人智大作业 2——表情识别

报告

班级： 自 93

学号： 2019010850

姓名： 王逸钦

完成日期： 2021/12/17

一　问题建模

本问题的实质是对 48*48 图像进行七分类，可以理解为 MNIST 十分类问题的强化。在 MNIST 分类问题中，我已经探索过使用 LeNet 进行分类，对于本次问题思路仍然是经典神经网络+softmax 函数的组合，将经典神经网络的最后一层通道数置为 7，经过 softmax 函数获得 7 个概率值，选取最大值以判断表情。

二　数据前处理

选择使用 pytorch 作为训练框架。对于给定的.csv 数据集，需要将其转为 torch.tensor 类型以输送给神经网络，为此使用 pandas 对数据进行前处理。读入.csv 后，首先分离出训练集和测试集，而后对每个数据(图片)的 Pixels 数据切分为逐个 pixel 值，转 int 后重构为 1*48*48 单通道图片尺寸。所有这样的图片组成列表，转为 tensor 类型即可。为便于后续直接调用，将该 tensor 保存为.pt 文件备用。

为便于神经网络使用，我还将编写了继承于 torch.utils.data.Dataset 类的定制化类 FacialExpressionDataset，重写__len__和__getitem__方法，以便被 DataLoader 调用制成数据迭代器。

本部分代码见 LeNet.ipynb 的 FacialExpressionDataset 类。

```
for i in range(self.L):
    xlst = data.loc[i].pixels.split()
    xlst = [int(x) for x in xlst]
    X = np.array(xlst).reshape([1,48,48])
    y = data.loc[i].emotion
    self.Xlst.append(X)
    self.ylst.append(y)
self.Xlst = torch.tensor(self.Xlst,dtype=torch.float32)
self.ylst = torch.tensor(self.ylst,dtype=torch.int64)
```

逐图片进行前处理

三　神经网络实验（问题 1）

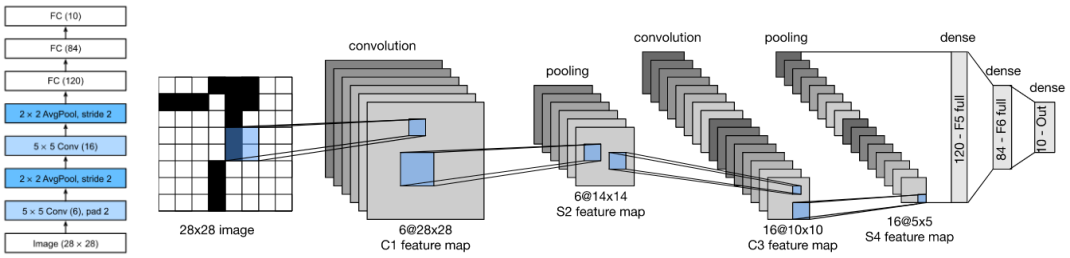
本节实验，尝试 LeNet，VGG-9（由 VGG11 简化而得），ResNet-11 三种网络，优化器均使用 SGD

1)　训练框架搭建

复用 coding4 中已经搭建好的 LeNet 网络中的 train 函数：接受网络、数据迭代器(已含批量大小)、迭代次数与学习率作为参数，建立优化器，确定损失函数，而后在 for 循环中逐 epoch 从数据迭代器中读取数据、优化器梯度置 0、数据送入网络计算损失并反向传播、优化器步进(根据梯度优化网络参数)。

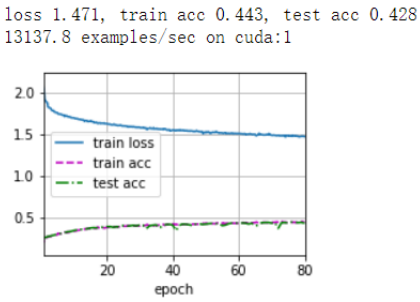
本部分代码我基于 d2l 包或 pytorch 文档进行了两种实现，分别见 LeNet.ipynb 的 train 函数，以及 vgg11aug_balance_pretrain 的 train_model 函数。

