神经网络报告 #HW2

吴雨欣 191240060 匡亚明学院

1 代码框架

本实验的代码主要分成三个部分：数据获取及预处理，网络构建，训练及输出。

1.1 数据获取及预处理

代码见 Step1

使用 pandas 库读取 train.csv 文件的数据，并将其转化为一个四维 tensor：第一维对应每组图像及其 lebal，图像是一位三维 tensor，每一维代表 channel、length 和 width，lable 是 $0-6$ 的情感标签。

如图为 processed_date[0]

```
[tensor([[[ 70.,  80.,  82.,  ...,  52.,  43.,  41.],
          [ 65.,  61.,  58.,  ...,  56.,  52.,  44.],
          [ 50.,  43.,  54.,  ...,  49.,  56.,  47.],
          ...,
          [ 91.,  65.,  42.,  ...,  72.,  56.,  43.],
          [ 77.,  82.,  79.,  ..., 105.,  70.,  46.],
          [ 77.,  72.,  84.,  ..., 106., 109.,  82.]]]),
 0]
```

图 1: Processed_date[0]

网络训练的数据集分为训练集和验证集，将 train.csv 中的数据随机打乱之后，取其中 80% 训练集，其余 20% 作为验证集。

尝试使用 torchvision.transforms 对数据进行翻转、随机旋转处理，但效果并不明显。

```
32      #随机旋转
33      rot = transforms.Compose([
34          transforms.ToPILImage(),
35          transforms.RandomRotation(10),
36          transforms.ToTensor()
37      ])
38      #水平翻转
39      flip = transforms.Compose([transforms.RandomHorizontalFlip(p=1)])
40      flipped = flip(origin)
```

图 2: 对图片进行翻转和旋转

1.2 网络构建

代码见 Step2

尝试使用了 AlexNet、VGG 等网络并对其结果进行比对之后，基于 VGG 构建了一个由 5 个 Stage 组成的网络，每个 Stage 包括 3 个 Conv2d，1 个 MaxPool2d 及 Dropout 组成。

```python
# AlexNet
class AlexNet(nn.Module):
    def __init__(self, num_calsses=7, init_waight=True):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2)
        )
```

图 3: AlexNetTry

```python
# VGG
class VGG(nn.Module):
    def __init__(self, arch: object, num_classes=7) -> object:
        super(VGG, self).__init__()
        self.in_channels = 1
        self.conv3_64 = self.__make_layer(64, arch[0])
        self.conv3_128 = self.__make_layer(128, arch[1])
        self.conv3_256 = self.__make_layer(256, arch[2])
        self.conv3_512a = self.__make_layer(512, arch[3])
        self.conv3_512b = self.__make_layer(512, arch[4])
        self.fc1 = nn.Linear(512, 4096)
        self.bn1 = nn.BatchNorm1d(4096)
        self.bn2 = nn.BatchNorm1d(4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, num_classes)
    def __make_layer(self, channels, num):
        layers = []
        for i in range(num):
            layers.append(nn.Conv2d(self.in_channels, channels, 3, stride=1, padding=1, bias=False))  # same padding
            layers.append(nn.BatchNorm2d(channels))
            layers.append(nn.ReLU())
            self.in_channels = channels
        return nn.Sequential(*layers)
```

图 4: VGGTry

```python
        self.stage1 = nn.Sequential(nn.Conv2d(1, 64, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(64),
                                    nn.Conv2d(64, 64, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(64),
                                    nn.Conv2d(64, 64, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(64),
                                    nn.MaxPool2d(kernel_size=2), nn.Dropout(p=0.5))
```

图 5: MyNetwork stage1 示例

```python
    def __init__(self):
        super(Net, self).__init__()
        self.stage1 = nn.Sequential(nn.Conv2d(1, 64, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(64),nn.Conv2d(
        self.stage2 = nn.Sequential(nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(128),nn.Conv
        self.stage3 = nn.Sequential(nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(128),nn.Con
        self.stage4 = nn.Sequential(nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(128),nn.Con
        self.stage5 = nn.Sequential(nn.Conv2d(128, 256, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(256),nn.Con
        self.stages = [self.stage1, self.stage2, self.stage3, self.stage4, self.stage5]

        self.fc = nn.Sequential(nn.Flatten(), nn.Linear(256, 1024), nn.ReLU(), nn.Dropout(p=0.5),
                                nn.Linear(1024, 1024), nn.ReLU(), nn.Dropout(p=0.5), nn.Linear(1024, 7))
```

