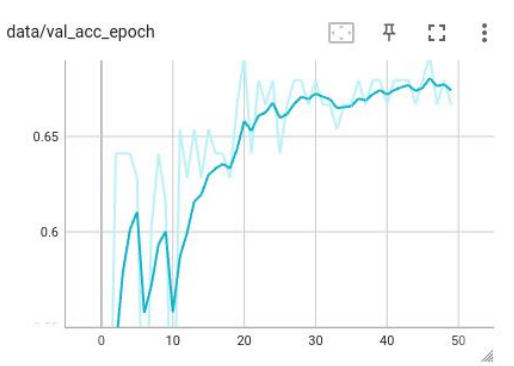

data/train_acc_epoch
0.6
0.55
0.5
0.45
0
10
20
30
40
50

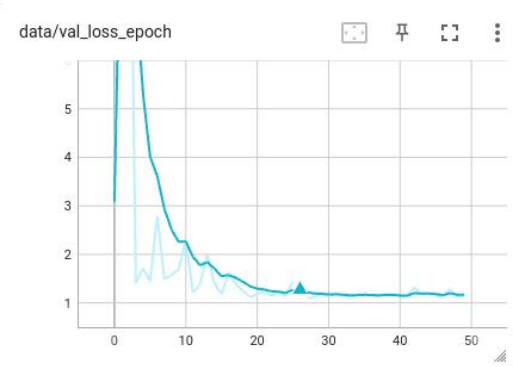

data/train_loss_epoch
3.5
3
2.5
2
1.5
0
10
20
30
40
50

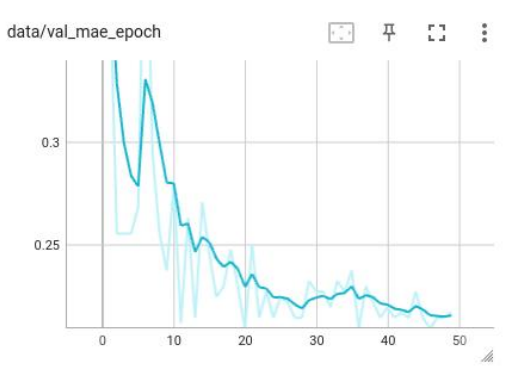

data/train_mae_epoch
0.34
0.32
0.3
0.28
0.26
0
10
20
30
40
50

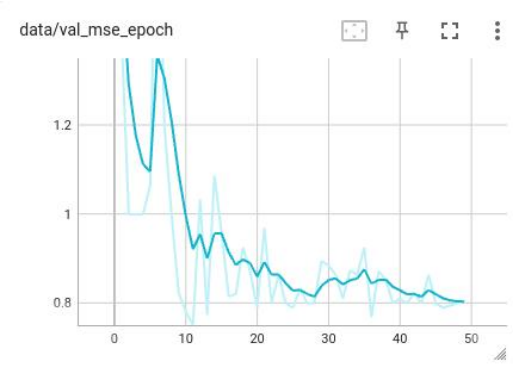

data/train_mse_epoch
1.3
1.2
1.1
0
10
20
30
40
50

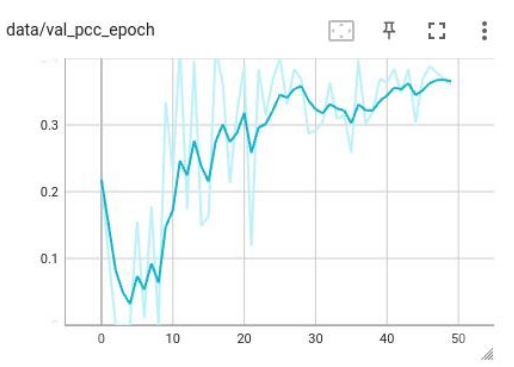

data/train_pcc_epoch
0.2
0.15
0.1
0.05
0
10
20
30
40
50


data/val_acc_epoch
0.65
0.6
0
10
20
30
40
50

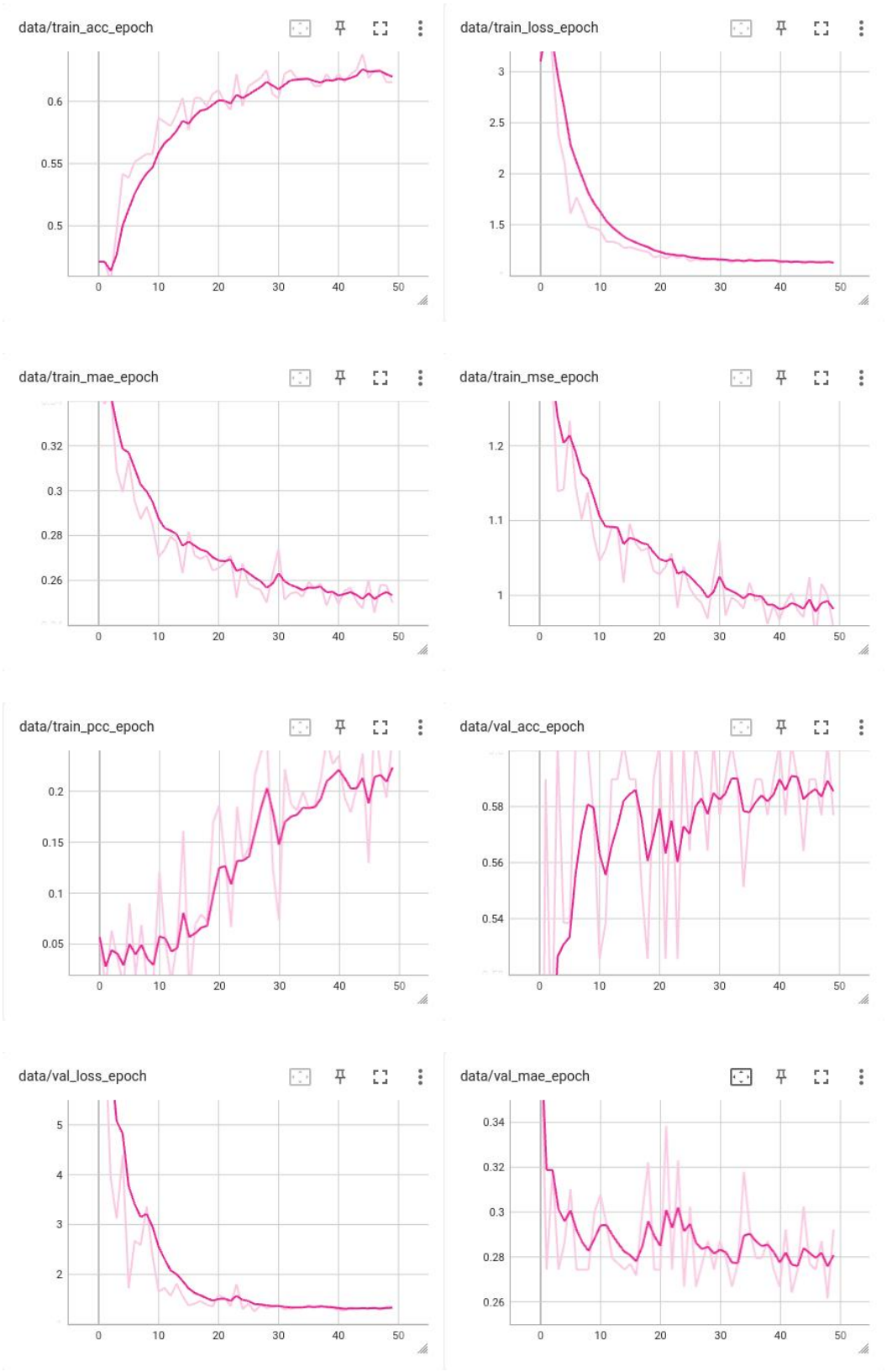

data/val_loss_epoch
5
4
3
2
1
0
10
20
30
40
50

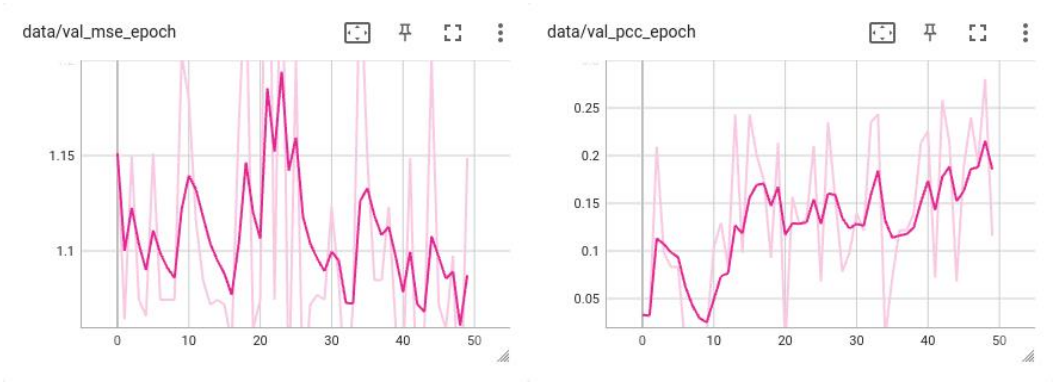

data/val_mae_epoch
0.3
0.25
0
10
20
30
40
50


data/val_mse_epoch