2)　LeNet【代码见 LeNet.ipynb】



LeNet

LeNet 结构由两个卷积层、两个池化层和三个全连接层组成，其设计之初就用于解决 MNIST 的十分类问题，结构简单，其基本使用在 coding4 中已有探索。实际对本次数据进行训练时 LeNet 效果一般。以 lr=0.001， num_epochs=80，batch_size=64 训练获得约 43%测试集精度，过程如下所示：

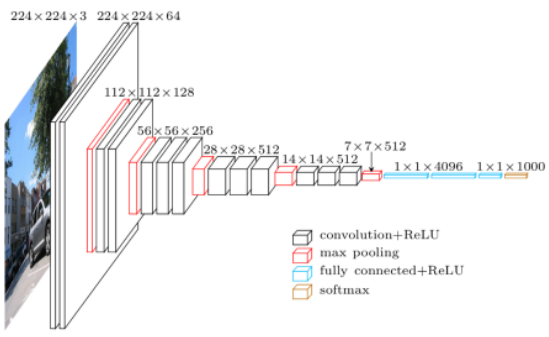


LeNet

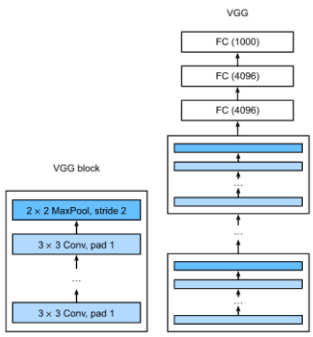
若进一步增大学习率或增大 num_epochs，将会导致各 epoch 的 test acc 抖动剧烈，甚至出现 acc 不足 30%的 epoch，稳定性很差。这可能是由于 LeNet 模型简单，未使用 dropout 等结构来减小过拟合，同时优化器中也未使用 weight_decay

3) VGG-9【代码见 vgg9.ipynb】

VGG 模型给出了一种 CNN 的构造方法论，创造了 VGG 块的概念，根据需求选择不同数量和卷积层的 VGG 块进行串联即可快速搭建一个 VGG 神经网络。给出 VGG 块及经典网络 VGG-11 的结构如下：



VGG-11



VGG Block

VGG-11 模型的设计是用于处理 224*224 图像的，对于本作业的 48*48 数据仍然过深，对其进一步简化如下：取消最后一个 2 层 512 通道的 VGG 块，首个全连接层输出通道降低至 2048。更改后的模型由 6 个卷积层和 3 个全连接层构成，不妨称其为“VGG-9”。

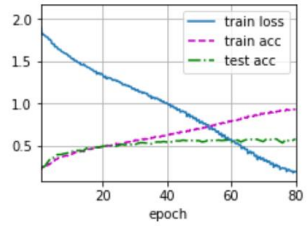
```
conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512))

def vgg(conv_arch):
    conv_blks = []
    in_channels = 1
    # 卷积层部分
    for (num_convs, out_channels) in conv_arch:
        conv_blks.append(vgg_block(num_convs, in_channels, out_channels))
        in_channels = out_channels

    return nn.Sequential(
        *conv_blks, nn.Flatten(),
        # 全连接层部分
        nn.Linear(out_channels * 3 * 3, 2048), nn.ReLU(), nn.Dropout(0.5),
        nn.Linear(2048, 2048), nn.ReLU(), nn.Dropout(0.5),
        nn.Linear(2048, 7))
```

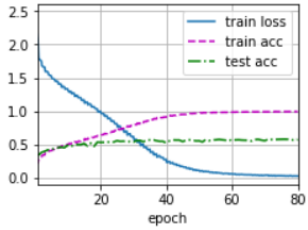
VGG-9

```
loss 0.198, train acc 0.932, test acc 0.584
3684.7 examples/sec on cuda:2
```



VGG-9

```
loss 0.028, train acc 0.994, test acc 0.574
3195.9 examples/sec on cuda:1
```



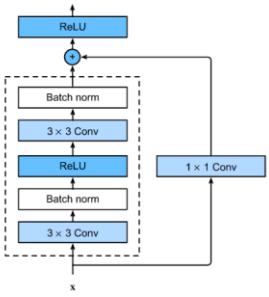
VGG-9 (BN)

