HW1 实验报告：基于 CNN 的面部表情识别

211240074　余今　匡亚明学院脑科学与人工智能

1　模型构建与优化

1.1　　初步尝试

在数据预处理阶段，我只是简单地提取了 train.csv 各图的灰度矩阵，并将其转换为 Tensor 格式。

随后，构建了一个比较简单的 CNN 网络结构，包括三个 block 以及最后的展平层和线性层。其中，一个 block 包括一个 Conv2d 层、一个 BatchNorm2d 层和一个 ReLU 层，还有一个可在实际使用中调节的池化层。

```
def block(input_channels, output_channels, pool=False):
    layer = [
        nn.Conv2d(input_channels, output_channels, kernel_size=5),
        nn.BatchNorm2d(output_channels),
        nn.ReLU(inplace=True),
    ]
    if pool:
        layer.append(nn.MaxPool2d(2))
    return nn.Sequential(*layer)
```

简单的 CNN 网络结构在 tensorboard 中展示如下图所示。（具体代码以及参数见 MyModule.py 的 block 类和 MyEmotionCNN 类）

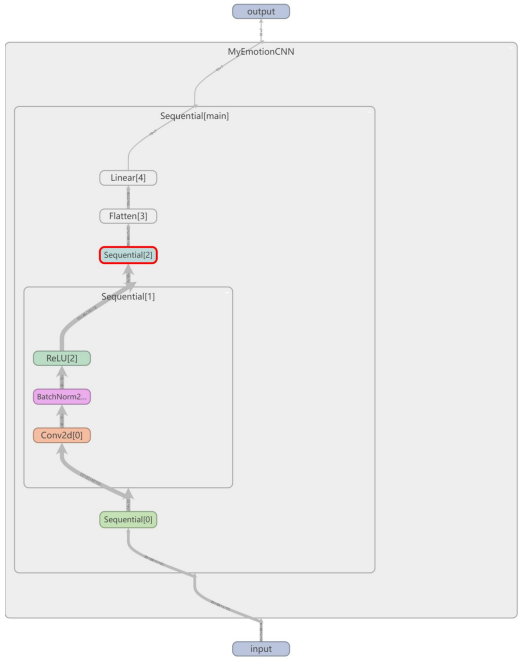

图 1 简单的 CNN 结构（Sequential 为 block，其中 Sequential[1]具体结构已展开）

在训练时，我选择了 Adam 作为优化器，交叉熵作为损失函数，训练了 50 轮并在每轮结束后输出训练集准确率、验证集准确率（train.csv 中 20%作为验证）和交叉熵损失，训练集和测试集的正确率仅在 40%上下波动，且损失函数无法收敛，模型欠拟合。

1.2 网络模型结构优化过程

首先，我尝试引入残差层，设计了 res_block。res_block 基本构造与 block 相同，但对 kernel_size 与 padding 有所调节，并且没有池化层，保证前后输出尺寸相同并可用于残差计算。res_block 代码如下图所示。

```
def res_block(input_channels, output_channels):
    layer = [
        nn.Conv2d(input_channels, output_channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(output_channels),
        nn.ReLU(inplace=True),
    ]
    return nn.Sequential(*layer)
```

然后，为了提高准确率，我加深了网络结构，在最终的展平层与线性层前堆叠了多个 block 和 res_block。

```
self.conv1 = block(1, 24, pool=True)    # 1*48*48 -> 24*22*22
self.res1 = nn.Sequential(res_block(24, 24), res_block(24, 24))    # 24*22*22
self.conv2 = block(24, 96, pool=True)    # 24*22*22 -> 96*9*9
self.res2 = nn.Sequential(res_block(96, 96), res_block(96, 96))    # 96*9*9
```

重新训练 50 轮，发现训练集准确率在后期提高至接近 80%，而验证集准确率在 55%处已收敛，这说明模型过拟合了。

在查阅资料后，为了抑制过拟合，我添加了 Dropout 层，考虑到靠近输入端的特征采样不应丢失过多信息，而靠近输出端时则需要抑制过拟合的倾向，我微调了 Dropout 的参数。最终得到了一个较为理想的模型（见 MyModule.py 中的 MyResNet 类），模型的具体结构见可视化部分。

1.3 其他优化操作

在数据预处理阶段，为了增强数据，我添加了随机旋转和随机裁剪（见 Module.py 的 transform 变量）。在训练模型是，我尝试调节了一些超参，如梯度下降的计算规模 batch_size、学习率 learning_rate 和权重衰减 wt_decay，并添加了 StepLR 学习率调度，在多次尝试中，得到了较优的设置 batch_size=256, epochs=100, learning_rate=1e-3, wt_decay=1e-5。

除此以外，我也尝试过换优化器和直方图均衡化灰度信息等操作，但效果都不是很理想，最终没有采用。

1.4 模型训练结果

在训练了 100 轮后，模型在测试集和验证集上的准确率和 loss 变化如图所示，可以看到，最终在训练集上的准确率稳定在 70%左右，在验证集上的准确率稳定在 60%左右，loss 也将近收敛。

