同济大学

TONGJI UNIVERSITY

《数据挖掘》

分类部分（小作业）

实验名称	SDNET 数据集分类实验
小组成员	林澍晖（1751751）
学院（系）	土木工程学院
专　　业	土木工程
任课教师	李洁
日　　期	2021 年 6 月 15 日

1. 实验目的

本课题使用的数据集来自于数据分析与数据挖掘竞赛 Kaggle，该竞赛为数据科学领域著名的国际性赛事之一。课题使用的数据集为带标签的图像数据集，包含带有裂痕和不带有裂痕的桥梁、墙和人行道图片。课题的目标为对于目标数据集，搭建相应的传统机器学习模型和深度神经网络模型，完成基于机器学习的图片分类任务，实现一个具有较优性能和较强稳定性的模型。整个实验课题包含数据准备、数据预处理、模型搭建、模型训练、模型优化、模型检测、实验总结等过程。

2. 数据准备与清洗

2.1 数据获取

数据来源是 Kaggle 竞赛的“Structural Defects Network (SDNET) 2018”，在 Kaggle 网站上直接下载即可。该数据集包含有桥梁表面、步行道表面和墙面三种不同的图片，每种图片都包含有带裂痕的图片和不带裂痕的图片共两类图片。数据集共包含 56000 张图片，其中图片上的建筑裂痕最窄为 0.06 毫米，最宽为 25 毫米。数据集中的部分图片可能存在着一定的遮挡干扰，例如阴影、建筑表面粗糙、建筑表面脱落、拍摄角度变换、建筑表面的孔洞和背景噪声，目标数据集的分类任务即为区分带裂痕和不带裂痕的图片。

2.2 数据准备

裂缝图像一共 8484 张、无裂缝图像一共 47608 张，比例为 1：5.6，数据倾斜较严重。

在深度学习部分，设置了三组对照实验，分别是纯 Resnet34 实验、Resnet34+迁移学习、Resnet34+迁移学习+数据增强；在纯 Resnet34 实验和 Resnet34+迁移学习实验中，对数据做了中心裁剪为 256*256 图像，再缩放大小到 224*224，最后进行标准化处理。在 Resnet34+迁移学习+数据增强实验中，对数据做了增强，包括水平镜像、垂直镜像、旋转任意角度、调整明亮度、调整对比度、调整渗透率、调整色调、仿射变换等操作，并且上述操作都 RandomApply 以一定的概率执行，最终达到倍增裂缝图像数据量的效果，裂缝图像扩大为 8484*2 张。

在机器学习部分，对图像进行了缩放操作，缩放后比例统一为 256*256，之后再对图片进行和黑白处理，将 3 通道压缩为单通道，并对图片进行了标准化，最后将图片扁平化。

2.3 数据清洗

数据集是图片数据集，并采用文件名来分类。一方面，采取了人工抽样检测的方式探索了数据集的可靠性，另一方面考虑到数据集来源 Kaggle 竞赛，作为图片数据集，数据格式统一，不存

在缺失值，标签明确，数据集的质量有保障。因此，本次实验未对数据集进行清洗。

3. 模型搭建

3.1 基于 Resnet34 模型的深度学习

3.1.1 Resnet 模型介绍

ResNet 网络是参考了 VGG19 网络，在其基础上进行了修改，并通过短路机制加入了残差单元。变化主要体现在 ResNet 直接使用 stride=2 的卷积做下采样，并且用 global average pool 层替换了全连接层。ResNet 的一个重要设计原则是：当 feature map 大小降低一半时，feature map 的数量增加一倍，这保持了网络层的复杂度。ResNet 相比普通网络每两层间增加了短路机制，这就形成了残差学习，其中虚线表示 feature map 数量发生了改变。当网络更深时，其进行的是三层间的残差学习，三层卷积核分别是 1x1，3x3 和 1x1，一个值得注意的是隐含层的 feature map 数量是比较小的，并且是输出 feature map 数量的 1/4。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 1 常用的 Resnet 框架