进行两次训练，分别使用无批量正则化层和有批量正则化层(BN)两种模型，测试集精度均收敛到 58%附近，显著好于 LeNet。可以发现，BN 层有效加速了网络的训练，但最终收敛时的测试集精度无提升。由此推断对于训练时间较长的大型网络，可以考虑使用 BN 层以加速收敛。对于本次作业其意义不明显。此外我还尝试在 VGG 块中加入 dropout 层，但会导致模型不收敛，结果在此略过。

了解到本次数据集似乎为“Fer2013”一致，而该数据集存在数据误标注甚至部分数据无表情的情况，数据质量不佳。在未作任何平衡、增强的原始数据集上获得 58%的测试集精度是可接受的。

4) ResNet-11【代码见 ResNet-11.ipynb】

为进一步探索何种模型最适合本次训练任务，我又使用了 ResNet-11 进行训练。ResNet 是现代神经网络中十分流行的网络，至今仍应用广泛，其模型参数较 VGG 而言更少，却往往达到接近或更佳的效果。ResNet 的核心结构是残差块，它可以将之前层的参数不经过下一层卷积网络直接传给 ReLU，实现参数的重复利用。重复上述残差块，即构成 ResNet 网络。

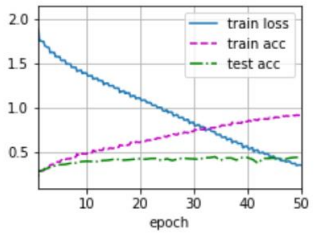


ResBlock

```
# ResNet-11
b1 = nn.Sequential(nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3),
                   nn.BatchNorm2d(64), nn.ReLU(),
                   nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
#首个stage通道不翻倍
b2 = nn.Sequential(*resnet_block(64, 64, 2, first_block=True))
b3 = nn.Sequential(*resnet_block(64, 128, 2))
b4 = nn.Sequential(*resnet_block(128, 256, 2))
b5 = nn.Sequential(*resnet_block(256, 512, 2))
net = nn.Sequential(b1, b2, b3, b4, b5, nn.AdaptiveAvgPool2d((1,1)),
                    nn.Flatten(), nn.Linear(512, 7))
```

ResNet-11

```
loss 0.355, train acc 0.919, test acc 0.442
1664.6 examples/sec on cuda:1
```



ResNet-11

然而，直接使用 ResNet-11 得到的结果仅和 LeNet 相当，训练过程从第 5 个 epoch 开始发生过拟合并逐渐扩大，测试集精度上升乏力。尚不清楚这一现象的原因。

5) 评估

模型	LeNet	VGG-9	ResNet-11
Test Acc	42.8%	58.4%	44.2%

VGG-9 在测试集精度上表现最佳，在问题 2 中继续使用 VGG-9 模型。

四 数据增强（问题 2）【代码见 vgg9-DataEnhance.ipynb】

使用 Pandas 对数据进行简单分析不难发现，标号为 1 的数据(Disgust)尤其少，约为其他类型图片平均数量的 10%:

```
In [6]: for i in range(7):
            print( '%d:%d'%(i,len(data[data.emotion==i])) )
```

```
0:4953
1:547
2:5121
3:8989
4:6077
5:4002
6:6198
```

数据不平衡

为解决这一问题，应使用数据增强的方法，将此类的每个数据扩充为 10 个。设计一个随机水平反转、随机剪裁、随机调节对比度与亮度三者组合的 transform 方法如下：(使用 torchvision 包)

```
def mytransform():
    transA = transforms.RandomHorizontalFlip()
    transB = transforms.RandomResizedCrop((48, 48), scale=(0.8, 1), ratio=(0.8, 1.25))
    transC = transforms.ColorJitter(brightness=0.7, contrast=0.7)
    return transforms.Compose([transA, transB, transC]) #三种变换的组合
```

mytransform 方法

然后对每个标号为 1 的训练数据集使用该 transform 方法。具体而言，为保证随机剪裁后图片仍可被 resize 为 48*48，我首先用 Haming 插值(减小锐度损失)将其插值到 64*64 后再应用 transform 方法：

```
if transform and y==1 and train:
    x = transform(xlst.resize(
        (64,64),resample=Image.HAMMING))
else:
    x = xlst
```

