黄品溢个人学习笔记

Title 视频运动放大原理的学习

Tips Learning Notes From 2022.06.08

学生姓名 黄品溢

指导教师 王磊

学　　院 计算机学院

目　　录

第一章 关于学习过程中的疑问

1.1 关于训练和验证部分的代码

1.1.1 关于训练部分

在 LOSO 每一轮的训练中，其使用到的邻接矩阵似乎不同，为什么每一轮要使用不同的 adj 进行训练？其代码如下：

```
for idx in range(len(train_list)):
    npz_file = np.load(f"{args.npz_file}/{idx}.npz")
    adj_matrix = torch.FloatTensor(npz_file["adj_matrix"]).to(
        device)
    .............
    model = FMER(adj_matrix=adj_matrix,
                 num_classes=args.num_classes,
                 device=device).to(device)
```

在 LOSO 训练的每一轮 15 次 epochs 迭代中，其保存了效果最佳的参数，而在下一轮训练时，则又重头开始训练，为什么不使用上一轮的最佳参数继续训练？

1.1.2 关于图像的预处理部分

关于图像为了输入 MagNet 所进行的预处理，为什么需要进行这些步骤？可能因为我对 MagNet 的原理还没有了解很深，之后需要去了解具体细节吗？

```
def unit_preprocessing(unit):
    unit = cv2.resize(unit ,(256, 256))
    unit = cv2.cvtColor(unit, cv2.COLOR_BGR2RGB)  # BGR To RGB
    unit = np.transpose(unit / 127.5 - 1.0, (2, 0, 1))
    unit = torch.FloatTensor(unit).unsqueeze(0)
    return unit

def unit_postprocessing(unit):
    unit = unit[0]
    # 将每个通道的图像归一化
```

```
    max_v = torch.amax(unit, dim=(1, 2), keepdim=True)
    min_v = torch.amin(unit, dim=(1, 2), keepdim=True)
    unit = (unit - min_v) / (max_v - min_v)
    unit = torch.mean(unit, dim=0).numpy()
    unit = cv2.resize(unit, (128, 128))
    return unit

def magnify_postprecessing(unit):
    unit = unit[0].permute(1, 2, 0).contiguous()
    unit = (unit + 1.0) * 127.5
    # 转变形状为[128, 128]
    unit = unit.numpy().astype(np.uint8)
    unit = cv2.cvtColor(unit, cv2.COLOR_RGB2GRAY)
    unit = cv2.resize(unit, (128, 128))
    return unit

# 具体实现部分
onset_frame = unit_preprocessing(cv2.imread(onset_file))
apex_frame = unit_preprocessing(cv2.imread(apex_file))
shape_representation, magnify = self.magnet(batch_A=onset_frame,
                                            batch_B=apex_frame,
                                            batch_C=None,
                                            batch_M=None,
                                            amp=amp,
                                            mode="evaluate")
magnify = magnify_postprecessing(magnify)
shape_representation = unit_postprocessing(shape_representation)
```

第二章　关于 MIT 的欧拉影像放大算法 *Eulerian Video Magnification*

在这一节下，一共介绍有两种视角下的计算机影像放大方法。

2.1　拉格朗日视角（*Lagrangian Perspective*）

在拉格朗日视角下，从跟踪图像中感兴趣的像素（粒子）的运动轨迹的角度分析。

- 何为“变”——感兴趣的像素点随着时间的运动轨迹，这类像素点往往需要借助人工或其他先验知识来辅助确定;
- 放大“变”——将这些像素点的运动幅度加大。

在拉格朗日视角下，存在着以下几点不足：

- 需要对粒子的运动轨迹进行精确的跟踪和估计，需要耗费较多的计算资源;
- 对粒子的跟踪是独立进行的，缺乏对整体图像的考虑，容易出现图像没有闭合，从而影响放大后的效果;
- 对目标物体动作的放大就是修改粒子的运动轨迹，由于粒子的位置发生了变化，还需要对粒子原先的位置进行背景填充，同样会增加算法的复杂度。

2.2　欧拉视角（*Eulerian Perspective*）（**重点**）

欧拉视角将视角固定在一个地点，假定整幅图像都在变，只是这些变化信号的频率、振幅等特性不同，而我们所感兴趣的变化信号就身处其中。

打个比方，同样是研究河水的流速，我们也可以坐在岸边，观察河水经过一个固定的地方时的变化，这个变化可能包含很多和水流本身无关的成分，比如叶子掉下水面激起的涟漪，但我们只关注最能体现水流速的部分。

- 何为“变”——整个场景都在变，而我们所感兴趣的变化信号藏在其中;
- 放大“变”——通过信号处理手段，将感兴趣的信号分离，并进行增强。

欧拉影像放大技术的流程如下：

1. 空间滤波：将视频序列进行金字塔多分辨率分解;
2. 时域滤波：对每个尺度的图像进行时域带通滤波，得到感兴趣的若干频带（其中，对每一个像素点在时间轴上使用 FFT 快速傅里叶变换，将其转换至频域，之后使用特