ResNet 使用两种残差单元，如图 2 所示。左图对应的是浅层网络，而右图对应的是深层网络。对于短路连接，当输入和输出维度一致时，可以直接将输入加到输出上。但是当维度不一致时（对应的是维度增加一倍），这就不能直接相加。有两种策略：（1）采用 zero-padding 增加维度，此时一般要先做一个 downsamp，可以采用 strde=2 的 pooling，这样不会增加参数；（2）采用新的映射（projection shortcut），一般采用 1x1 的卷积，这样会增加参数，也会增加计算量。短路连接除了直接使用恒等映射，当然都可以采用 projection shortcut。

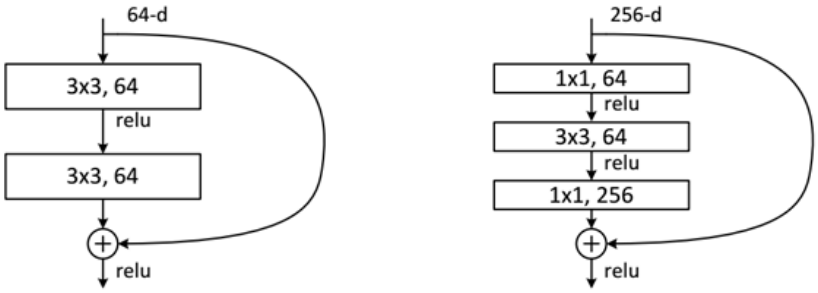


图 2 Resnet 中两种残差单元

该次实验中，采用了 Resnet34 网络框架，残差单元采用的是图 2 中的左图。

3.1.2 迁移学习

迁移学习是一种机器学习方法，就是把为任务 A 开发的模型作为初始点，重新使用在为任务 B 开发模型的过程中。通过迁移学习，可以在优秀模型的参数上，针对自己的数据集的特点进行二次训练，在提高训练速度的同时，提高了准确度。实验中，迁移学习用到的模型是 pytorch 提供的 Resnet34 模型，下载地址为：https://download.pytorch.org/models/resnet34-333f7ec4.pth。

3.1.3 基于 pytorch 框架搭建 Resnet34

在 SDNET 竞赛中，一方面考虑到问题相较于一般分类问题更为抽象复杂，另一方面考虑到时间和算力的约束，故采用了 Resnet34 网络结构。

模型的搭建源码见 model.py 文件。 基本结构有 BasicBlock 类和 Bottlenec 类，它们中有不同的卷积层、激活函数与卷积层与批标准化的相对位置也不同，我们通过 forward 函数来拟定一个向前转播的过程，其中也定义了 shortcut 实现方法。在本次实验中，我们采用 BasicBlock 类作为 Resnet34 的基本模块，Bottlenec 类适用于 50 层以上的 Resnet 的基本模块。

```
class BasicBlock(nn.Module):
    expansion = 1
    def __init__(self, in_channel, out_channel, stride=1, downsample=None, **kwargs):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=in_channel, out_channels=out_channel,
                               kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(in_channels=out_channel, out_channels=out_channel,
                               kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        if self.downsample is not None:
            identity = self.downsample(x)
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        out += identity
        out = self.relu(out)
        return out
```

图 3 BasicBlock 类定义

接下来是 Resnet 类，其中最重要的是实现_make_layer 函数，用于使用 basicblock 制作 resnet 的每一层，然后在 Resnet 的初始化函数中通过调用_make_layer 函数生成 Resnet 框架。

3.2 基于被动攻击算法的机器学习

3.2.1 PassiveAggressiveClassifier 模型介绍

被动攻击算法适用于大规模学习的算法。它和感知器一样不需要学习率。然而，与感知器相反，它有一个正则化参数 c。该算法支持大批量数据的训练，操作是将数据分为多个 batch，进行增量学习。