1.2
1
0.8
0
10
20
30
40
50


data/val_pcc_epoch
0.3
0.2
0.1
0
10
20
30
40
50

**R3D:**


data/train_acc_epoch
0.6
0.55
0.5
0 10 20 30 40 50
data/train_loss_epoch
3
2.5
2
1.5
0 10 20 30 40 50
data/train_mae_epoch
0.32
0.3
0.28
0.26
0 10 20 30 40 50
data/train_mse_epoch
1.2
1.1
1
0 10 20 30 40 50
data/train_pcc_epoch
0.2
0.15
0.1
0.05
0 10 20 30 40 50
data/val_acc_epoch
0.58
0.56
0.54
0 10 20 30 40 50
data/val_loss_epoch
5
4
3
2
0 10 20 30 40 50
data/val_mae_epoch
0.34
0.32
0.3
0.28
0.26
0 10 20 30 40 50




对上述数据进行分析，发现 C3D 模型各项评价指标变化得较为平滑，训练集 ACC 在第 50 个 epoch 时达到了 90%，测试集 ACC 在第 50 个 epoch 时达到了 85%，各项指标数据表现得较为优越；R2Plus1D 和 R3D 模型各项评价指标变化得比较震荡，各项指标数据不如 C3D。我猜测原因可能是在载入 C3D 模型时加载了预训练的模型，而 R2Plus1D 和 R3D 没有加载预训练模型，使用的为随机权重，故效果并不是很好。