定的滤波器进行滤波；带通滤波的含义是：允许特定频段的波通过，同时屏蔽其他频段）；

3. 放大滤波结果：对每个频带的信号用泰勒级数来差分逼近，线性放大逼近的结果；
4. 合成图像：合成经过放大后的图像。

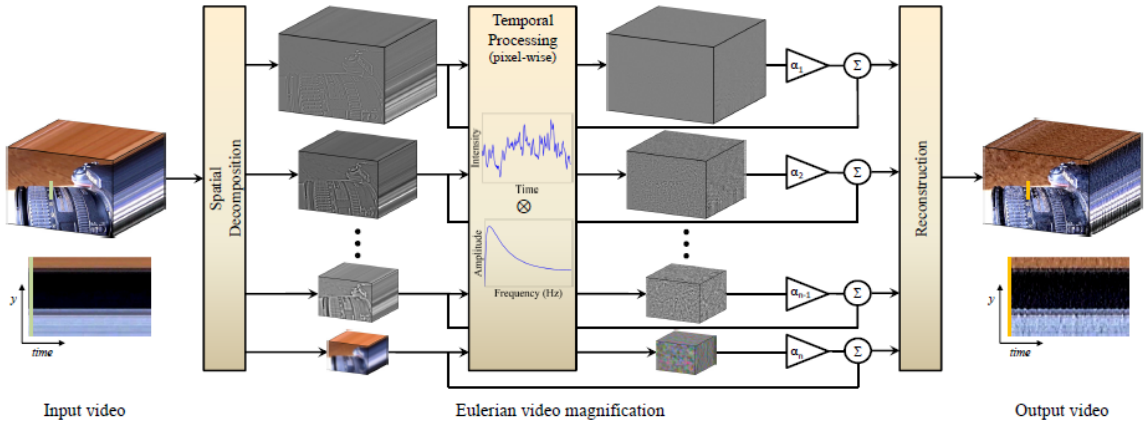

图 2.1　欧拉影像放大技术流程图

这些技术通常包括三个阶段：将帧分解成运动表示、操纵该表示，以及将操纵的表达重构为增强的帧。通常可以使用一阶泰勒展开式的空间分解，或使用复杂的空间金字塔提取基于相位的表示。

欧拉技术有助于揭示微妙的运动，但它们是手工设计的，没有考虑到许多问题，如遮挡和大运动等。正因为如此，它们容易产生噪声，并且经常遭受过度模糊。

但这种方法同样具有局限性：利用人工设计的过滤器来提取动作表示，也许不是最佳的方法。

2.3　基于神经网络的视频动作增强 MagNet

在这个方法之前，所有的视频放大方法都饱受噪声和过度模糊等问题的困扰，尤其是当放大倍数很大时。MagNet 也属于欧拉方法，但其分解是直接从例子中学习的，所以它具有更少的边缘伪影和更好的噪声特性。

最近的技术通过使用光流或像素移位卷积核明确地移位像素，这可以有高质量的结果。然而，当改变操纵因子的时候，这些技术通常需要重新训练。

对于帧内插，MagNet 的运动表示可以直接配置为不同的放大系数，不需要重新训练。并且 MagNet 不使用任何明确的像素移动，这将需要一个可微分的双线性采样模块，它可能不适合于子像素运动。

对于帧外推最近有一系列工作直接合成 RGB 像素值来预测未来的动态视频帧，但他

们的结果往往很模糊。我们的工作专注于放大视频中的运动，而不关心未来会发生什么。

并且，MagNet 在不使用时间滤波器的情况下，实现了与最新方法相似的效果。

MagNet 直接从样例中使用 CNN 学习分解滤波器；

MagNet 设计一个提取运动表示的网络，这样就可以通过简单地乘法，并重建一个放大的帧。

MagNet 神经网络主要由三个部分构成：

- 空间分解滤波器 Encoder：提取一个动作表示，类似于金字塔；
- 动作表示操作器 Manipulator：接收这个动作表示，并操作它来增强动作，（通过乘以差异）；
- 重构滤波器 Decoder：将修改后的表示法重建为生成的运动放大帧。

与欧拉视角的影像放大技术的各个部分的区别如下表所示：

Table 1 神经网络模型训练参数表

使用的方法	欧拉视角增强	MagNet
空间分解	拉普拉斯金字塔	深度卷积层
动作分离	时域带通滤波	减法或时域带通滤波
去噪表示	——	可训练卷积

Encoder 和 Decoder 全是卷积的，这使得他们能在任意分辨率下起作用；并且使用 Residual Blocks 残余模块生成高质量输出。为了节约内存空间和提升感受野，使用 Strided Convolution（即 stride > 1）的卷积进行下采样；通过使用一个卷积层跟着一个最近临域上采样，来避免棋盘状伪影。

