

Report for CMR Orientation Adjust

一、工作简述

1. 模型训练
2. UI升级

二、具体工作

1. 复现模型
2. UI升级
 - 2.1 程序打包：
 - 2.2 模型更换：
 - 2.3 图像类型扩充：
 - 2.4 图像可视化调整：
 - 2.5 模型自由切换：
 - 2.6 图片预测优化：

三、细节实现--模型部分

1. 论文理解
2. 网络搭建
3. 模型训练与参数选择
4. 准确率

四、细节实现--UI软件部分

1. 打包成可执行程序并配备安装包
2. 模型替换
3. 图像类别扩充
4. 页数调整
5. 模型切换
6. 图片预测

一、工作简述





1. 模型训练

在详细阅读、理解了张可学姐的文章《Recognition and standardization of cardiac MRI orientation via multi-tasking learning and deep neural network》、以及弄懂了所用代码之后。我用pytorch重新复现了学姐搭建的神经网络进行训练。通过对C0 T2 LGE三个数据集分别进行800、400、800轮次的训练之后，得到了在总数据集上准确率达到99.4%的模型(如下表)。

C0			T2			LGE		
Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
556/55 6	67/70	68/70	556/55 6	67/70	70/70	556/55 6	70/70	65/70
Total Accuracy: 99.4%								

2. UI升级

在此之上，我将学姐所做的UI软件《CMR-Orientation-Adjust-Tool》其中内部基于tensorflow搭建模型替换成了我基于pytorch所训练的模型，并做了一些细节上的优化，进而把它打包程序《CMR-Orientation-Adjust-Tool2.exe》，还为用户增加了安装包，以便下载和安装。同时，通过模型的替换，成功将软件应用程序从700M的大小优化到400M（如下图）。

	CMR Orientation Adjust Tool.exe	2022/6/22 21:13	应用程序	744,975 KB
	Setup for CMR Orientation Adjust To...	2022/6/22 21:32	应用程序	732,758 KB
	Set Up For CMR Orientation Adjust T...	2022/9/2 17:10	应用程序	444,249 KB
	CMR Orientation Adjust Tool2.0.exe	2022/9/2 16:44	应用程序	405,645 KB

二、具体工作

1.复现模型

复现了调整图像方向的CNN模型，总数据集准确率达到99.4%，在训练集、验证集、测试集准确率分别达到100%、97.1%、96.7%。

2.UI升级

将UI工具"Orientation-Adjust-Tool"进行性能升级至2.0版本。包括以下六方面：

2.1 程序打包：

将py文件打包成了exe文件，并且配备了安装包，以便于用户下载、安装。

2.2 模型更换：

将之前基于tensorflow神经网络的模型加载、预测部分，替换为基于pytorch的神经网络模型。因为tensorflow库相较于torch库很大，之前1.0版本得到的文件有700M，2.0版本只有400M。（在打包应用程序时已经使用pipenv纯化库环境，所以大小已经达到python打包exe的最小程度）

2.3 图像类型扩充：

在原有仅支持nii nii.gz mha文件的基础上，添加了png，jpg图像，和纯二维图像（png与jpg在用SimpleITK读入时依然是三维图像，所以并不属于这里所说的纯二维图像）

2.4 图像可视化调整：

在默认页数上作出微调，保证用户在打开图片时看到的是最中间的切片图像。

2.5 模型自由切换：

在1.0版本中，如果选择其他模型会让图片进行经验式的旋转，不能实际切换模型。在2.0版本中，通过对内部类参数的调整，可以实现用户自主切换C0,T2,LGE模型进行预测（依然会根据图片名称选择初始模型）。

2.6 图片预测优化：

在1.0版本中，对三维心脏核磁共振图像的方向预测，是把图像最短边切成多个切片，每个切片是一个二维图像，然后对每一个图像进行预测，最终取预测概率值最大的作为结果。在2.0版本中，采用投票式预测——每个切片图像生成一个预测结果，然后根据多数图像的预测方向决定最终的方向。之所以作出这种优化，用一个生动的比喻来形容，是因为当每一个基模型的准确率非常高时（本模型高达99%），共和制（少数服从多数）的决策显然要优于君主立宪制（预测结果只依赖与最强的学习器）。