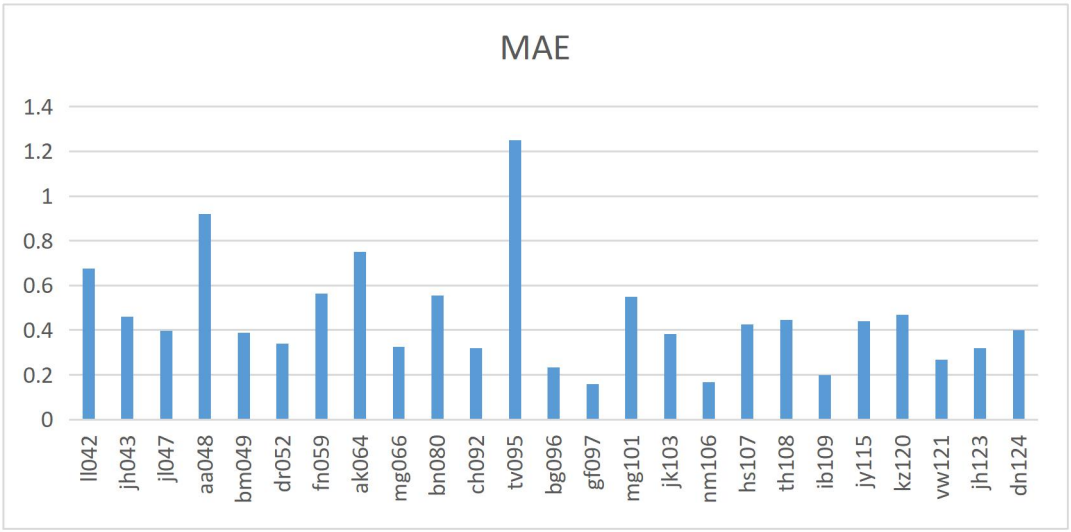
## 2.3 数据集 25 折交叉验证结果分析

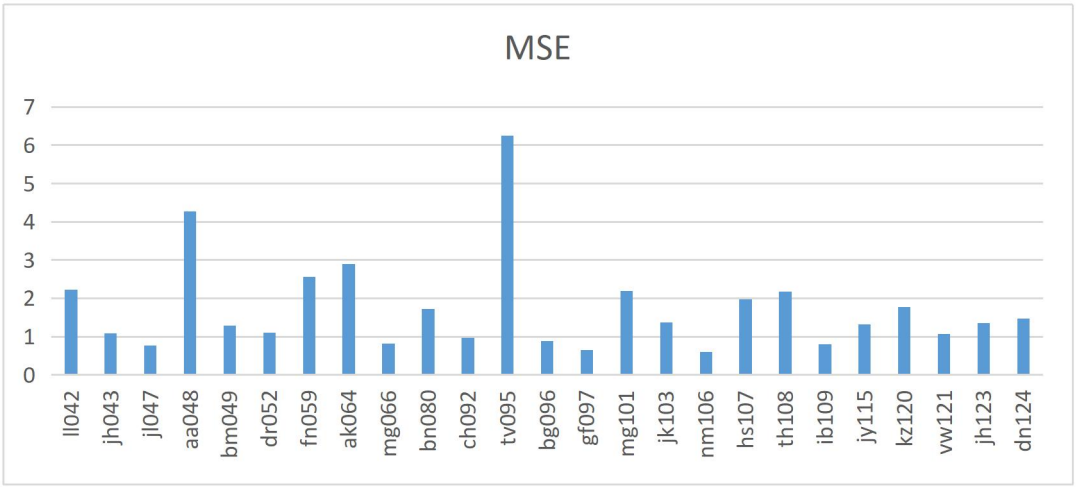
运行 cross_validation.py。

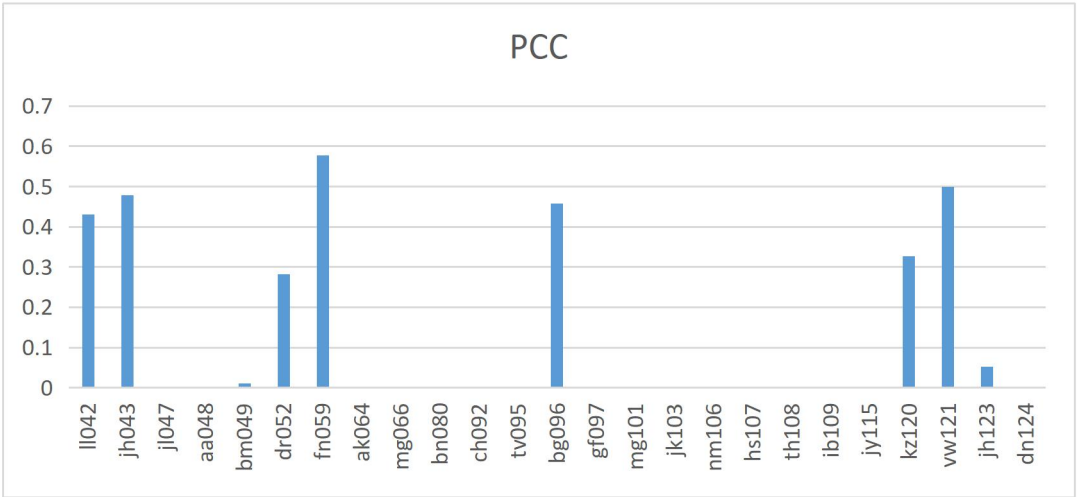
**C3D：**

| | MAE | MSE | PCC |
|---|---|---|---|
| ll042 | 0.675 | 2.225 | 0.431182782 |
| jh043 | 0.459259259 | 1.081481481 | 0.478091444 |
| jl047 | 0.396428571 | 0.760714286 | |
| aa048 | 0.92 | 4.28 | |
| bm049 | 0.3875 | 1.2875 | 0.011549315 |
| dr052 | 0.34 | 1.1 | 0.281718085 |
| fn059 | 0.5625 | 2.5625 | 0.577350269 |
| ak064 | 0.75 | 2.9 | 2.78E-17 |
| mg066 | 0.325 | 0.825 | |
| bn080 | 0.553571429 | 1.725 | |
| ch092 | 0.32 | 0.96 | |
| tv095 | 1.25 | 6.25 | |
| bg096 | 0.233333333 | 0.877777778 | 0.457539682 |
| gf097 | 0.157142857 | 0.642857143 | |
| mg101 | 0.548148148 | 2.192592593 | |
| jk103 | 0.38125 | 1.3625 | |
| nm106 | 0.166666667 | 0.604761905 | |
| hs107 | 0.425 | 1.975 | |
| th108 | 0.446153846 | 2.169230769 | |
| ib109 | 0.2 | 0.8 | |
| jy115 | 0.44 | 1.32 | |

| | | | |
|---|---|---|---|
| kz120 | 0.467857143 | 1.775 | 0.326164037 |
| vw121 | 0.266666667 | 1.066666667 | 0.5 |
| jh123 | 0.32 | 1.36 | 0.051571062 |
| dn124 | 0.4 | 1.464285714 | |
| 平均值 | 0.455659117 | 1.742714733 | 0.311516668 |

