華中科技大學

电子信息与通信学院
《数字图像处理课设》
课程设计报告

小组成员 陆国航 易子闳 赵永辉

班　　级 种子 1601 班

时　　间 2019 年 5 月 17 日

目录

1 题目理解

组织病理学癌症检测 Histopathologic Cancer Detection

需要识别从较大的数字病理扫描中获取的小图像补片中的转移性癌症。此竞赛的数据是PatchCamelyon（PCam）基准数据集的略微修改版本（原始 PCam 数据集由于其概率抽样而包含重复图像，但是，在 Kaggle 上呈现的版本不包含重复项）。PCam 数据集将临床相关的转移检测任务打包成为，二分类任务，类似于 CIFAR-10 和 MNIST。模型可以在几个小时内在单个 GPU 上轻松训练，并在 Camelyon16 肿瘤检测和整个幻灯片图像诊断任务中获得竞争分数。此外，任务难度和易处理性之间的较为平衡，可以学习研究基础机器学习模型不确定性和可解释性。

这是一个二值分类问题，需要确认 96x96 大小的图片中是否存在癌细胞，但是对于阳性样本（癌细胞样本）的标记是根据图片中心的 32x32 大小区域中的切片情况。所以可能需要我们对图片进行剪裁，去除标记中心外围的干扰。

2 数据处理

由上节分析可知，具有不规则形状的细胞核的细胞可以被认为是转移的癌细胞。对于肉眼来说，尚难以分辨，现在的问题是如何将图片做处理，能够使其被模型读取理解变得更容易。

2.1 图像尺寸

由于图像的标签只受中心区域(32 x 32px)的影响，因此我们的初步想法是将图片裁剪，只保留中心区域（32 x 32 px）。但是后面经过分析学习其他类似的情况，考虑到按照上述方法实施可能会丢失些有用的信息，比如周围的环境信息。而这部分信息很大程度上可能是我们所需要的。为此我们也设计了实验进行验证。得出的结论是用 32 x 32px 的大小和 48 x 48px 的大小作比较，前者得出的准确率较高。因此我们采用 48 x 48px 的尺寸进行训练。对于大于 48 x 48px 的尺寸我们采用了 64 x 64 px 的大小进行对比训练，发现准确率稍低于 48 x 48px。但由于没有采用更多彩的尺寸逼近，所以无法确定 48 x 48px 是否是最佳尺寸。但可以确定的是 48 x 48px 在一个最佳尺寸范围内。

2.2 图像增强

首先，我们可以检查数据是否包含坏数据(过于分散或损坏)，删除这些数据以提高整体质量。

为了出现过度拟合的情况，采用了如下几种方式：利用更多的数据，交叉验证，对图像进行增强处理。对于图像的操作，采用了 OpenCV。对于图像的处理采用了以下几种方式：

1. 随机旋转
2. 随机裁剪
3. 水平或竖直翻转
4. 修改亮度
5. 修改对比度

以上图像增强部分我们参考了一些其他人的处理方式，对比发现上述几个条件的改变能够在一定程度上增强数据集的质量。处理方式如下

```
def readstrengthImage(path):
    # 用于图像读取增强
    # OpenCV reads the image in bgr format by default
    bgr_img = cv2.imread(path)
    # We flip it to rgb for visualization purposes
    b,g,r = cv2.split(bgr_img)
    rgb_img = cv2.merge([r,g,b])

    # 随机旋转
    rotation = random.randint(-RANDOM_ROTATION,RANDOM_ROTATION)
    if(RANDOM_90_DEG_TURN == 1):
        rotation += random.randint(-1,1) * 90
    M = cv2.getRotationMatrix2D((48,48),rotation,1)   # the center point is the rotation anchor
    rgb_img = cv2.warpAffine(rgb_img,M,(96,96))

    # 随机平移 x,y-shift
    x = random.randint(-RANDOM_SHIFT, RANDOM_SHIFT)
    y = random.randint(-RANDOM_SHIFT, RANDOM_SHIFT)

    # crop to center and normalize to 0-1 range
    start_crop = (ORIGINAL_SIZE - CROP_SIZE) // 2
    end_crop = start_crop + CROP_SIZE
    rgb_img = rgb_img[(start_crop + x):(end_crop + x), (start_crop + y):(end_crop + y)] / 255

    # 随机对比度
    flip_hor = bool(random.getrandbits(1))
    flip_ver = bool(random.getrandbits(1))
    if(flip_hor):
        rgb_img = rgb_img[:, ::-1]
    if(flip_ver):
        rgb_img = rgb_img[::-1, :]

    # 随机亮度
```

```
    br = random.randint(-
RANDOM_BRIGHTNESS, RANDOM_BRIGHTNESS) / 100.
    rgb_img = rgb_img + br

    # 随机对比度
    cr = 1.0 + random.randint(-
RANDOM_CONTRAST, RANDOM_CONTRAST) / 100.
    rgb_img = rgb_img * cr

    # clip values to 0-1 range
    rgb_img = np.clip(rgb_img, 0, 1.0)

    return rgb_img
```