三、细节实现——模型部分

1. 论文理解

由于心脏核磁共振图像的方向对于后续进一步处理非常关键，所以论文提出通过深度学习模型预测图片方向的方法。

论文提出最核心的思想是：先对三维图像进行切片、标准化，将切片后的二维图像放入训练好的CNN网络中进行方向识别预测，根据预测到的方向进行调整和保存。

因此，根据这一思想，学姐制作了两个文件——一个用来批量调整图像方向；另外一个用于户交互软件，便于可视化和单个图片方向预测和调整。

理解了论文之后，我复现的思路也分为两个步骤：

第一个是对CNN模型的重建，尝试提升准确率；

第二个是把UI的py文件改进成应用程序（EXE软件），并把它打包，以便于用户更方便的下载、安装和使用。

2. 网络搭建

代码见train.py。将图片划分为训练集、验证集和测试集，放入搭建好的CNN网络（如下代码所示）进行训练。

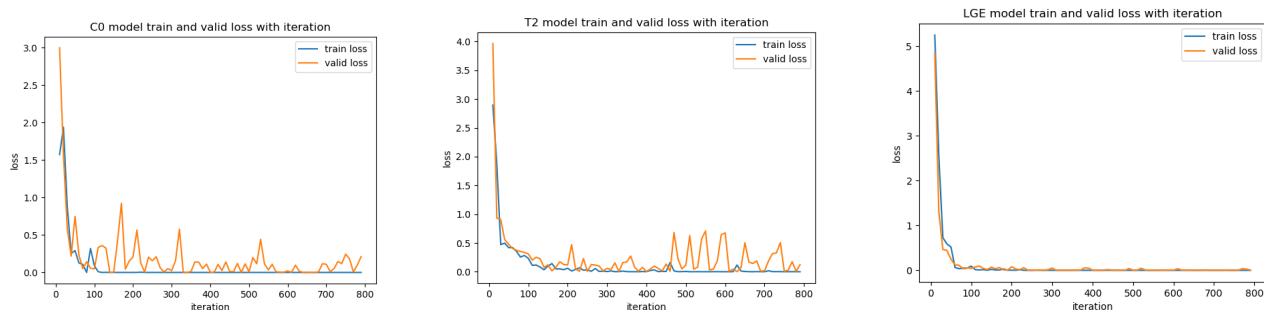
```

1 class cnn(nn.Module): # construction of netral network
2     def __init__(self):
3         super(cnn, self).__init__()
4         self.relu = nn.ReLU()
5         self.conv1 = nn.Sequential(
6             nn.Conv2d( # 1 224 224
7                 in_channels=1,
8                 out_channels=16,
9                 kernel_size=5,
10                stride=1,
11                padding=2 # if stride = 1 padding = (kernel_size - 1)/2
12            ),
13            nn.BatchNorm2d(16),
14            nn.ReLU(),
15            nn.MaxPool2d(kernel_size=2), # 16,128,128
16        )
17        # 16 224 224
18        self.conv2 = nn.Sequential( # 16,128,128
19            nn.Conv2d(16,32,5,1,2), # 32 128 128
20            nn.BatchNorm2d(32),
21            nn.ReLU(),
22            nn.MaxPool2d(kernel_size=2), # 32 64 64
23        )
24        #
25        self.conv3 = nn.Sequential(
26            nn.Conv2d(32,64,5,1,2), # 64 32 32
27            nn.BatchNorm2d(64),
28            nn.ReLU(),
29            nn.MaxPool2d(kernel_size=2), # 64 16 16
30        )
31        self.fc1 = nn.Linear(64*32*32, 64)
32        self.out= nn.Linear(64, 8)