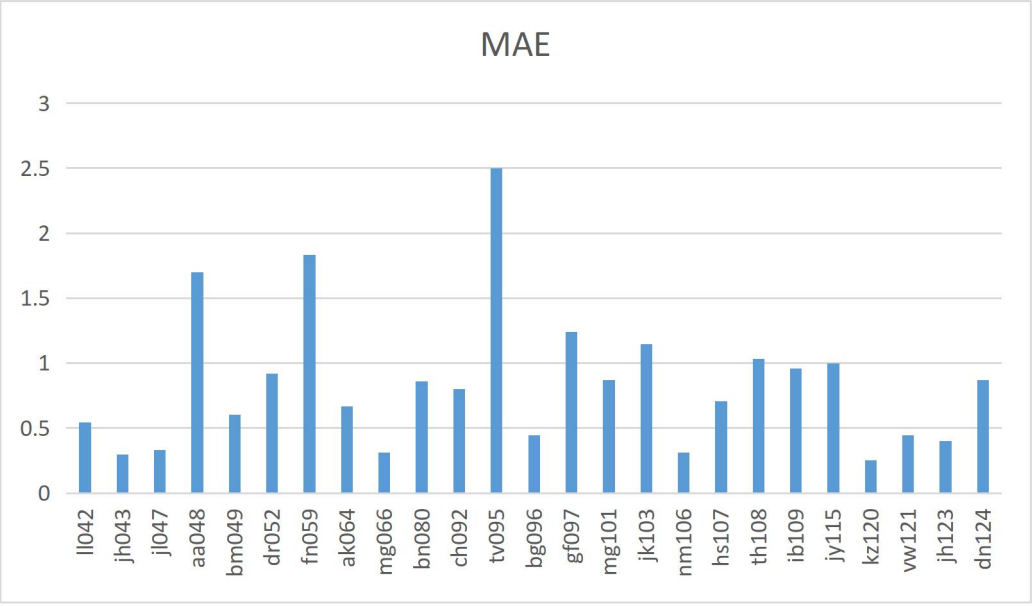


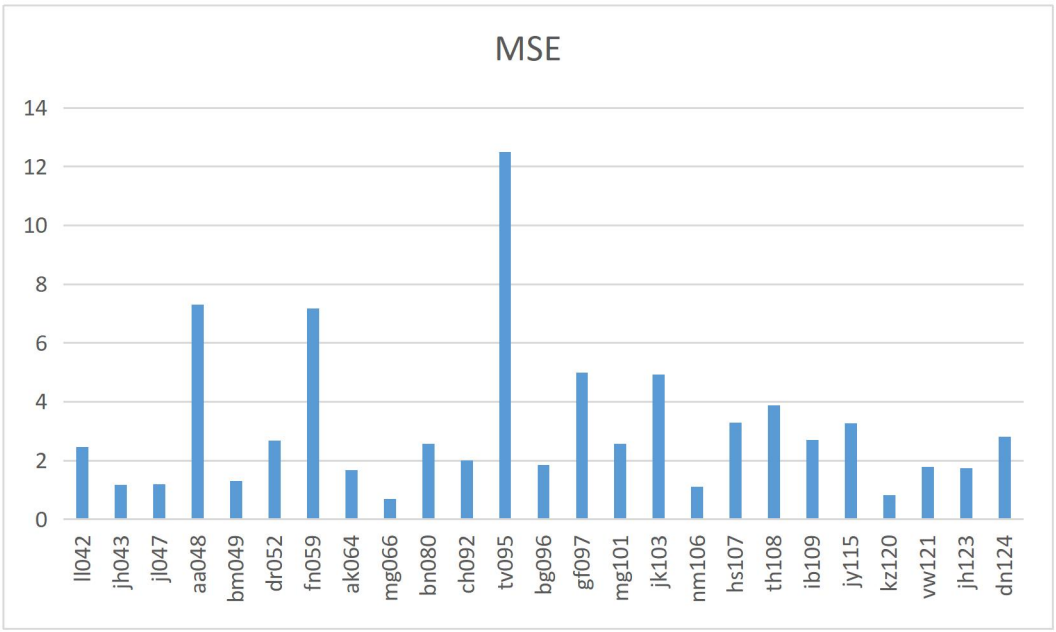



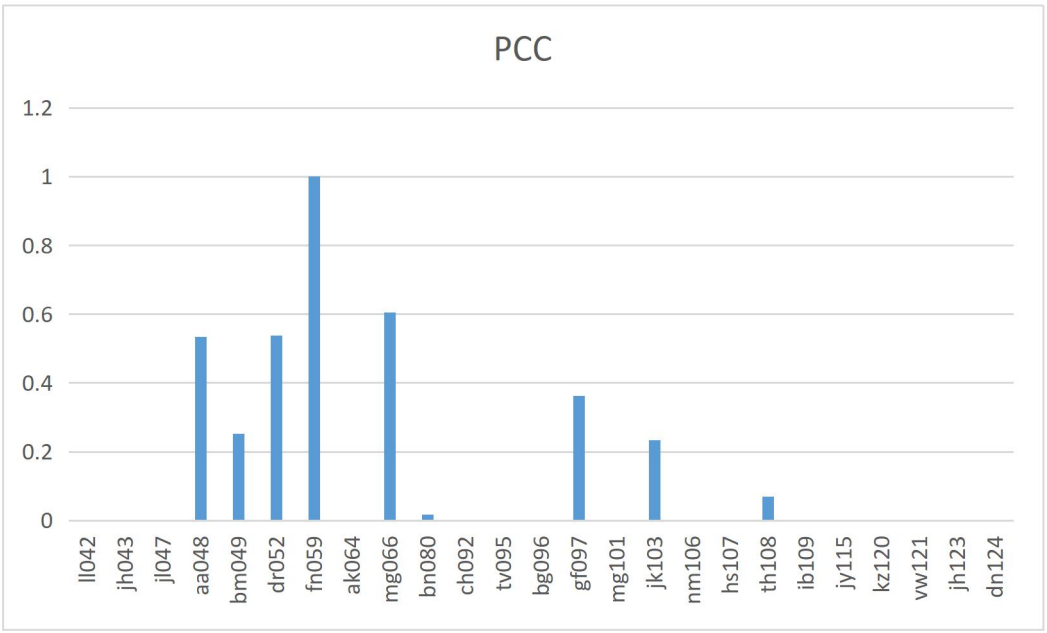




**R2Plus1D：**

| | MAE | MSE | PCC |
|---|---|---|---|
| ll042 | 0.541666667 | 2.458333333 | |
| jh043 | 0.296296296 | 1.185185185 | |
| jl047 | 0.333333333 | 1.19047619 | |
| aa048 | 1.7 | 7.3 | 0.534522484 |
| bm049 | 0.604166667 | 1.3125 | 0.252407794 |
| dr052 | 0.916666667 | 2.683333333 | 0.537914354 |
| fn059 | 1.833333333 | 7.166666667 | 1 |
| ak064 | 0.666666667 | 1.666666667 | |
| mg066 | 0.3125 | 0.6875 | 0.605365319 |
| bn080 | 0.857142857 | 2.571428571 | 0.018077538 |
| ch092 | 0.8 | 2 | |
| tv095 | 2.5 | 12.5 | |
| bg096 | 0.444444444 | 1.851851852 | |
| gf097 | 1.238095238 | 5 | 0.363410559 |
| mg101 | 0.87037037 | 2.574074074 | |
| jk103 | 1.145833333 | 4.927083333 | 0.233723197 |
| nm106 | 0.30952381 | 1.119047619 | |
| hs107 | 0.708333333 | 3.291666667 | |
| th108 | 1.032051282 | 3.878205128 | 0.070480911 |
| ib109 | 0.958333333 | 2.708333333 | |
| jy115 | 1 | 3.266666667 | |
| kz120 | 0.25 | 0.821428571 | |
| vw121 | 0.444444444 | 1.777777778 | |
| jh123 | 0.4 | 1.733333333 | |
| dn124 | 0.869047619 | 2.821428571 | |
| 平均值 | 0.841289988 | 3.139719475 | 0.401766906 |



