机器学习在导航定位技术中的应用
课程报告

指导老师：牛小骥 教授

学生姓名：葛雯斐

学生学号：2018206180011

日期：2018.09.03

目录

第一章　课程背景 4

第二章　选题及内容 4

2.1 选题 4

2.2 竞赛题目 5

2.3 竞赛数据 5

2.4 比赛规则 9

2.5 提交说明 9

第三章　基础概念介绍 11

3.1 keras 简介 11

3.1.1 keras 的傻瓜式训练数据组织模块：`fit_generator` 12

3.1.2 数据增强 13

3.1.3 keras 中设置 Dropout 15

3.1.4 keras 自定义损失函数 16

3.1.5 keras 调用主流的网络模型及其参数 17

3.1.6 keras 优化器选择 18

3.1.7 keras ModelCheckpoint 实现断点续训功能 19

3.1.8 keras 可视化训练过程 20

第四章 主要思路 22

4.1 初赛 22

4.2 复赛 23

第五章 心得体会 26

附录 程序使用的源代码 27

1 源程序 27

2 AUC 27

3 mAP 34

第一章 课程背景

《机器学习在导航定位技术中的应用》是由武汉大学卫星导航定位技术研究中心的牛小骥教授开设的一门研究生课程，该课程为本身是导航定位领域的学生提供一个新的研究思路，开阔了眼界，为我们将机器学习应用在导航定位技术领域之中提供理论知识基础以及提供基本的方向。

在该的课程的学习中，我初次接触机器学习以及深度神经网络，在课程后期牛教授请了很多该领域内的专业人士给我们开了很多讲座，让我对机器学习等有了很大的学习兴趣，课程期间了解了一些基础的理论知识，借暑假空闲的机会，希望能够对相关知识有更加深入的了解。

第二章 选题及内容

2.1 选题

为了能够更快速地掌握机器学习的一些基础知识，上课获取的一些理论知识并没有很深的体会，因此决定通过参加比赛的方式让自己能够有针对性地在有限的时间内收获更多的经验。

天池大数据竞赛系列是阿里云打造的竞赛平台，以打造国际高端算法竞赛，让选手用算法解决社会或业务问题为目标，是一个非常好的学习和竞赛平台。

无锡太湖新城雪浪小镇以物联网新思维为引领，以发展物联网产业为支撑，与江苏的制造业紧密结合，构建制造业与物联网深度融合的基础，唤醒江苏乃至中国传统制造业的全面升级。雪浪小镇首倡并喊响制造业“唤醒计划”，通过一年一度的雪浪大会和世界物联网博览会以及一系列高频度的行业领袖人物和创新创业新锐人物的分享交流活动，为江苏乃至中国的制造业带来物联网的新思维、新理念、新机遇。

2018 雪浪制造挑战赛由江苏省无锡经济开发区（太湖新城）、阿里云计算有限公司联合主办。大赛基于阿里云天池平台，提供数千份精标注布样数

据，以“视觉计算辅助良品检验”为主题，聚焦布匹疵点智能识别，开展大数据与人工智能技术在布匹疵点识别上的应用探索，助力工业制造良品提升。

作为本次大赛的数据提供方，江苏阳光集团提供丰富和完善的布料样本，包括布样、取样环境、疵点判断标准，以及工艺专家的专业指导，从软硬件环境诸多方面提供大赛支撑。

2.2 竞赛题目

布匹疵点检验是纺织行业生产和质量管理的重要环节，目前的人工检验速度慢、劳动强度大，受主观因素影响，缺乏一致性。2016 年我国布匹产量超过 700 亿米，且产量一直处于上升趋势，如果能够将人工智能和计算机视觉技术应用于纺织行业，对纺织行业的价值无疑会是巨大的。

本次大赛要求选手开发算法模型，通过布样影像，基于对布样中疵点形态、长度、面积以及所处位置等的分析，判断瑕疵的种类。通过探索布样疵点精确智能诊断的优秀算法，提升布样疵点检验的准确度，降低对大量人工的依赖，提升布样疵点质检的效果和效率。