```

3.模型训练与参数选择

将C0 T2 LEG分别进行训练。得到训练集、验证集损失函数随轮次（iteration）的变化(如下图)。



其中T2很明显在iter大于400时发生过拟合，所以最终分别三个模型确定轮次为800、400、400。

4.准确率

在训练好模型后，重新使用C0,T2,LGE模型在它们的训练集、测试集、验证集上进行预测，得到的准确率如下图。

```

-----calculating C0 model <800 iter> accuracy, please waiting...-----
C0 model 800 iter train set accuracy : 1.0
预测失败: 图片data_png/valid_data_C0/000/0_patient5_C0.png, 误预测为100!
预测失败: 图片data_png/valid_data_C0/001/2_patient26_C0.png, 误预测为000!
预测失败: 图片data_png/valid_data_C0/011/0_patient11_C0.png, 误预测为111!
C0 model 800 iter valid set accuracy : 0.9571428571428572
预测失败: 图片data_png/test_data_C0/100/2_patient27_C0.png, 误预测为000!
预测失败: 图片data_png/test_data_C0/111/4_patient6_C0.png, 误预测为011!
C0 model 800 iter test set accuracy : 0.9714285714285714

-----calculating T2 model <400 iter> accuracy, please waiting...-----
T2 model 400 iter train set accuracy : 1.0
预测失败: 图片data_png/valid_data_T2/010/0_patient35_T2.png, 误预测为011!
预测失败: 图片data_png/valid_data_T2/100/1_patient27_T2.png, 误预测为010!
预测失败: 图片data_png/valid_data_T2/101/0_patient4_T2.png, 误预测为001!
T2 model 400 iter valid set accuracy : 0.9571428571428572
T2 model 400 iter test set accuracy : 1.0

-----calculating LGE model <800 iter> accuracy, please waiting...-----
LGE model 800 iter train set accuracy : 1.0
LGE model 800 iter valid set accuracy : 1.0
预测失败: 图片data_png/test_data_LGE/001/0_patient27_LGE.png, 误预测为101!
预测失败: 图片data_png/test_data_LGE/011/3_patient1_LGE.png, 误预测为100!
预测失败: 图片data_png/test_data_LGE/011/4_patient1_LGE.png, 误预测为100!
预测失败: 图片data_png/test_data_LGE/110/0_patient38_LGE.png, 误预测为010!
预测失败: 图片data_png/test_data_LGE/110/0_patient4_LGE.png, 误预测为010!
LGE model 800 iter test set accuracy : 0.9285714285714286

```

可以将其整理为表格：

C0			T2			LGE		
Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
556/55 6	67/70	68/70	556/55 6	67/70	70/70	556/55 6	70/70	65/70
Total Accuracy: 99.4%								

四、细节实现--UI软件部分

1.打包成可执行程序并配备安装包

代码见main.py

使用pyinstaller制作应用程序，使用NSIS配备安装包。前者必须使用pipenv建立虚拟python环境，从而减少应用程序的文件大小。并且需要在py文件前面加入一段代码使得pytorch与pyinstaller兼容。在作出应用程序的demo后，在CMD下通过多次debug添加所需要的库，才能最终得到应用程序。

2.模型替换

具体代码见main.py中的auto类以及 MainWindow类。

需要将main.py中的预测部分进行修改，对auto_adjust中的predict函数进行较大的修改。同时将基于encoding-decoding模式的tensorflow读取数据模式调整成torch读取模型和数据的模式。