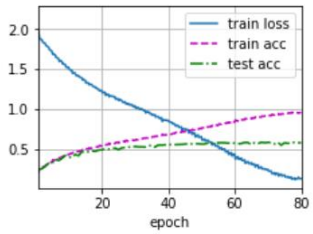
逐图使用 transform 方法

transform 效果

需要注意，torchvision.transforms 函数需要接收 PIL Image 类型的数据，因此需要首先将 tensor 转为 PIL Image，而此步转换有要求 tensor 数据类型为 unit8，因此这一过程设计诸多类型转换，详见代码。

使用增强后的数据对 VGG-9 再次训练，测试集精度由 58.4%提升至 58.8%，无明显变化。

```
loss 0.135, train acc 0.956, best test acc 0.588
3660.6 examples/sec on cuda:1
```



VGG-9(数据平衡&增强)

五　模型改进与预训练（自主选做）

代码见 vgg11aug_balance_pretrain.ipynb, ResNet18aug_balance_pretrain.ipynb, ResNet18aug_balance_pretrain2.ipynb

为进一步挖掘数据集的潜力，我尝试使用 ImageNet 预训练模型微调的方法。

torchvision.dataset 自带的预训练模型中并没有小于 VGG-11 的 VGG 模型和小于 ResNet-18 的 ResNet 模型，而上述两者都需要输入 224*224 图像。为适应模型需要，我对上述“数据增强”代码进行细微调整，将所有图像都插值到 224*224 进行

实验。

微调，即使用已经预训练好的模型参数，增加最后一层全连接层，对 n 分类问题这最后一层的输出参数即为 n。容易理解，最后一层的模型参数未经预训练，其学习率理应比之前层更大，以期更快收敛。

```
#输出层拥有十倍学习率
def train_fine_tuning_vgg(net, train_iter, test_iter, learning_rate=5e-5, num_epochs=10, device=torch.device('cpu')):
    loss = nn.CrossEntropyLoss()
    params_1x = [param for name, param in net.named_parameters() if name not in ["classifier.6.weight", "classifier.6.bias"]]
    trainer = torch.optim.SGD([{'params': params_1x},{'params': net.classifier[6].parameters(),'lr': learning_rate*10}],
                              lr=learning_rate, weight_decay=0.001)
    dataloaders_dict = {'train':train_iter, 'val':test_iter}
    finetune_net, hist = train_model(net, dataloaders_dict, loss, trainer, num_epochs, device)
    return finetune_net, hist
```

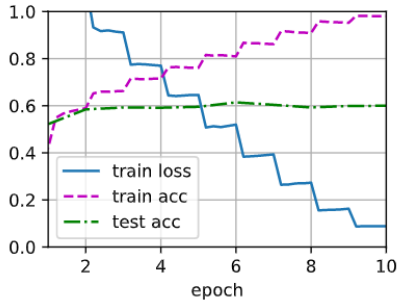
微调 输出层学习率更高

对 VGG-11 预训练模型和 ResNet-18 预训练模型都进行微调训练，结果如下：

```
train Loss: 0.5540 Acc: 0.7957
val Loss: 1.2227 Acc: 0.6206
```

VGG-11 预训练微调

```
loss 0.089, train acc 0.980, test acc 0.600
578.1 examples/sec on [device(type='cuda', index=1)]
```



ResNet-18 预训练微调（第一种实现）

```
train Loss: 0.5562 Acc: 0.8070
val Loss: 1.2392 Acc: 0.5911

Training complete in 22m 22s
Best val Acc: 0.591109
```

ResNet-18 预训练微调（第二种实现）

可以看到，两个预训练模型均突破了常规训练 VGG-9 的 58.8%测试集精度，其中 VGG-11 预训练模型达到了 62.1%。这也是本次作业中我所训练出的精度最高的模型。