图 6: MyNetwork

1.3 训练及输出

代码见 Step3

训练参数：train_lr=0.002,train_epochs=50,batch_size=512。训练结束后选择训练过程中验证集正确率最高的模型 (输出为 Net.param)，使用该模型对 test.csv 中的数据进行预测。

在每一轮的训练中，记录 loss,train_acc 和 valid_acc 并绘制散点图，以此判断训练出现拐点的时刻并及时停止以避免不必要的开销。

2 Saliency Map

Saliency map：显著性是一种图像分区的模式，而显著图是显示每个像素独特性的图像。显著图的目标在于将一般图像的表示简化或是改变为更容易分析的样式。举例来说，当人眼看到一张图像时，会被其中某些区域吸引住，然后再看其他次吸引的区域，这些区域就是根据显著性来进行分析。

原理：计算与图像像素对应的正确分类中的标准化分数的梯度。以本实验的数据为例，图像的 tensor 是 (1, 48, 48)，这个梯度的形状也是 (1, 48, 48)；对于图像中的每个像素点，梯度显示了当像素点发生轻微改变时，该改动对**最后一层**输出结果的影响，影响越大代表此像素对模型越重要。

实现：首先进行前向操作，将输入图像传入已经训练好的 Net model，然后进行反向传播，从而得到对应图像。

```
386  def compute_saliency_map(series, model):
387      fig = torch.unsqueeze(processed_data[series][0], dim=0)
388      model.eval()
389      X_var = torch.autograd.Variable(fig, requires_grad=True)
390      #前向操作
391      scores = model(X_var.to(try_gpu()))[0]
392      #反向传播
393      scores.backward(torch.FloatTensor([1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]).to(try_gpu()))
394      #输入图像的像素梯度
395      saliency_map = X_var.grad.data
396      saliency_map = saliency_map.abs()
397      saliency_map, i = torch.max(saliency_map, dim=1)
398      saliency_map = saliency_map.squeeze()
399
400      return saliency_map
```