3.2.2 PassiveAggressiveClassifier 的实现

```
from sklearn.linear_model import PassiveAggressiveClassifier
clf = PassiveAggressiveClassifier(max_iter=1000, tol=1e-3)

epochs = 20
train_score = 0
batch_size = int(len(train_list) / epochs)

print("开始训练第 1 / 20 批训练")
X, y= get_X_y(train_list[:batch_size], "MachineLearning")
clf.partial_fit(X, y, classes=np.unique(y))
train_score += clf.score(X, y)

for i in range(1,epochs):
    print("开始训练第 %s / 20 批训练" % (i+1))
    X, y = get_X_y(train_list[i*batch_size+1:(i+1)*batch_size], "MachineLearning")
    clf.partial_fit(X, y)
    train_score += clf.score(X, y)
```

3.3 基于 SGD 的机器学习

3.3.1 SGD 模型介绍

随机梯度下降(Stochastic Gradient Descent, SGD)是梯度下降算法的一个扩展。随机梯度下降法(Stochastic Gradient Descent, SGD)：由于批量梯度下降法在更新每一个参数时，都需要所有的训练样本，所以训练过程会随着样本数量的加大而变得异常的缓慢。随机梯度下降法正是为了解决批量梯度下降法这一弊端而提出的。随机梯度下降是通过每个样本来迭代更新一次。SGD 伴随的一个问题是噪音较 BGD 要多，使得 SGD 并不是每次迭代都向着最优化方向进行。

3.3.2 SGD 实现

```
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier(max_iter=1000, tol=1e-3)

epochs = 20
train_score = 0
batch_size = int(len(train_list) / epochs)

print("开始训练第 1 / 20 批训练")
X, y= get_X_y(train_list[:batch_size], "MachineLearning")
clf.partial_fit(X, y, classes=np.unique(y))
train_score += clf.score(X, y)

for i in range(1,epochs):
    print("开始训练第 %s / 20 批训练" % (i+1))
    X, y = get_X_y(train_list[i*batch_size+1:(i+1)*batch_size], "MachineLearning")
    clf.partial_fit(X, y)
    train_score += clf.score(X, y)
```