阿里云将为参赛选手提供机器学习 PAI 平台，复赛团队可申请使用。入围决赛的参赛团队方案里，必须包含深度学习作为主要算法。

2.3 竞赛数据

本次大赛涵盖了纺织业中素色布的各类重要瑕疵。数据共包括 2 部分：原始图片和瑕疵的标注数据。

训练数据文件结构

a) 我们将提供用于训练的图像数据和标注数据，文件夹结构：

- 正常
- 薄段
- 笔印

…

o 织稀

b） 正常 ： 存放无瑕疵的图像数据，jpeg 编码图像文件。图像文件名如：XXX.jpg

c） 薄段、笔印、…、织稀：按瑕疵类别分别存放瑕疵原始图片和用矩形框进行瑕疵标注的位置数据。图像文件 jpeg 编码。标注文件采用 xml 格式，其中 filename 字段是图像的文件名，name 字段是瑕疵的类别，bndbox 记录了矩形框左上角和右下角的位置。图像左上角为（0，0）点，向右 x 值增加，向下 y 值增加。

原始图片和标注 xml 文件的示意如图 1.1-1.5。

图 1.1 训练数据中布匹瑕疵类别按照子文件夹分布（每一个瑕疵名称文件夹下面的每一个 jpg 图片对应一个同名 xml 标签文件，子文件夹“正常”下面只有正常的布匹照片，没有 xml 文件）

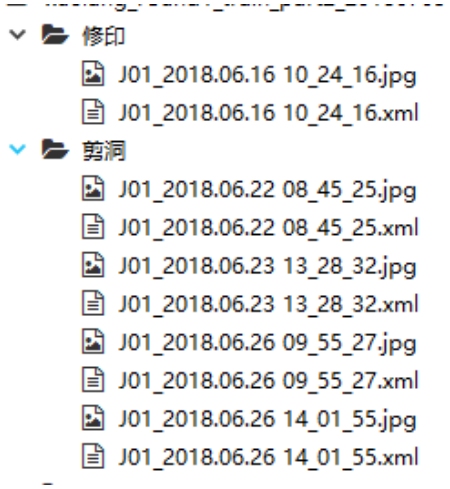
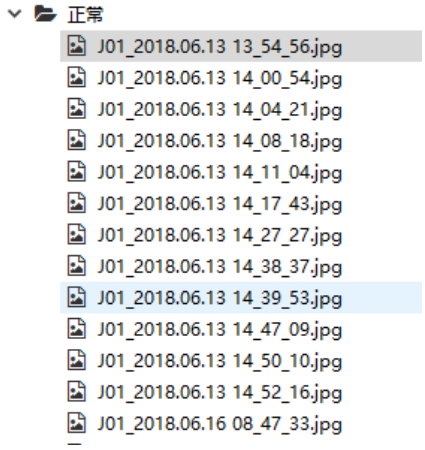


图 1.3 瑕疵类别为“修印”和瑕疵类别为“剪洞”的子文件夹下面图片与标签的对应关系示意



1.3 “正常”文件夹下面都是正常的布匹图片

（a） （b）

（c）　　（d）

图 1.4 a-c 为瑕疵图片，d 为正常图片

```
<annotation>
    <filename>J01_2018.06.22 08_45_25.jpg</filename>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>2560</width>
        <height>1920</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>剪洞</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>1205</xmin>
            <ymin>716</ymin>
            <xmax>1351</xmax>
            <ymax>792</ymax>
        </bndbox>
    </object>
</annotation>
```

1.5 瑕疵图片对应的 xml 文件示意

2.4 比赛规则

大赛分为初赛和复赛两个阶段。初赛阶段考察瑕疵检出能力，将提供全量瑕疵的图片数据用于训练不良布匹的识别模型。复赛阶段将聚焦部分重要的瑕疵，兼顾考察瑕疵检出和瑕疵分类能力，将提供部分瑕疵的图片数据用于训练瑕疵的分类模型。

在比赛中，使用多个模型的结果进行简单融合（如坐标位置/标签概率求平均等等），被认为是多模型策略。我们不鼓励过度堆砌模型和硬件来刷高比赛得分的行为。为倡导比赛算法的创新性和实用性，我们将在复赛阶段对“多模型策略”进行限制。分 2 个阶段进行说明：

a) 预测阶段

复赛要求预测阶段仅能使用不超过 2 个的模型，并且我们认定一个模型的大小（Param Mem）必须小于 600MB（即不超过 VGG19 的模型大小）。

注意 1：同一网络结构但参数不同也被认为是不同模型。

b) 预处理阶段

我们认定预处理阶段的产出结果仅限于：图像编辑（Image Augmentation）和辅助决策信息。预处理阶段不做模型大小和数量的限制，但预处理阶段的运算量和运算时间也会纳入决赛评委对方法实用性的考量，请合理设计。产出检验定位结果属于预处理，直接产出分类标签则属于预测阶段。