为了进一步增强数据集的质量，将数据集中过于黑暗或者过曝的图像删除。不过后来对比发现在删除这些无效数据前后准确率并没有可见的提升。猜测是由于无效数据占比较少的原因

2.3 数据集处理

为了进行交叉验证，将训练数据分成 90%的训练和 10%的验证部分。

3 模型选择

我们使用 fastai 框架可以快速建立模型，但是在对模型的选择上我们需要进行对比实验选择对这个问题变现比较好的模型。查阅相关资料后，我们选择了部分近几年提出的 cnn 模型。然后在同样数据集和训练次数下进行对比实验，从中选择表现较好的模型进行参数调优。

3.1 密集连接网络(DenseNet)

Densenet 是 2017 CVPR 最佳论文中提出的一种架构，其网络结构不算复杂，密集连接结构缓解了深层网络梯度消失的问题，加强特征传播，鼓励重复利用参数，减小了参数量。

在 50000 样本量的情况下，densenet169 准确率达到了 88.2%

```
ARCH = densenet169
def getLearner():
    return cnn_learner(imgDataBunch, ARCH, pretrained=True, path='
.', metrics=accuracy, ps=0.5, callback_fns=ShowGraph)

learner = getLearner()
```

```
learner.fit(1)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.312675	0.285903	0.882400	38:23

3.2 残差网络(ResNet)

ResNet 是 2015 年提出的新架构，它的结构区别于传统的 cnn，提出了新思路。ResNet 的一个重要的思想是：输出的是两个连续的卷积层，并且输入时绕到下一层去。这让 ResNet 的深度可达 1000 层以上。可以保留丰富的高维信息。

在 50000 样本量的情况下，resnet18 准确率达到了 86.4%

```
ARCH = resnet18
def getLearner():
    return cnn_learner(imgDataBunch, ARCH, pretrained=True, path='
.', metrics=accuracy, ps=0.5, callback_fns=ShowGraph)

learner = getLearner()
learner.fit(1)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.363527	0.323169	0.864400	14:49

3.3 压缩网络(SqueezeNet)

SqueezeNet 于 2016 年提出，相较于 AlexNet 它在相同精度的条件下更节约内存空间带来更高效的训练效果。

在 50000 样本量的情况下，resnet18 准确率达到了 82.2%

```
ARCH = squeeze1_0
def getLearner():
    return cnn_learner(imgDataBunch, ARCH, pretrained=True, path='
.', metrics=accuracy, ps=0.5, callback_fns=ShowGraph)

learner = getLearner()
learner.fit(1)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.406343	0.387174	0.822400	15:41

在上述网络模型中我们选则了 DenseNet 模型，在相同数量样本下，它更准确，在参数调优后有更大的提升空间。

4 参数调优

在选择适合的模型后，需要进行参数调优来优化模型，而所有参数中 learn rate 是最重要的参数之一。

4.1 Weight Decay 调整

首先我们进行 WD（weight decay）d 的选择，由 cnn 权重更新公式：

$$W(t+1) = w(t) - lr * wd * W(t)$$

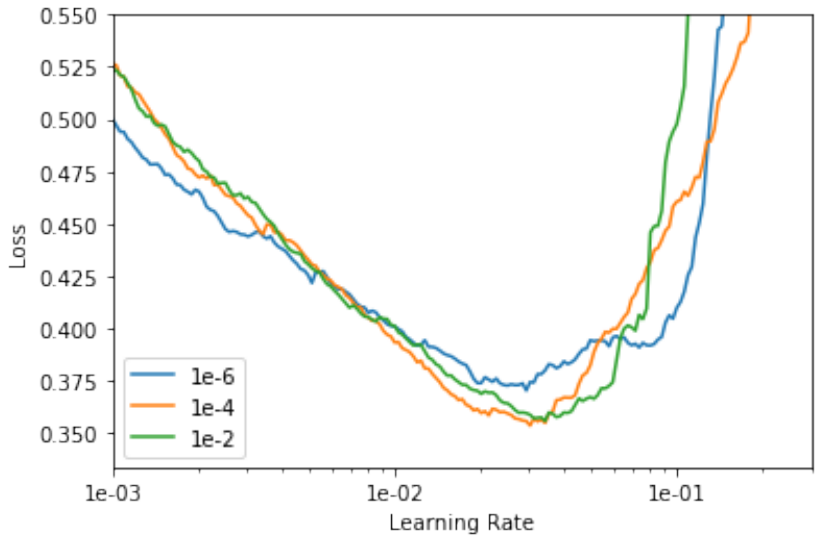
可知，WD 用于防止过拟合。我们需要寻找到不同 WD 下 lr 和 Loss 之间的关系，在较大的 WD 下，可以允许模型以较高的学习率进行训练，可以提高学习训练效率。

```
# WEIGHT DECAY = 1e-6
learner.lr_find(wd=1e-6, num_it=iter_count)