4.模型训练测试

4.1训练环境与配置

租用了矩池云上面的 GPU 来进行计算，主机参数如下：

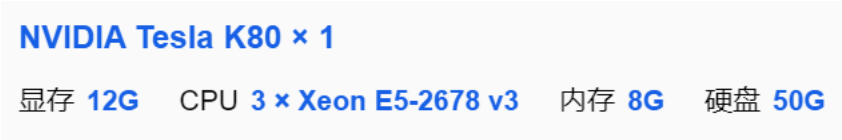


图 4 主机参数

系统镜像选择如下：

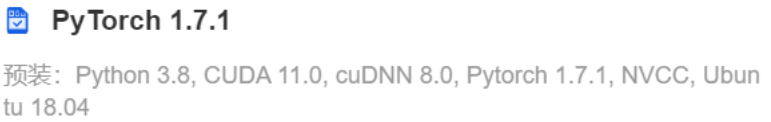


图 5 系统镜像

在此基础上，安装了 tqdm 用于训练进度可视化，安装了 tensorboardX 用于模型评价指标可视化和训练过程可视化。

4.2 模型训练过程

4.2.1 Resnet34 模型

对于纯 Resnet34 模型，训练 40 个 epoch，学习率为 1e-4；对于 Resnet34+迁移学习模型，训练 10 个 epoch，学习率为 5e-5；对于 Resnet34+数据增强+迁移学习，训练 10 个 epoch，学习率为 5e-6。一下分别展示三个模型训练过程的 5 个 epoch。

```
train epoch[36/40] : 100%|██████████| 1603/1603 [11:15<00:00,  2.37it/s]
valid epoch[36/40]: 100%|██████████| 401/401 [00:58<00:00,  6.85it/s]
  0%|          | 0/1603 [00:00<?, ?it/s]
[epoch 36]
 train_loss: 0.011  val_loss: 0.506
 val_accuracy: 0.920
 precision_cracked: 0.790  precision_nonCracked: 0.939
 recall_cracked:0.650  recall_nonCracked:0.969
train epoch[37/40] : 100%|██████████| 1603/1603 [11:16<00:00,  2.37it/s]
valid epoch[37/40]: 100%|██████████| 401/401 [00:58<00:00,  6.84it/s]
  0%|          | 0/1603 [00:00<?, ?it/s]
[epoch 37]
 train_loss: 0.013  val_loss: 0.479
 val_accuracy: 0.923
 precision_cracked: 0.804  precision_nonCracked: 0.939
 recall_cracked:0.651  recall_nonCracked:0.971
train epoch[38/40] : 100%|██████████| 1603/1603 [11:16<00:00,  2.37it/s]
valid epoch[38/40]: 100%|██████████| 401/401 [00:58<00:00,  6.84it/s]
  0%|          | 0/1603 [00:00<?, ?it/s]
[epoch 38]
 train_loss: 0.012  val_loss: 0.507
 val_accuracy: 0.922
 precision_cracked: 0.830  precision_nonCracked: 0.934
 recall_cracked:0.617  recall_nonCracked:0.977
train epoch[39/40] : 100%|██████████| 1603/1603 [11:15<00:00,  2.37it/s]
valid epoch[39/40]: 100%|██████████| 401/401 [00:58<00:00,  6.82it/s]
  0%|          | 0/1603 [00:00<?, ?it/s]
[epoch 39]
 train_loss: 0.012  val_loss: 0.474
 val_accuracy: 0.923
 precision_cracked: 0.814  precision_nonCracked: 0.938
 recall_cracked:0.645  recall_nonCracked:0.973
train epoch[40/40] : 100%|██████████| 1603/1603 [11:15<00:00,  2.37it/s]
valid epoch[40/40]: 100%|██████████| 401/401 [00:58<00:00,  6.85it/s]
[epoch 40]
 train_loss: 0.012  val_loss: 0.453
 val_accuracy: 0.922
 precision_cracked: 0.800  precision_nonCracked: 0.940
 recall_cracked:0.654  recall_nonCracked:0.971
Finished Training
```

图 6　纯 Resnet34 训练过程

```
train epoch[1/10] : 100%|██████████| 1403/1403 [10:35<00:00,  2.21it/s]
valid epoch[1/10]: 100%|██████████| 351/351 [00:53<00:00,  6.51it/s]
[epoch 1]
 train_loss: 0.223  val_loss: 0.192
 val_accuracy: 0.932
 precision_cracked: 0.960  precision_nonCracked: 0.929
 recall_cracked:0.568  recall_nonCracked:0.996
train epoch[2/10] : 100%|██████████| 1403/1403 [10:35<00:00,  2.21it/s]
valid epoch[2/10]: 100%|██████████| 351/351 [00:53<00:00,  6.54it/s]
[epoch 2]
 train_loss: 0.163  val_loss: 0.175
 val_accuracy: 0.941
 precision_cracked: 0.854  precision_nonCracked: 0.953
 recall_cracked:0.727  recall_nonCracked:0.978
train epoch[3/10] : 100%|██████████| 1403/1403 [10:37<00:00,  2.20it/s]
valid epoch[3/10]: 100%|██████████| 351/351 [00:53<00:00,  6.55it/s]
  0%|          | 0/1403 [00:00<?, ?it/s]
[epoch 3]
 train_loss: 0.127  val_loss: 0.189
 val_accuracy: 0.939
 precision_cracked: 0.834  precision_nonCracked: 0.955
 recall_cracked:0.739  recall_nonCracked:0.974
train epoch[4/10] : 100%|██████████| 1403/1403 [10:38<00:00,  2.20it/s]
valid epoch[4/10]: 100%|██████████| 351/351 [00:53<00:00,  6.53it/s]
[epoch 4]
 train_loss: 0.090  val_loss: 0.183
 val_accuracy: 0.943
 precision_cracked: 0.892  precision_nonCracked: 0.950
 recall_cracked:0.704  recall_nonCracked:0.985
train epoch[5/10] : 100%|██████████| 1403/1403 [10:37<00:00,  2.20it/s]
valid epoch[5/10]: 100%|██████████| 351/351 [00:54<00:00,  6.43it/s]
  0%|          | 0/1403 [00:00<?, ?it/s]
[epoch 5]
 train_loss: 0.056  val_loss: 0.225
 val_accuracy: 0.940
 precision_cracked: 0.895  precision_nonCracked: 0.945
 recall_cracked:0.675  recall_nonCracked:0.986
```