经过测试，Encoder 中使用 3 个 3×3 残余模块和 Decoder 中使用 9 个残余模块产生较好的效果。

第三章 关于 FGRMER 模型的训练、验证部分代码

3.1 train.py

这部分代码用于实现模型的训练和验证。在训练部分采用了 LOSO（*Leave One Subject Out*，留一交叉验证）方法；而验证部分则是一次对所有留下的 Subject 样本进行验证。

在这一部分中，一共定义了三个函数，分别为：

- LOSO_train：用于实现留一交叉验证过程的函数；
- train：用于实现 LOSO 每一趟过程中的训练过程的函数；
- evaluate：用于实现每趟留一后，其余所有样本的识别准确率测试的函数。

接下来对每个函数的细节进行剖析学习。

3.1.1 LOSO_train

其大致的步骤如下：

1. 根据 LOSO 原则，生成一个 train_list 和 test_list：这两个变量都是列表的形式，但其列表的每个元素都是一个 pandas.DataFrame 变量，列表的长度即为 Subjects 的数量 27；
2. 根据每个元素的下标，选取相应的 adj_file 作为参数，构建 Model；
3. 使用交叉熵 CrossEntropyLoss 作为 Criterion，Adam 作为优化器，调用 train 函数进行训练；
4. 训练完成后，在测试集上进行测试。将每次测试的准确率和 F1-Score 累加，最后取平均，由此查看模型的整体效果。并将每轮 LOSO 的结果写入一个 log 日志文件中。

3.1.2 train

在这一部分进行模型的训练，使用 Criterion 求 loss，并根据反向传播，利用选择的 Adam 优化器进行优化。

对每一轮 LOSO，train 函数接收一个 epochs 参数，即训练轮数。在每一轮的训练中，若模型准确率大于之前的最大准确率，就将该参数进行保存。

在 LOSO 进行完一轮时，就实现了将最佳参数保存的效果。

3.1.3 evaluate

在这一节中，使用了 model.eval() 的验证模式，同时还使用了 torch.no_grad () 来防止模型参数的改变。

之后就是使用本轮 LOSO 中最佳的模型参数，来对测试集进行验证，分别求出了准确

率和 F1-Score，并返回值。

3.2 read_file.py

在这个文件中，实现了对 csv 文件的读取和返回操作。返回了一个 pd.DataFrame 变量，和一个 label_mapping，即情绪分类：序号的字典类型。

值得一提的是，在 label_mapping 变量生成的过程中，其是根据 data.loc[:, "Estimated Emotion"] 来获取数据的，即仅获取了这一列中的每一行。同时为了防止字典中出现重复的情绪，其还使用了 np.unique 函数，在去重的同时还进行了排序。

3.3 dataloader.py

在这个文件中，定义了两个函数：

- get_loader：实现了对自定义的 DataLoader 的返回。由于 DataLoader 的生成过程分为两步：第一步是写一个继承自 torch.utils.data.Dataset 类的自定义 Dataset 类；第二步是构造一个 Dataloader 对象。在这一个函数中，调用了其他文件中的自定义 Dataset 类，并生成了 DataLoader 对象，实现了返回。
- LOSO_sequence_generate：通过留一样本交叉验证 LOSO 方法，生成了一个 train_list 和一个 test_list，两个列表的长度均为 Subjects 的数量。
  这个函数写的是真妙！**反复学习!!**

3.4 dataset.py

在这个文件中，实现了一个自定义的 Dataset 类，用于构造 DataLoader，以供后续进行训练和验证。在这个部分，其对原始的图像进行了一系列的预处理，并在 get_ item 方法中实现了这一系列操作。

简单来说，其主要实现了以下几个步骤：

- 输入一个 item，通过 item 获取该表情的 label 标签；
- 读取一个 Onset Frame 图像和一个 Apex Frame 图像，首先进行预处理：转换图像的通道，并对每个通道进行归一化操作，使每个通道中的像素值映射在 [-1, 1] 之间。
- 随机选择一个放大倍数，并通过预训练好的 MagNet 模型进行输出，即得到一个形状表示 shape_representation 和放大后的图像 magnify。
- 将形状表示进行归一化操作，将像素值映射到 [0, 1] 之间，并将各个维度的像素值求平均，最后转变为 128×128 的图像。
- 放大后的图像 magnify 则将像素值从 [-1, 1] 区间映射回 [0, 255] 区间。并将其转化为灰度图像，将其形状变为 [128, 128]。

- 最后，在 magnify 图像中获取人脸的特征点的坐标，在 shape_representation 中获取 30 个 patches，并于标签一起返回。