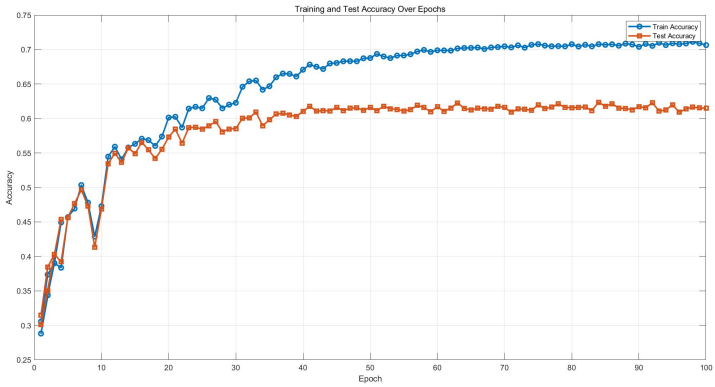

图 2 模型在训练集和验证集上的准确率变化

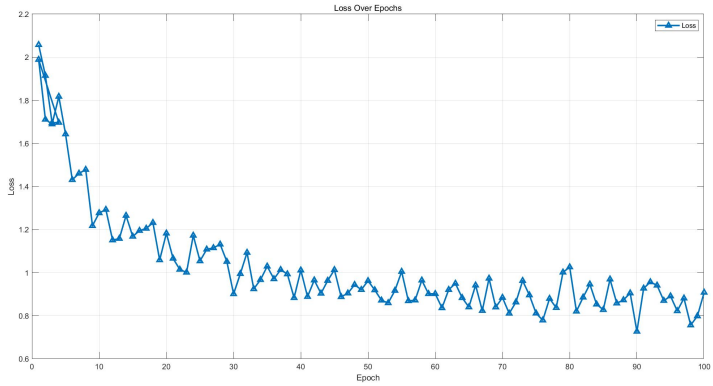

图 3 模型训练时的 loss 变化

选取准确率变化平稳后的几个 epoch 训练出的 model 来对 test.csv 文件进行预测后上传 kaggle 平台，可以在提交记录中看到预测准确率与验证集上的表现接近。

test_predictions88.csv Complete · 18h ago	0.61103	☐
test_predictions85.csv Complete · 18h ago	0.61465	☐
test_predictions82.csv Complete · 18h ago	0.60936	☐
test_predictions71.csv Complete · 19h ago	0.61493	☐
test_predictions51.csv Complete · 19h ago	0.61326	☐
test_predictions.csv Complete · 21h ago	0.61131	☐

图 4 Kaggle 评测记录

2 Saliency Maps 的绘制

为了绘制出 Saliency Map，需要计算逆梯度以计算像素权重，具体代码见 Saliency_Map.py。绘制了前 16 张图片及其 Saliency map。通过观察可以发现，模型在做 classification 时，大多 focus 在图片的五官部分，主要是眼睛、嘴唇等，在面部肌肉起伏明显时，focus 也会在额头、两颊（靠眼周与嘴周）等地方。

图 5 用于示例的 16 张图片及其 Saliency Map

3 观察特定层的 filter

3.1 训练图在每个layer层的滤波图像

使用 Gradient Acsent 方法，输出了部分图像在每个 Conv2d 层的 filter 图像，以下为其中两张图的 filter，可以看到眼睛与嘴巴部分的信号会得到放大。（代码见 Feature_Map.py）

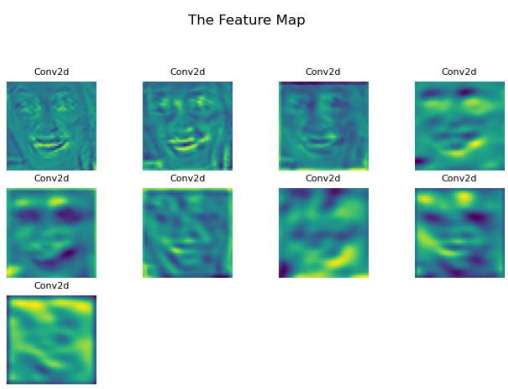

图 6 Filter1

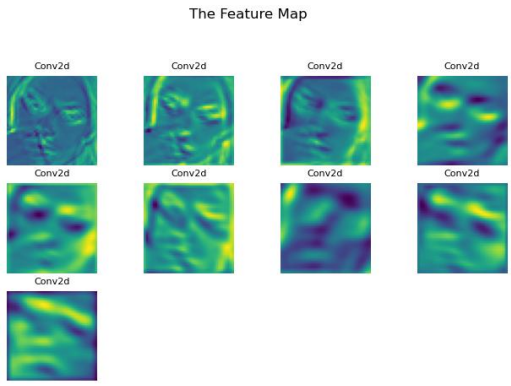

图 7 Filter2

3.2 不同类型图片在self.conv1上的表现

选取 self.conv1 层使用梯度上升方法观察特定层的哪些图像最容易激活，计算像素权重后，得到的部分图像如下（emotion1-7）。从定性的角度出发，发现 emotion 为 0/3/6 的类型较容易激活该层。