图 7 Resnet34+迁移学习训练过程

```
train epoch[1/10] : 100%|██████████| 1615/1615 [12:13<00:00,  2.20it/s]
valid epoch[1/10]: 100%|██████████| 404/404 [01:01<00:00,  6.58it/s]
[epoch 1]
 train_loss: 0.341  val_loss: 0.238
 val_accuracy: 0.909
 precision_cracked: 0.903  precision_nonCracked: 0.911
 recall_cracked:0.727  recall_nonCracked:0.973
train epoch[2/10] : 100%|██████████| 1615/1615 [12:12<00:00,  2.21it/s]
valid epoch[2/10]: 100%|██████████| 404/404 [01:01<00:00,  6.59it/s]
[epoch 2]
 train_loss: 0.219  val_loss: 0.204
 val_accuracy: 0.921
 precision_cracked: 0.896  precision_nonCracked: 0.929
 recall_cracked:0.787  recall_nonCracked:0.968
train epoch[3/10] : 100%|██████████| 1615/1615 [12:12<00:00,  2.20it/s]
valid epoch[3/10]: 100%|██████████| 404/404 [01:01<00:00,  6.58it/s]
[epoch 3]
 train_loss: 0.158  val_loss: 0.176
 val_accuracy: 0.935
 precision_cracked: 0.919  precision_nonCracked: 0.940
 recall_cracked:0.823  recall_nonCracked:0.975
train epoch[4/10] : 100%|██████████| 1615/1615 [12:12<00:00,  2.20it/s]
valid epoch[4/10]: 100%|██████████| 404/404 [01:01<00:00,  6.59it/s]
[epoch 4]
 train_loss: 0.102  val_loss: 0.155
 val_accuracy: 0.945
 precision_cracked: 0.903  precision_nonCracked: 0.960
 recall_cracked:0.883  recall_nonCracked:0.967
train epoch[5/10] : 100%|██████████| 1615/1615 [12:12<00:00,  2.20it/s]
valid epoch[5/10]: 100%|██████████| 404/404 [01:01<00:00,  6.57it/s]
[epoch 5]
 train_loss: 0.065  val_loss: 0.136
 val_accuracy: 0.957
 precision_cracked: 0.913  precision_nonCracked: 0.973
 recall_cracked:0.923  recall_nonCracked:0.969
```

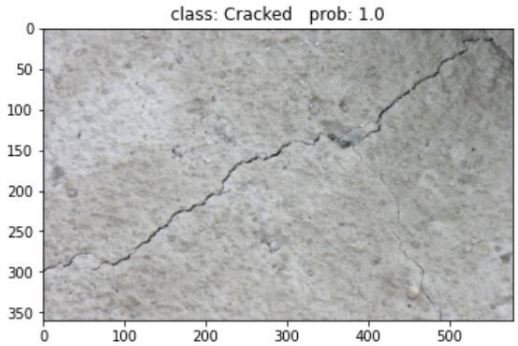
图 8 Resnet34+迁移学习+数据增强的训练过程

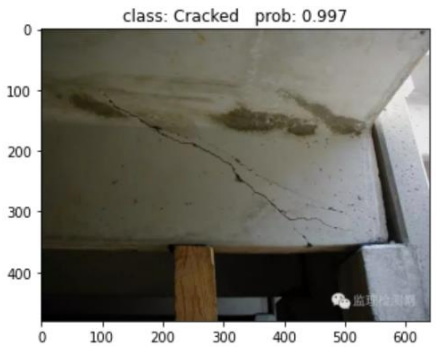
4.2.2 传统机器学习模型训练

如果将数据集一次性读入内存，会导致内存溢出错误，因此采用增量训练的方法对模型进行训练，SGDClassifier 和 PassiveAggressiveClassifier 都提供了 partial_fit 方法进行小批量的增量训练。(具体训练过程见 MachineLearning.ipynp 文件)