3.图像类别扩充

将jpg和png补充到可以打开的文件格式，并且允许用户输入二维矩阵转化成的图像（如下代码）。

```

1  ▾ if len(img.shape) == 2: # 如果是二维图像
2      std = img.std()
3  ▾      if std == 0.0: # 这里是为了防止std为0
4          std = 1
5          img_st = (img - img.mean()) / std # 将图像像素点进行 (0, 1) 标准化
6          img_tensor = transform(Image.fromarray(img_st)) # 要变成Image类型才能后
续用transform转换
7          img_tensor4d = img_tensor.unsqueeze(0)
8          predict = float(torch.max(model(img_tensor4d), 1).indices) # 把标准化好
的图片放进神经网络中进行预测
9          self.direct = self.directs[int(predict)]
10
11 ▾ else: # 如果是三维图像
12     # img = img[:, :, 0] # 方法一: 如果是png的三维图像, 取第一个slice图像作为目标
13     slice_num = img.shape[2] # 方法二: 投票法
14     list_img = list()
15 ▾     for slice_id in range(slice_num):
16         img_slice = img[:, :, slice_id]
17         std = img_slice.std()
18 ▾         if std == 0.0: # 这里是为了防止std为0
19             std = 1
20             img_st = (img_slice - img_slice.mean()) / std # 将图像像素点进行
(0, 1) 标准化
21             img_tensor = transform(Image.fromarray(img_st)).unsqueeze(0) # 要
变成Image类型才能后续用transform转换
22             list_img.append(img_tensor)
23             img_tensor4d = torch.cat(list_img, dim=0)
24             predict = torch.max(model(img_tensor4d), 1).indices # 各个图片的分类预测
25             predict_result = int(max(set(predict), key=list(predict).count)) # 根据
投票结果选出最终方向
26             self.direct = self.directs[predict_result]

```

4.页数调整

调整了openfile函数中的imgIndex, 使得用户总能看到处在中间的图像切片:


```

1 ▾ if self.imgDim >= 3:
2     self.imgIndex = int(self.imgDim / 2 - 1) # imgIndex相当于最短边的中点位置
        (z/2) , (x,y,z/2) 最能代表这个立体图像
3 ▾ else:
4     self.imgIndex = int(self.imgDim / 2 - 0.1)

```

5.模型切换

重新书写了setclass函数，使得用户在下拉框中切换C0 T2 LGE时，确实可以改变模型：

```

1 ▾ def setClass(self, index):
2 ▾     try:
3         name = self.classItems.itemText(index)
4         if name == 'C0':
5             self.model = self.model_C0
6         if name == 'T2':
7             self.model = self.model_T2
8         if name == 'LGE':
9             self.model = self.model_LGE
10 ▾     except:
11         QMessageBox.information(self, "Tip", "Model Changing Failed! Please check your operation!")

```

6.图片预测

把基于最大概率的预测改为投票式预测：

```

1 def predict(self, img, model): # 改动: 这里img只是 二维图像经过标准化之后延展成的
    四维的Tensor
2     transform = transforms.Compose([ # transform to figure, for further p
    assing to nn
3         transforms.Resize((256, 256)),
4         transforms.ToTensor(), # ToTensor会给灰度图像自动增添一个维度
5     ])
6
7     if len(img.shape) == 2: # 如果是二维图像
8         std = img.std()
9         if std == 0.0: # 这里是为了防止std为0
10             std = 1
11         img_st = (img - img.mean()) / std # 将图像像素点进行 (0, 1) 标准化
12         img_tensor = transform(Image.fromarray(img_st)) # 要变成Image类型才
    能后续用transform转换
13         img_tensor4d = img_tensor.unsqueeze(0)
14         predict = float(torch.max(model(img_tensor4d), 1).indices) # 把标
    准化好的图片放进神经网络中进行预测
15         self.direct = self.directs[int(predict)]
16
17     else: # 如果是三维图像
18         # img = img[:, :, 0] # 方法一: 如果是png的三维图像, 取第一个slice图像作
    为目标
19         slice_num = img.shape[2] # 方法二: 投票法
20         list_img = list()
21         for slice_id in range(slice_num):
22             img_slice = img[:, :, slice_id]
23             std = img_slice.std()
24             if std == 0.0: # 这里是为了防止std为0
25                 std = 1
26             img_st = (img_slice - img_slice.mean()) / std # 将图像像素点进行
    (0, 1) 标准化
27             img_tensor = transform(Image.fromarray(img_st)).unsqueeze(0)
    # 要变成Image类型才能后续用transform转换
28             list_img.append(img_tensor)
29             img_tensor4d = torch.cat(list_img, dim=0)
30             predict = torch.max(model(img_tensor4d), 1).indices # 各个图片的分类
    预测
31             predict_result = int(max(set(predict), key=list(predict).count)) #
    根据投票结果选出最终方向
32             self.direct = self.directs[predict_result]
33         return self.direct

```