# WEIGHT DECAY = 1e-4
learner.lr_find(wd=1e-4, num_it=iter_count)

# WEIGHT DECAY = 1e-2
learner.lr_find(wd=1e-2, num_it=iter_count)
```

分别在 WD = 1e-6、1e-4、1e-2 的情况下画出学习率和 loss 之间的关系如下图：



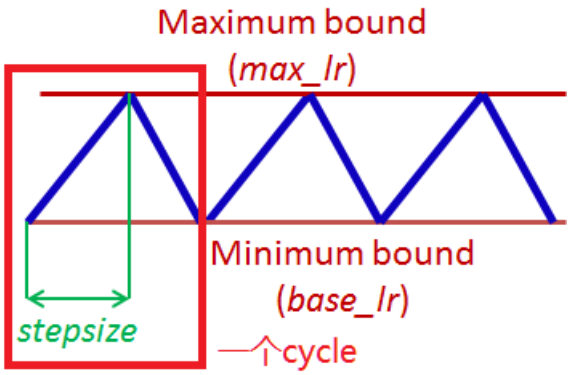
观察图中橙色曲线可以看出在 WD = 1e-4 时 loss 有最低值，可以让我们用比较大的学习率进行训练。

4.2 学习率调整

在选定 WD = 1e-4 条件下，对于学习率的调整有多种策略；这些策略可以大致分为 3 类，一类是固定调整策略例如基于迭代次数的指数衰减等；这类方法不一定能适应不同数据集的不同特种，导致学习效果较差。第二类是自适应调整方法，可以解决上一类的问题但是计算成本较高。第三类方法是周期性学习率调优。

4.2.1 循环学习率(CLR)

CLR 方法不是使用固定的或降低的学习速率，而是允许学习速率在合理的最小和最大界限之间连续振荡。一个 CLR 循环包括两个步骤；学习率增加的学习率和学习率降低的学习率。每个步骤都有一个大小（称为步长），这是学习速率增加或减少的迭代次数（例如 1k，5k 等）；两个步骤形成一个循环。如下图所示：



CLR 方法计算成本不高，虽然会暂时得不到较好的网络，但是整体来说会提高网络的效果。有相关研究证明在 cifar-10 数据集上 25000 次迭代的达到的准去率于传统方法迭代 70000 次的结果，提高了学习效率。

这个方法的关键在于对 max_lr 的选择，在满足 loss 仍在下降的条件下 max_lr 需要尽可能的大，基于上面关于 lr 和 loss 的关系图，我们选择了 max_lr 为 2e-2，在橙色曲线的这个点上 loss 仍在下降。

5 训练策略

我们采用分段训练的方法来训练模型。对于模型头部用较大的 lr 进行训练，并将模型后端进行冻结；之后再用较小的 lr 对模型后端进行训练，前面已经训练好的部分参数不变。

由于硬件设备的原因最终模型并没有把所有数据训练完，最后仅跑完 1/8 的数据集得到

的准确率为 91.8%，优于最开始的 88.2%说明调优效果较明显。

```
# 只训练头部其它部分冻结
max_lr = 2e-2
wd = 1e-4
# CLR 策略
learner.fit_one_cycle(cyc_len=8, max_lr=max_lr, wd=wd)

# 交叉验证
interp = ClassificationInterpretation.from_learner(learner)
interp.plot_confusion_matrix(title='Confusion matrix')

# 保存模型
learner.save(MODEL_PATH + '_stage1')

# 加载模型
learner.load(MODEL_PATH + '_stage1')

# 解冻模型继续训练
learner.unfreeze()

# 减小学习率
learner.fit_one_cycle(cyc_len=12, max_lr=slice(4e-5,4e-4))

# 再次交叉验证
interp = ClassificationInterpretation.from_learner(learner)
interp.plot_confusion_matrix(title='Confusion matrix')

# 保存模型
learner.save(MODEL_PATH + '_stage2')
```

build model

12.50% [1/8 2:43:14<19:02:42]

epoch	train_loss	valid_loss	accuracy	time
0	0.217427	0.215302	0.917875	2:43:14

6 总结

本次机器学习的课设是对之前课程内容的一个考察。通过本次课程的联系，了解到了机器学习在工程方面的应用价值。我们的选题为癌细胞的识别检测，利用机器学习的方法可以辅助医学研究人员确定和复核细胞是否发生病变，从而帮助其提高诊断的准确率。

在本次实验中我们也遇到了一些问题，由于这是第一次接触到一个偏向于实践应用的深度学习的题目，在题目的需求分析方面，刚开始是没有思路的，后面经过对其它解题思路的分析慢慢找到了方向。对于数据处理方面我们在之前的课程中是从来没有接触过的。如何处理图像才能使之更好的进行训练，而同时又不至于使得训练结果过拟合。在模型的选择和设计方面，我们对于一些常见的模型进行了了解，虽然如此，但并不太清楚到底哪个模型更适合于解决这个问题。于是我们采用了一种简单粗暴的方式。先用少量的数据用不同的模型算法进行测试，取最优结果进行全部数据的训练。我们也明白这可能不是最优的解决方案，可能后续还要一些理论的学习和支撑。