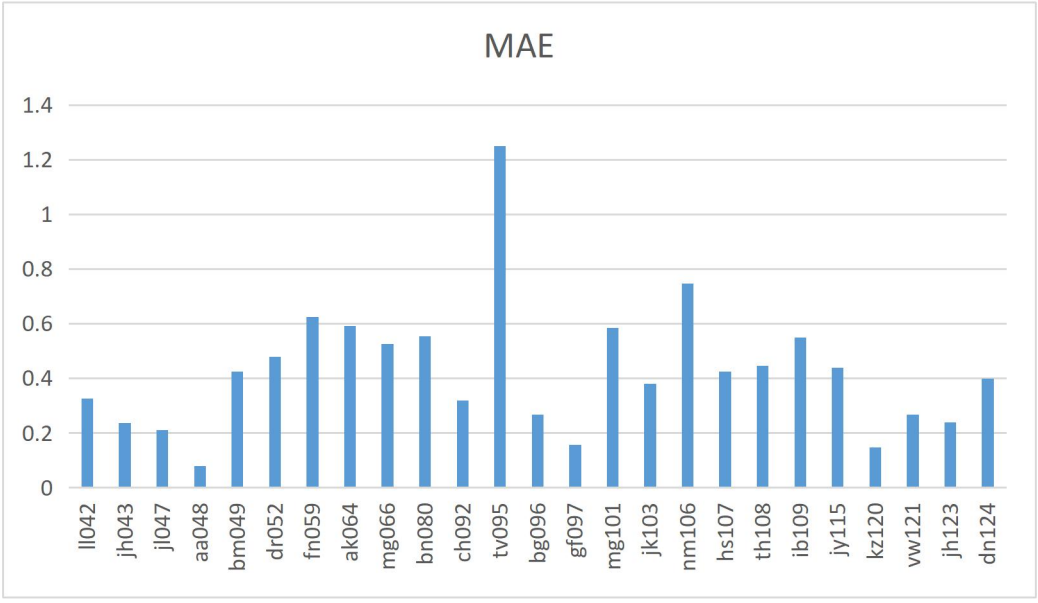


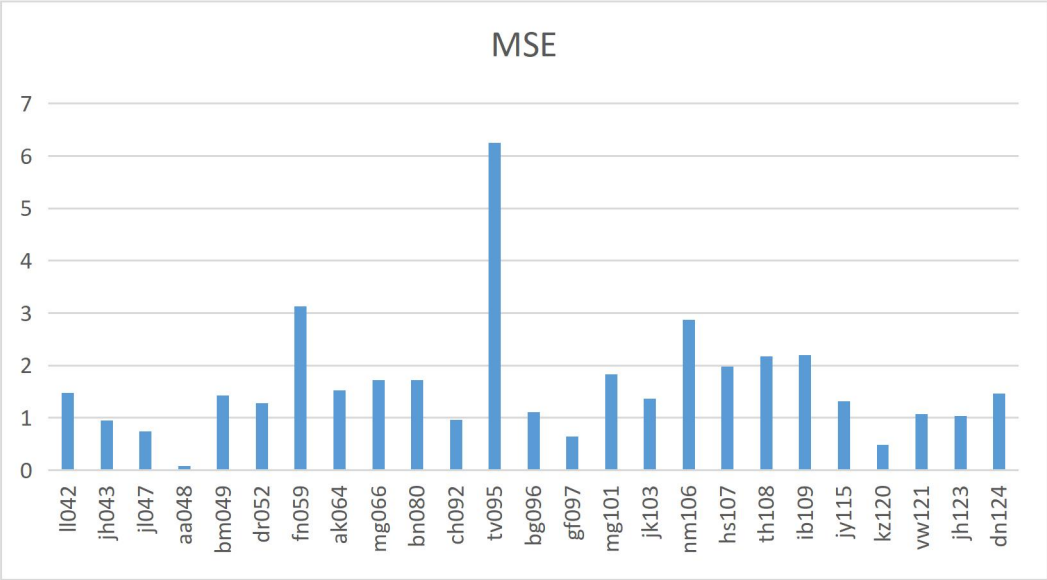



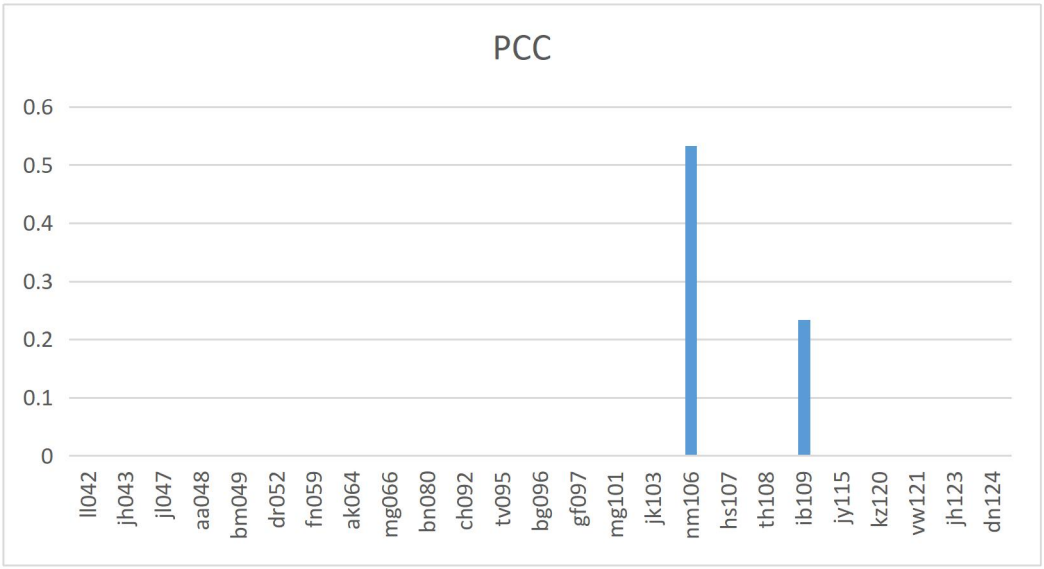

**R3D：**

| | MAE | MSE | PCC |
|---|---|---|---|
| 11042 | 0.325 | 1.475 | |
| jh043 | 0.237037037 | 0.948148148 | |
| j1047 | 0.210714286 | 0.746428571 | |
| aa048 | 0.08 | 0.08 | |
| bm049 | 0.425 | 1.425 | |
| dr052 | 0.48 | 1.28 | |
| fn059 | 0.625 | 3.125 | |
| ak064 | 0.591666667 | 1.525 | |
| mg066 | 0.525 | 1.725 | |

| | | | |
|---|---|---|---|
| bn080 | 0.553571429 | 1.725 | |
| ch092 | 0.32 | 0.96 | |
| tv095 | 1.25 | 6.25 | |
| bg096 | 0.266666667 | 1.111111111 | |
| gf097 | 0.157142857 | 0.642857143 | |
| mg101 | 0.585185185 | 1.82962963 | |
| jk103 | 0.38125 | 1.3625 | |
| nm106 | 0.747619048 | 2.871428571 | 0.533333333 |
| hs107 | 0.425 | 1.975 | |
| th108 | 0.446153846 | 2.169230769 | |
| ib109 | 0.55 | 2.2 | 0.233549683 |
| jy115 | 0.44 | 1.32 | |
| kz120 | 0.146428571 | 0.489285714 | |
| vw121 | 0.266666667 | 1.066666667 | |
| jh123 | 0.24 | 1.04 | |
| dn124 | 0.4 | 1.464285714 | |
| 平均值 | 0.42700409 | 1.632262882 | 0.383441508 |