2.5 提交说明

参赛者提交 CSV 格式文件。

初赛第一行标记每一列的名称，一共 2 列，分别为图像文件名（filename）和概率（probability）。从第二行之后的每一行都记录一张图片包含瑕疵的概率。格式如下：

filename,probability

XXX.jpg,0.9

XXX.jpg,0.2

复赛第一行标记每一列的名称，一共 3 列，分别为图像文件名（filename），瑕疵名（defect）和概率（probability）。从第二行之后的每一行都记录一张图片包含瑕疵的概率。格式如下：

filename|defect,probability

XXX.jpg|defect_1,0.2

XXX.jpg|defect_2,0.1

defect code 和瑕疵的对应关系:

norm	defect_1	defect_2	defect_3	defect_4	defect_5	defect_6	defect_7	defect_8	defect_9	defect_10
正常	扎洞	毛斑	擦洞	毛洞	织稀	吊经	缺经	跳花	油/污渍	其他

表 1.1

注：“其他”代表剩余所有类型的瑕疵

初赛计算 AUC（Area Under Curve）

复赛计算 AUC*0.7+mAP*0.3，其中 mAP 为 Mean Average Precision。

第三章 基础概念介绍

在比赛中用到了一些基础的知识概念和理论以及程序的主要框架，在这里简单介绍一下。

3.1 keras 简介

TensorFlow 是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。节点（Nodes）在图中表示数学操作，图中的线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）。它灵活的架构让你可以在多种平台上展开计算，例如台式计算机中的一个或多个 CPU（或 GPU），服务器，移动设备等等。TensorFlow 最初由 Google 大脑小组（隶属于 Google 机器智能研究机构）的研究员和工程师们开发出来，用于机器学习和深度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。

Keras 则是一个高层神经网络 API，Keras 由纯 Python 编写而成并基 Tensorflow、Theano 以及 CNTK 后端。Keras 为支持快速实验而生，能够把 idea 迅速转换为结果，具有简易和快速的原型设计（keras 具有高度模块化，极简，和可扩充特性）支持 CNN 和 RNN，或二者的结合、无缝 CPU 和 GPU 切换等特点。

对于初次接触机器学习和深度神经网络的新手来说，快速上手以及傻瓜式使用方法无疑让 keras 和 tensorflow 的组合成了我们最好的选择。

关于这两者的介绍网上有很多的资料，不再赘述。这里主要介绍一些我在比赛中用到的一些内容以及刚开始不清楚后面才理解其魅力的一些技巧。

3.1.1 keras 的傻瓜式训练数据组织模块：`fit_generator`

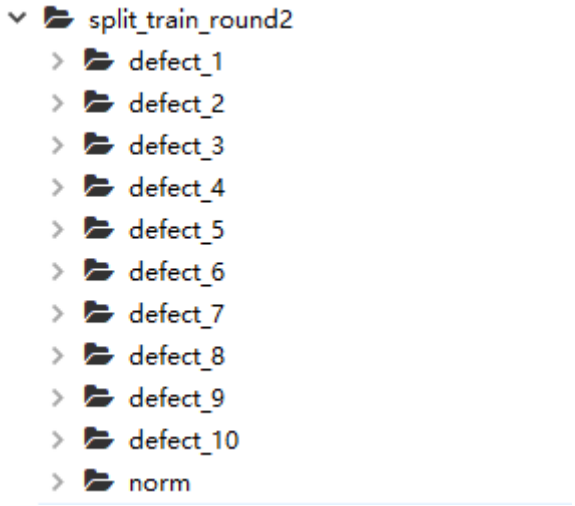


图 3.1 训练数据结构示意

如果训练数据中每一种类别名称的文件夹下都对应该类别的图片，如何能够不手动设置标签而使得模型能够进行训练？

```
trainGen320 = train_datagen.flow_from_directory(
        r'./data/split_train_round2',
        target_size=(512,512),
        batch_size=12,
        seed = seed,
        class_mode='categorical')
```

```
resultA = modelA.fit_generator(
        trainGen320,
        epochs=30,verbose=1,
        callbacks=callbacks_list,
        validation_data=valicGen320,
        class_weight='auto')
```