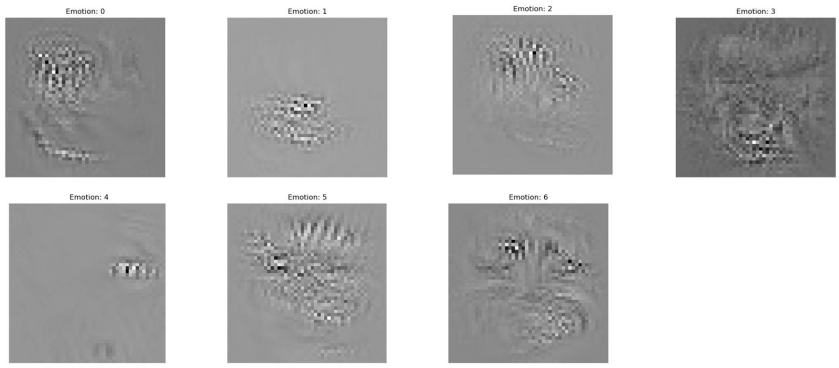

图 8 在 self.conv1 上的像素权重

4 对每个 label 的准确率分析

在 train.csv 中，模型对每个 label 单独划分的准确率如图所示。可以看到，我的模型对于 2 -恐惧的分类准确率最低，对 3-快乐的分类准确率最高

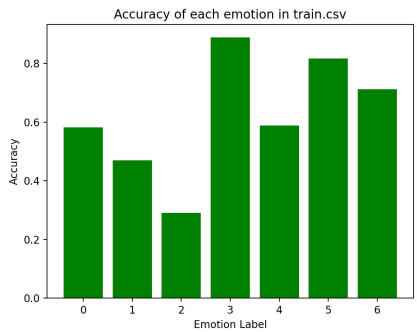

图 9 模型对每个 label 的准确率

由于时间和能力限制，暂时还未用 Lime 对具体的原因进行分析，希望在后面学到更多技能以后能回头来分析我的模型在某些 label 表现得特别好。

5 观察模型的训练

5.1 模型结构可视化

在可视化时，我采用了 tensorboard 和 neuron 的方式。tensorboard 的效果见图 1，美观性不是很好。Neuron 则能很好地展示我最终的模型结构。（由于模型过深，我截断了结构图，左中右分别为模型的前段、中段和后段）

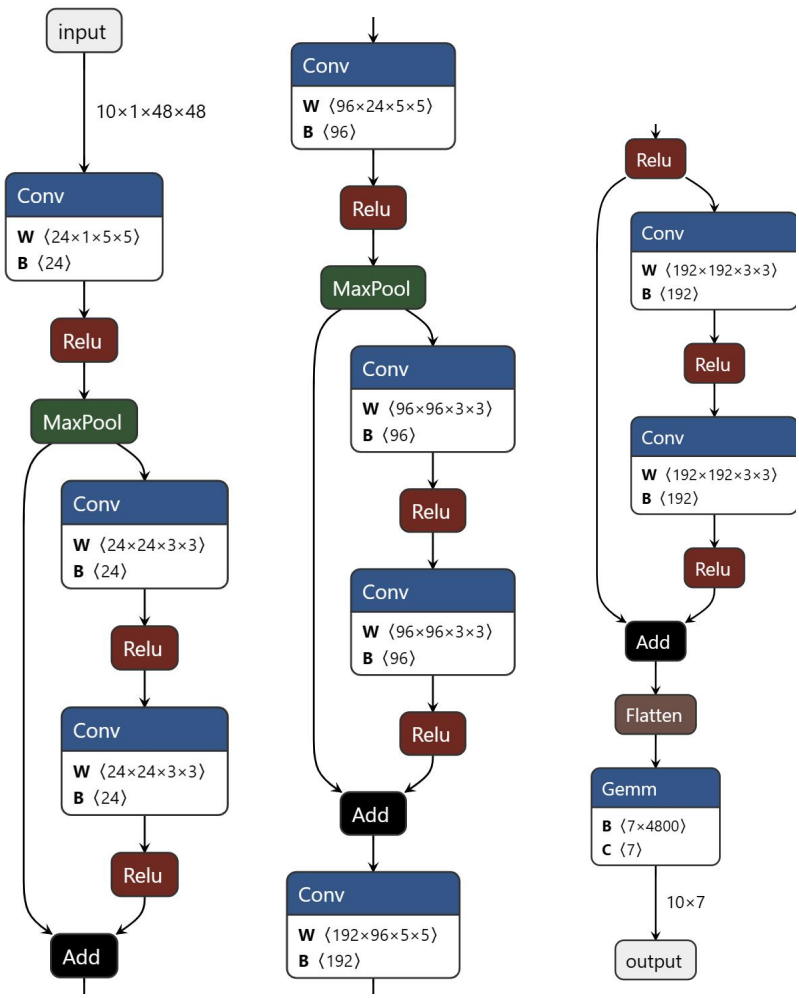

图 9 neuron 可视化

5.2 模型训练过程可视化

在每一个 epoch 中，我都记录了模型在训练集和验证集上的准确率以及损失函数的值的变化（见图 2 与图 3），可以从数据上判断模型的强化与收敛过程。