图 7: Compute Saliency Map

如下为 label 为 Disgust，Fear，Happy 及 Sad 被正确分类图像的 saliency map

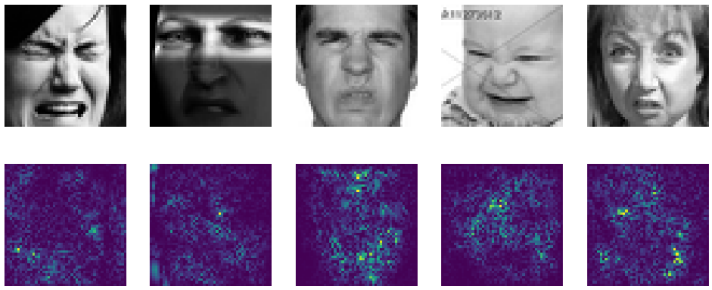


图 8: Saliency Map - Disgust

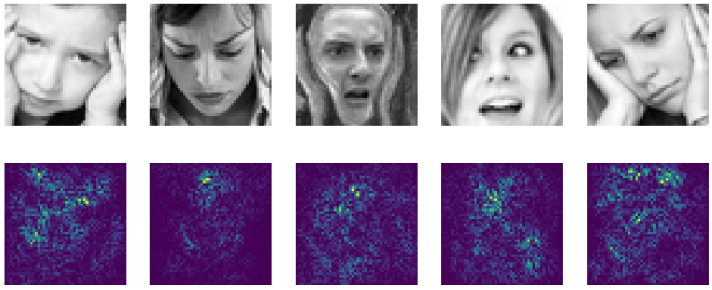


图 9: Saliency Map - Fear

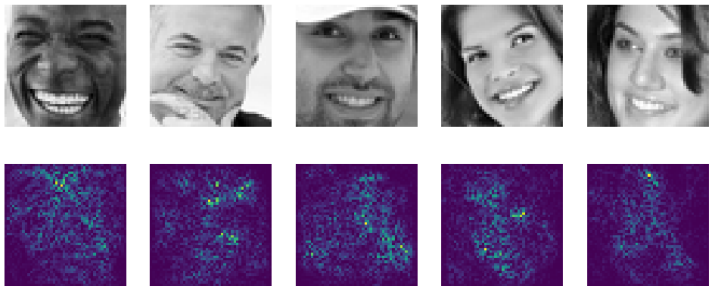


图 10: Saliency Map - Happy

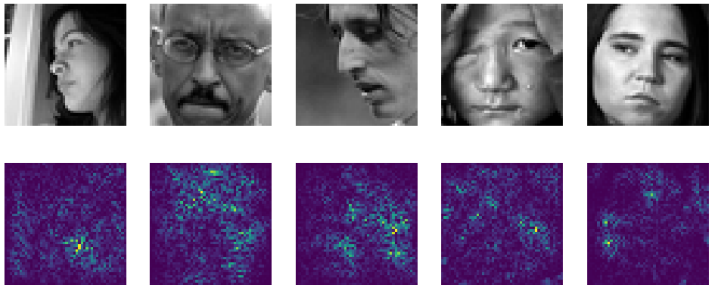


图 11: Saliency Map - Sad

观察得，峰值对应的位置多为

a. 面部肌肉，特别是集中在脸部肌肉、口部肌肉及眼部肌肉。但值得一提的是，得到的显著性图峰值落在眼部的情况非常少，反倒是针对皱眉等相关肌肉动作反应较大。

b. 手部动作。如实例图，有手部动作的图片都在相应的手部动作位置得到了显著性表现。

3 利用 gradient ascent 观察卷积层的激活特性

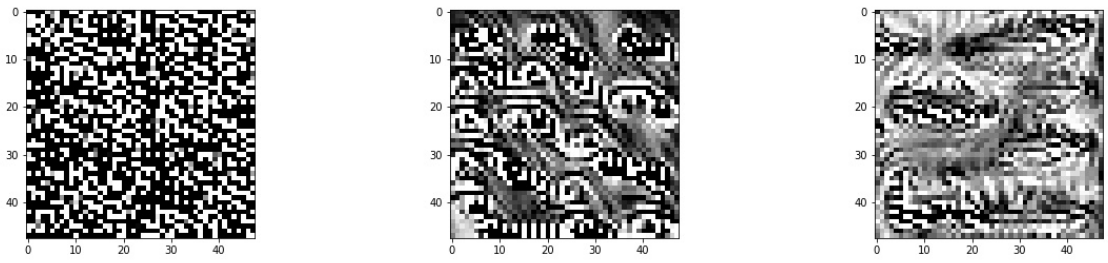
原理：对于特定层的 filter，利用其输出矩阵的元素和表示激活情况，和越大则激活程度越高。

实现：随机初始化图像，将输入图像传入已经训练好的 Net model，首先计算特定层的 filter 的输出矩阵元素及其和 loss，然后对 loss 进行反向传播。

```
428  # gradient ascent
429  def gradient_ascent(net, pic, lr, decay, epochs, stage):
430      for round in range(0, epochs):
431          #计算特定层的filter的输出矩阵元素及其和的相反数loss
432          loss = net.stage_calculate(stage, pic).sum() * (-1)
433          #对loss进行反向传播
434          loss.backward()
435          with torch.no_grad():
436              pic.data.sub_(lr * pic.grad)
437              for i in range(0, 48):
438                  for j in range(0, 48):
439                      if pic[0][0][i][j] < 0:
440                          pic[0][0][i][j] = 0
441                      elif pic[0][0][i][j] > 255:
442                          pic[0][0][i][j] = 255
443          pic.grad.zero_()
444          lr = lr * decay
445      return pic
```

图 12: Compute Gradient Ascent

如下为效果较为显著的纹理图激活情况:



(a) stage2 (b) stage3 (c) stage4

图 13: Active Situation

上图分散在整个网络的不同位置，但都具有明显、易分辨的纹理，表明该纹理对该层 filter 对具有高激活特性。

4 分析模型对表情的判断方式

根据结果绘制模糊矩阵，如下图

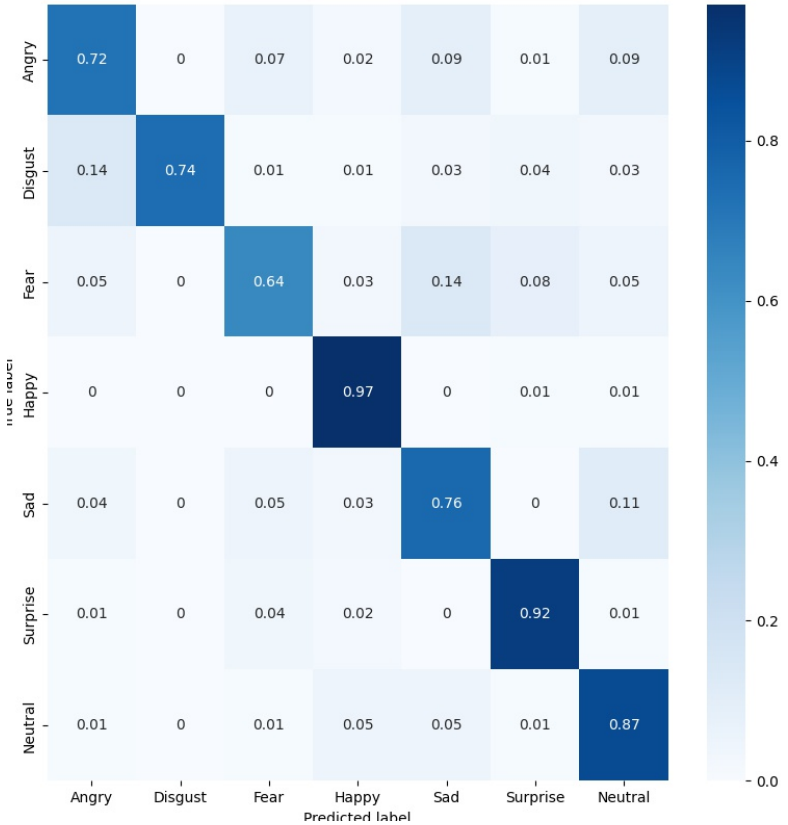


图 14: Confusion matrix

观察得，在 7 种 label 中，Happy 的分类准确率是最高的，而 Fear 是最低的。通过观察原图，原因之一在于 Happy 本就是最容易区分的一种情绪；而 Fear 的确更为难以辨认，且有些 label 为 Fear 的图片在人眼看来不应是这种情绪。

图 15: Wrong Labelled(Fear) Data

5 观察模型训练

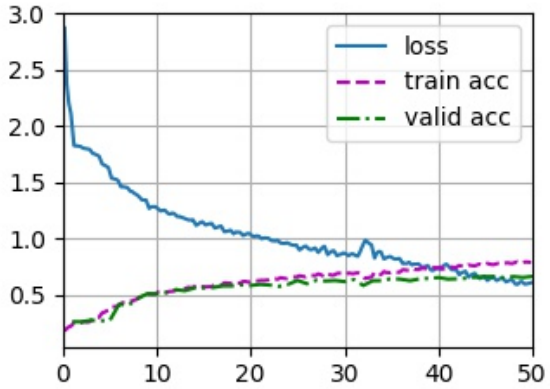


图 16: Train Result

在每一轮的训练中，记录 loss,train_acc 和 valid_acc 并绘制散点图，呈现如下：

观察得，训练集和验证集的正确率都在逐步上升，说明对参数 batch_size 和 train_lr 的选取较为合适。在训练轮次 40 之后时验证集正确率出现不再增长的趋势，说明应该在这一轮附近停止训练，以取得更高的性价比。

6 参考资料

[1]www.d2l.ai

[2]https://oldpan.me/archives/pytorch-visualising-image-classification-models-and-saliency-maps

[3]https://zhuanlan.zhihu.com/p/263527295

[4]https://zhuanlan.zhihu.com/p/180554948

[5]https://wizardforcel.gitbooks.io/pytorch-04-doc/content/36.html