在 keras 中，准备好数据之后只需使用`.flow_from_directory()`来从我们的 jpgs 图片中直接产生数据和标签。

对于图 3.1 中如 defect_1 文件夹下的样本，可以直接生成“01000000000”这样的标签与其对应。使用者无需操心对应问题，可以直接进行训练。

3.1.2 数据增强

Keras 提供了一个很棒的数据增强接口，`ImageGenerator`。

主要功能：每次训练的时候，对拿到的数据进行随机变换处理，这使得拿到相同数据的不同的 epoch 随机之后喂给模型的数据是不一样的，增加随机性。

使用方法：

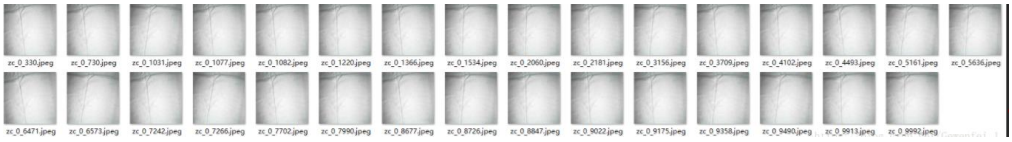
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.1,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   rotation_range = 90,
                                   fill_mode = 'constant',
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   channel_shift_range=10,
                                   cval = 0)
```

参数：（具体参数的含义请参考我的博客：https://blog.csdn.net/Gewenfei_1/article/details/81215355）

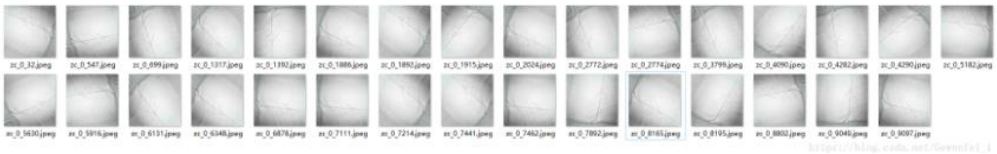
```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization = False,
    samplewise_std_normalization = False,
    zca_whitening = False,
    rotation_range = 0.,
    width_shift_range = 0.,
    height_shift_range = 0.,
    shear_range = 0.,
    zoom_range = 0.,
    channel_shift_range = 0.,
    fill_mode = 'nearest',
    cval = 0.0,
    horizontal_flip = False,
    vertical_flip = False,
    rescale = None,
    preprocessing_function = None,
    data_format = K.image_data_format(),
)
```

以 `rotation_range` 参数为例，整数，为数据提升时图片随机转动的角度。随机选择图片的角度，是一个 0~180 的度数，取值为 0~180。 比如给一个 40，那么后续图像随机转动的角度会在-40~40 之间。

比如给 rotation_range=10,后续得到的图片如图：



如果给 rotation_range=180，后续会得到：



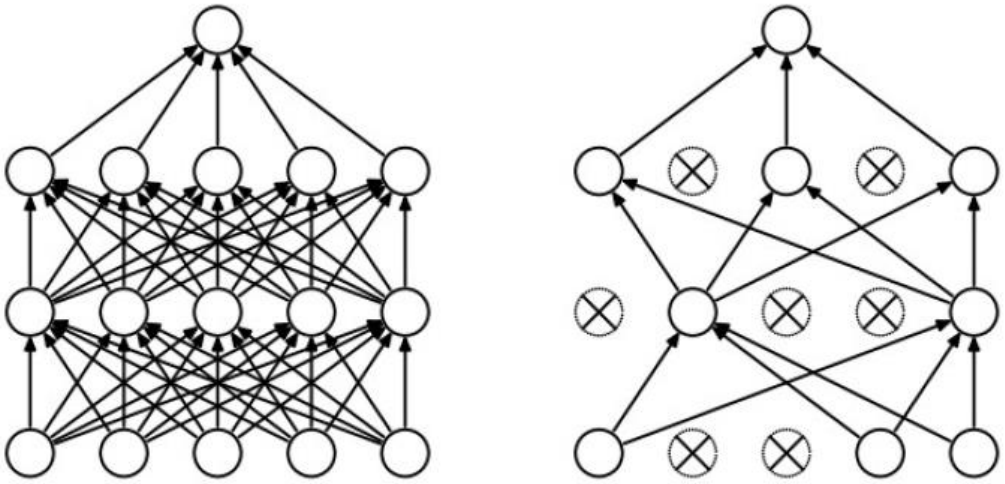
其余参数类似，此处提供一小段代码用于测试各个参数的效果：