4.3 模型测试

选用模型效果效果最好的 Resnet34+迁移学习+数据增强得到的模型来进行预测。从网上随机选取了几张有裂缝、无裂缝的图使用训练好的模型进行测试。





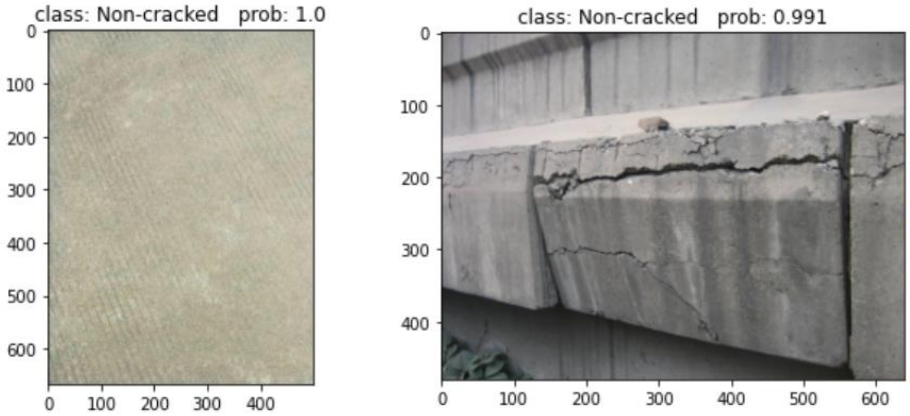


图 9 Resnet34 模型测试

5．结果可视化

5.1 Resnet34 模型实验结果

为了评判模型的好坏并衡量模型是否收敛，设置了损失率、平均准确度、精确度和召回率等指标，并利用 tensorboardX 库，对这些指标进行可视化。一下展示 Resnet34 结合迁移学习并经过数据增强后的各项指标。

从可视化结果很容易看出，模型在什么时候到达最佳状态，继续训练会造成过拟合。

计算损失率时，用的是交叉熵损失函数。交叉熵损失函数是一个平滑函数，其本质是信息理论（information theory）中的交叉熵（cross entropy）在分类问题中的应用。由交叉熵的定义可知，最小化交叉熵等价于最小化观测值和估计值的相对熵（relative entropy），即两者概率分布的 Kullback-Leibler 散度，因此其是一个提供无偏估计的代理损失。交叉熵损失函数是表中使用最广泛的代理损失，对应的分类器例子包括 logistic 回归、人工神经网络和概率输出的支持向量机。