模型	VGG-9 数据增强	VGG-11 预训练	ResNet-18 预训练
Test Acc	58.8%	62.1%	59.1%
模型大小	70MB	500MB	45MB
测试速度	30fps	4fps	10fps

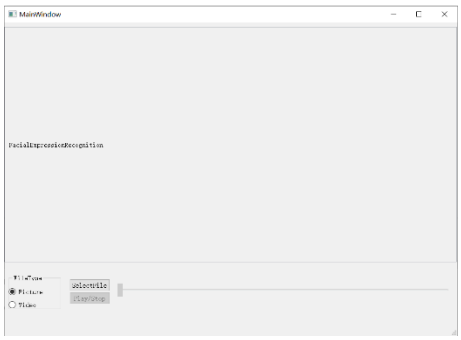
由上表可知，虽然 VGG-11 精度最高，但其参数很多模型较大，单次测试效率耗时也较高(见 TestTime.ipynb，测于笔记本 i5-7300HQ)，且精度较 VGG-9 提升并不显著。因此问题 3 中仍使用 VGG-9 作为测试模型。

注：给出 ResNet-18 预训练的两种实现代码，附件链接中模型来自第二种实现。

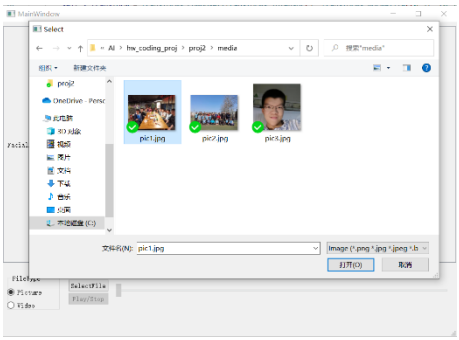
六　人脸识别、GUI 与图像视频接口（问题 3）

1)　GUI 使用方法与效果展示

运行 myui.py，出现如下方左图所示主界面，在图片和视频二选一按钮中选择想要打开的文件类型，而后点击 SelectFile 按钮，在下方右图所示的文件窗口中选择文件：



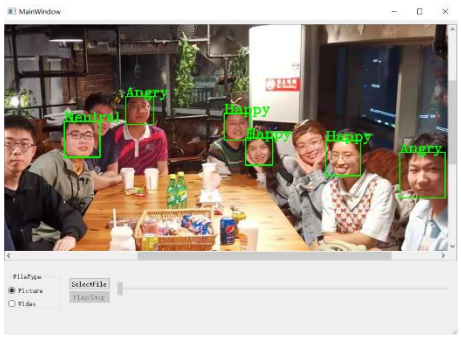
主界面



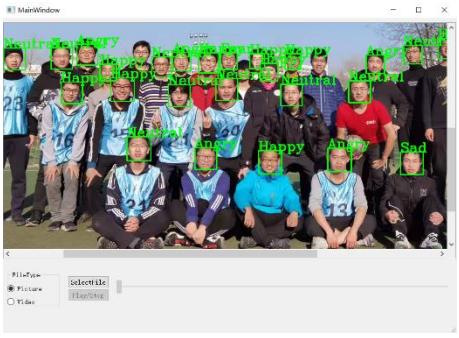
选择文件

若选择图片文件，则表情识别后直接显示在窗体中，无需其他操作；若选择视频文件，则应点击“Play/Stop”按钮以开始播放或停止播放视频，还可以拖动底部进度条调节视频播放位置。