由于我们所使用的数据集较不平衡，无疼痛图像 PSPI 评分标签的数量远高于其他标签，导致在计算 PCC 的过程中很容易出现分母为 0 得到 nan 值（预测值都为一个数或者标签都为一个数也会导致此结果），很难获取有效的 PCC 来进行指标的评价，所以我们主要以 MAE 和 MSE 为主要指标进行分析。

根据 C3D，R2Plus1D 和 R3D 的图表，发现 tv095 的 MAE 和 MSE 都偏高，对该样本进行分析，发现 tv095 的数据标签都为 5，即最高疼痛标签。由于数据集中最高疼痛的标签数量十分少，所以 tv095 的 MAE 和 MSE 偏高。

## 2.4 心得体会

在这次课设中，我学到了很多关于数据处理、模型建立和预测的知识。在课程设计的过程中，我需要使用 Python 编程语言来实现各种算法和模型，并对数据进行处理和分析。通过这个过程，我不仅掌握了编程技能，还深入了解了机器学习和模式识别的基本概念和应用场景。

首先，我学习了数据预处理的基础知识。在实际应用中，我们需要对原始数据进行清洗、标准化、归一化等处理，以确保数据的准确性和一致性。此外，我们还需要对数据进行特征

提取和选择，以便更好地应用于模型训练和预测。在这个过程中，我学会了如何使用 Python 中的 torch 库来处理和分析数据。

其次，我学习了常用的视频分类模型，比如 C3D，R2Plus1D 和 R3D。这些模型都是基于深度学习的视频分类算法，它们在处理大规模视频数据时表现出色。其中，C3D 是一种卷积神经网络模型，通过多层卷积和池化操作提取视频特征；R2Plus1D 是一种循环神经网络模型，通过将视频帧作为时间序列输入，实现对视频内容的分类；R3D 则是一种基于残差网络的模型，通过引入残差连接来解决深层网络中的梯度消失问题。

在课程设计中，我使用这些模型对视频数据进行了分类，并比较了它们的性能。通过实验结果，我发现这些模型在处理不同类型的视频数据时表现不同，需要根据具体情况进行选择和调参。同时，我也发现了一些常见的问题，如过拟合、欠拟合等，需要采取相应的措施来解决。

由于这次课设时间紧张，我并没有尝试更多的模型和算法，实属遗憾，同样我也缺少创新，没有提出新的模型（无论效果）。这些都是我的不足之处。不过，我通过这次课程设计，也发现了自己在编程和数据分析方面的不足之处。例如，在处理大规模数据时，我的代码效率较低，需要进行优化；在模型选择和调参方面，我还需要更多的实践和经验。

因此，我会继续学习和探索机器学习和模式识别领域的知识，不断提升自己的技能和能力。同时，我也会尝试提出新的模型和算法，为实际应用做出贡献。