$$L = -[ylog\,\hat{y} + (1-y)log\,(1-\hat{y})]$$

图 10 交叉熵损失函数

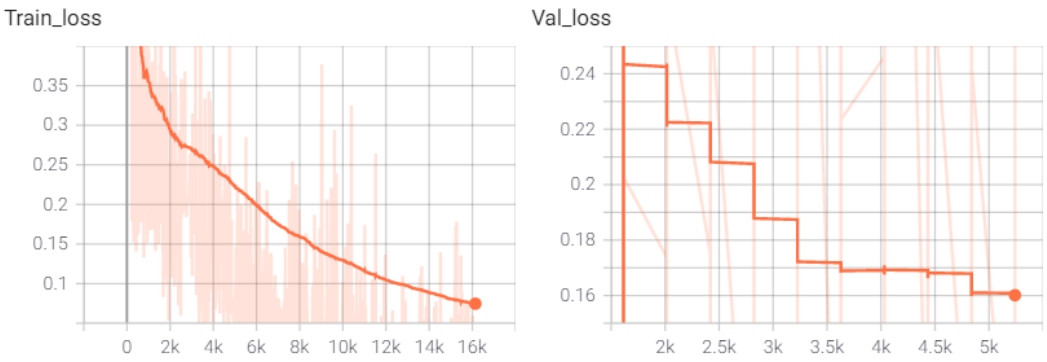


图 11 损失率变化

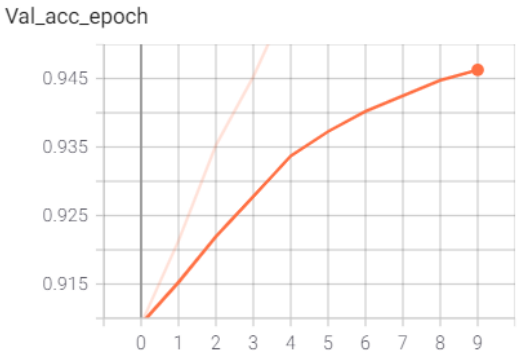


图 12 验证集平均准确度变化

精确度 Precision，是预测为正例的那些数据里预测正确的数据个数比例；召回率 Recall，真实为正例的那些数据里预测正确的数据个数

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives} \qquad precision = \frac{true\ positives}{true\ positives + false\ positives}$$

图 13 Precision 与 Recall 计算公式

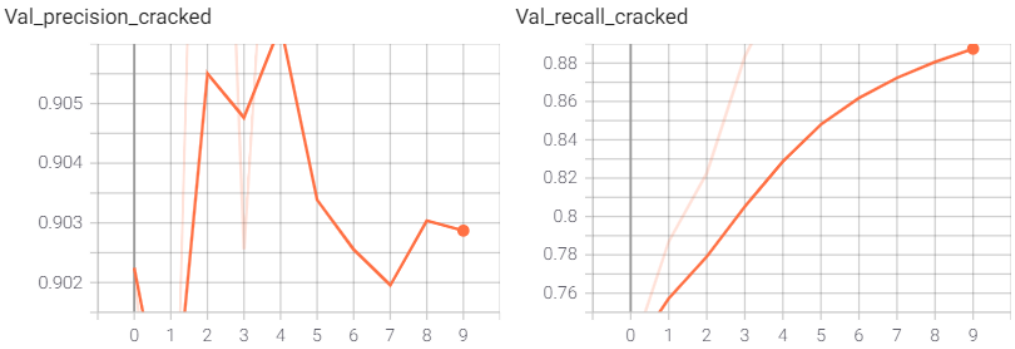


图 14 验证集中裂缝图片识别的精确度和召回率

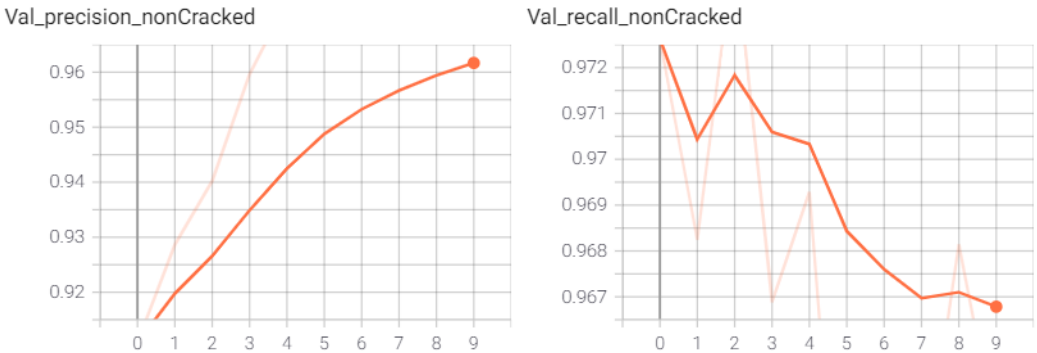


图 15 验证集中非裂缝图片识别的精确度和召回率

装 订 线

5.2 PassiveAggressiveClassifier 模型实验结果

效果十分差，传统的机器学习很难提取出裂缝这种高级特征。测试集精度小于 0.5，还不如乱猜。

```
训练集精度:
0.7484968480156884

测试集精度:
0.15

分类识别报告:
              precision    recall  f1-score   support

           0       1.00      0.01      0.02      1719
           1       0.14      1.00      0.25       281

    accuracy                           0.15      2000
   macro avg       0.57      0.51      0.14      2000
weighted avg       0.88      0.15      0.05      2000


混淆矩阵:
[[  19 1700]
 [   0  281]]
```