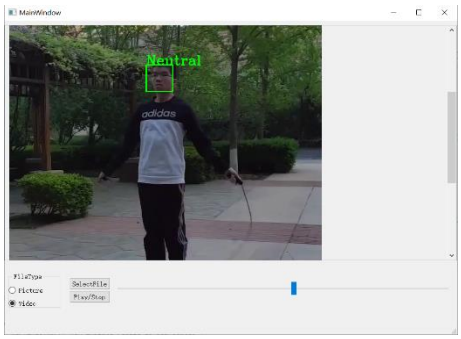
打开不同文件的识别效果展示：



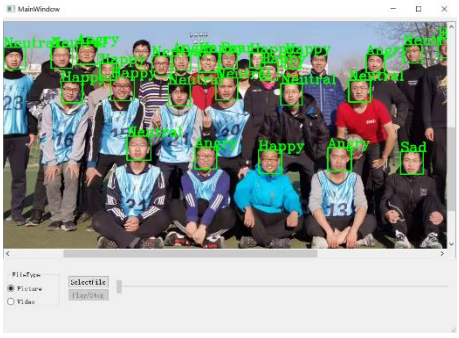
图片 1



图片 2



视频 1



视频 2

2)　myui.py 编写思路

首先在 qt designer 中绘制图形界面，使用 pyuic5 命令转为.py，并在自动生成的含有图形界面的.py 文件上进一步嵌入算法。算法主要分两步：识别人脸、判断表情。识别人脸使用 Open-CV 自带函数，判断表情使用前文所述 vgg9net.pkl 网络。

本部分代码的难点在于 1.如何对视频逐帧识别，加上识别框之后再播放；2.如何处理 CV2 图像、PIL Image、不同类型的 torch.Tensor 之间的类型转换。对于前者，我使用了 PyQt 定时器，以原始视频的帧率即时，不断显示出标记好的图片，即达到“播放”的效果；对于后者，我查阅相关文档，对各类不符合预期的情况使用单步调试解决。

七　亮点、困难与解决方案、收获

1) 亮点

- 在问题 1 中使用多种网络进行实验，对比其效果，并探索了 BN 层和 Dropout 层加入对实验效果的影响。
- 在问题 2 中对数据实现了增强
- 在问题 3 中设计了图形界面和视频接口，结合 CV2 人脸识别算法，实现一个展示性较强的成品
- 补充进行了两种使用预训练模型和微调的神经网络实验，对比其识别精度并分析其运行效率。
- 设计了一种双层循环算法，用于将.csv 数据处理为 Tensor 类型数据，保存为.pt 文件；这一方法比单层循环的处理方法高效得多。详见 vgg9-DataEnhance.ipynb 的 FacialExpressionDataset 类。

2) 困难与解决方案

- 开始我使用 VGG-9 达到略超过 50%的精度，认为精度过低，判断模型选择或训练方法有误；后来在做其他课程实验时与同学交流模型选择心得，才发现大家的精度都并不太高，受到数据质量的制约。
- 数据类型问题是本次 Bug 的最大来源。仅图片文件就涉及到：python list、numpy array、pandas dataframe、torch tensor(uint8)、torch tensor(float32)、PIL Image、QImage 如此之多的数据类型。不论是数据前处理、数据增强，还是最终在图形界面进行表情预测，都涉及频繁的数据类型转换。特别是两种 tensor 类型如果忘记转换，并不会引起报错，但会产生不符合预期的结果，这也是我碰到的一个 Bug。
- 微调预训练模型需要对末层使用更大的学习率，从而需要对 train 函数进行较多修改，这一步也消耗了较多精力。

3) 收获

- 掌握使用 pytorch，利用经典神经网络进行深度学习的基本方法
- 掌握使用 pandas 进行数据预处理，使用 torchvision 进行图像增强的基本方法
- 虽然本次作业中我所使用的诸多方法（加入 BN 层、数据增强、采用预训练模型微调）都是实际优化神经网络的常用手段，但受制于本次数据质量较差，数据信息量较低，上述调优方法几乎没有表现出应有的效果。希望助教们在以后课程中可以给出更具优化潜能的数据集。
- 在 20 岁生日当天完成本次作业，将会是属于我的独特回忆。

附：训练完毕模型及增强后数据下载地址(保留至 2022-1-30)

https://cloud.tsinghua.edu.cn/d/0ad9f295e50849b1af98/

附：参考资料

李沐-《动手学机器学习》课程示例代码：
https://zh-v2.d2l.ai/d2l-zh.zip
Sasank Chilamkurthy-定制 Dataset 并制作 Dataloader：
https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
Nathan Inkawhich-对预训练模型进行微调：
https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
菜鸟教程-Pandas DataFrame
https://www.runoob.com/pandas/pandas-dataframe.html