图 16 PassiveAggressiveClassifier 实验结果

5.3 SGD 模型实验结果

效果十分差，传统的机器学习很难提取出裂缝这种高级特征。SGD 分类器模型倾向于将几乎所有图片都归为无裂缝。

```
训练集精度:
0.7133402962161741

测试集精度:
0.857

分类识别报告:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      1719
           1       0.14      0.00      0.01       281

    accuracy                           0.86      2000
   macro avg       0.50      0.50      0.46      2000
weighted avg       0.76      0.86      0.79      2000


混淆矩阵:
[[1713    6]
 [ 280    1]]
```

图 16 SGD 实验结果

6．模型优化

对于图片分类而言，深度学习的效果是普遍比传统机器学习好的，因此对于该 SDNET 数据集，我着重对于 Resnet34 网络进行了优化：

6.1 迁移学习

迁移学习是一种机器学习方法，就是把为任务 A 开发的模型作为初始点，重新使用在为任务 B 开发模型的过程中。通过迁移学习，可以在优秀模型的参数上，针对自己的数据集的特点进行二次训练，在提高训练速度的同时，提高了准确度。实验中，迁移学习用到的模型是 pytorch 提供的 Resnet34 模型，下载地址为：https://download.pytorch.org/models/resnet34-333f7ec4.pth。

6.2 数据增强

裂缝图像一共 8484 张、无裂缝图像一共 47608 张，比例为 1：5.6，数据倾斜较严重。造成的后果是，直接采用这样比例的数据进行训练，会造成裂缝图像的召回率难以取得高分，同时模型对于裂缝的识别可能会过拟合。为了解决这种数据倾斜问题，我设置了一个针对 SDNET 比较合理的数据增强框架，包括水平镜像、垂直镜像、旋转任意角度、调整明亮度、调整对比度、调整渗透率、调整色调、仿射变换等操作，使得增强之后裂缝图像的数据量能够增加，也不会因为图片相似造成过拟合。

```
transList = [transforms.RandomHorizontalFlip(p=0.5),  # 水平镜像
             transforms.RandomVerticalFlip(p=0.5),  # 垂直镜像
             transforms.RandomApply(
                 transforms.RandomRotation((-90, 90), resample=False, expand=True, center=None),
                 p=0.5
             ),  # 旋转角度
             transforms.RandomApply(
                 transforms.ColorJitter(brightness=0.5),
                 p=0.5
             ),  # 调整明亮度
             transforms.RandomApply(
                 transforms.ColorJitter(contrast=0.5),
                 p=0.5
             ),  # 调整对比度
             transforms.RandomApply(
                 transforms.ColorJitter(saturation=0.5),
                 p=0.5
             ),  # 调整渗透率
             transforms.RandomApply(
                 transforms.ColorJitter(hue=0.5),
                 p=0.3
             ),  # 调整色调
             transforms.RandomApply(
                 transforms.Grayscale(num_output_channels=3),
                 p=0.1
             ),  # 改为灰度图
             transforms.RandomApply(
                 transforms.RandomAffine(degrees=0, shear=(0, 0, 0, 45)),
                 p=0.2
             )  # 仿射变换
             ]
```

图 17 数据增强框架

6.3 调整学习率

在模型训练的过程中，根据每次训练的结果，反复调整了学习率，并作了经验总结:

train loss 不断下降，test loss 不断下降，说明网络仍在学习;（最好的）

train loss 不断下降，test loss 趋于不变，说明网络过拟合;（max pool 或者正则化）

train loss 趋于不变，test loss 不断下降，说明数据集 100%有问题;（检查 dataset）

train loss 趋于不变，test loss 趋于不变，说明学习遇到瓶颈，需要减小学习率或批量数目;

train loss 不断上升，test loss 不断上升，说明网络结构设计不当，训练超参数设置不当，数据集经过清洗等问题。（最不好的情况）

6.4 调整 batch_size 和 num_workers

这部分的优化主要目的是增块模型训练速度。通过反复测试和比较，最终选定 batch_size=32, num_worders=8 来进行模型训练，它们最符合我租用的云端主机的性能，使得 GPU 的利用率能达到 70%-80%。