```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

datagen = ImageDataGenerator(rotation_range=180)#####修改此处参数即可用于测试


img = load_img('zc_0.jpg')  # this is a PIL image
x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 150, 150)
# the .flow() command below generates batches of randomly transformed images
# and saves the results to the `preview/` directory
i = 0
for batch in datagen.flow(x, batch_size=1,
                          save_to_dir='lalala', save_prefix='zc', save_format='jpeg'):
    i += 1
        if i > 30:

            break  # otherwise the generator would loop indefinitely
```

3.1.3 keras 中设置 Dropout

Dropout 通过防止一层看到两次完全一样的模式来防止过拟合，相当于也是一种数据提升的方法。（你可以说 dropout 和数据提升都在随机扰乱数据的相关性）

dropout 是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。注意是暂时，对于随机梯度下降来说，由于是随机丢弃，故而每一个 mini-batch 都在训练不同的网络。



(a) Standard Neural Net　　(b) After applying dropout.

当前 Dropout 被大量利用于全连接网络，而且一般人为设置为 0.5 或者 0.3，而在卷积隐藏层由于卷积自身的稀疏化以及稀疏化的 ReLu 函数的大量使用等原因，Dropout 策略在卷积隐藏层中使用较少。需要根据具体的网路，具体的应用领域进行尝试。

```
model.add(Flatten())  # this converts our
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

3.1.4 keras 自定义损失函数

损失函数（或称目标函数、优化评分函数）是编译模型时所需的主要参数之一：

```
model.compile(loss='mean_squared_error', optimizer='sgd')
```

```
from keras import losses

model.compile(loss=losses.mean_squared_error, optimizer='sgd')
```

Keras 内部提供了很多定义好的损失函数，可以根据具体是回归问题还是分类问题选择使用，回归问题一般使用均方差损失函数，分类问题一般选用交叉熵损失函数。均方差损失函数一般有平均平方误差，平均绝对误差，平均绝对百分率误差等，交叉熵损失函数一般有绝对交叉熵，稀疏绝对交叉熵等，具体源码可参考：https://github.com/keras-team/keras/blob/master/keras/losses.py，交叉熵损失函数公式如下：

$$H(p,q) = -\sum_{x}(p(x)\log q(x) + (1-p(x))\log(1-q(x)))$$

在我们的问题中，由于类别不平衡，在通过数据预处理，滑动窗裁剪增强数据后，就会改变数据集的分布，同时在评价标准上，正常类别的权重是大于剩下的 10 类瑕疵的权重，如果直接调用交叉熵损失函数，在数据增强后类别平衡的条件下，则其每个类别的权重相等，这种就会对收敛的目标产生很大的影响，我们综合考虑了 11 类原始样本的分布，设计了新的损失函数如下：

```python
def new_loss(y_true,y_pred):

    y_pred /= tf.reduce_sum(y_pred,-1, True)
    y_pred=tf.clip_by_value(y_pred,1e-10,1.0-1e-10)
    a_0=-(y_true[:,0] * tf.log(y_pred[:,0]) + (1-y_true[:,0]) * tf.log(1-y_pred[:,0]))
    a_1=-(y_true[:,1] * tf.log(y_pred[:,1]) + (1-y_true[:,1]) * tf.log(1-y_pred[:,1]))
    a_2=-(y_true[:,2] * tf.log(y_pred[:,2]) + (1-y_true[:,2]) * tf.log(1-y_pred[:,2]))
    a_3=-(y_true[:,3] * tf.log(y_pred[:,3]) + (1-y_true[:,3]) * tf.log(1-y_pred[:,3]))
    a_4=-(y_true[:,4] * tf.log(y_pred[:,4]) + (1-y_true[:,4]) * tf.log(1-y_pred[:,4]))
    a_5=-(y_true[:,5] * tf.log(y_pred[:,5]) + (1-y_true[:,5]) * tf.log(1-y_pred[:,5]))
    a_6=-(y_true[:,6] * tf.log(y_pred[:,6]) + (1-y_true[:,6]) * tf.log(1-y_pred[:,6]))
    a_7=-(y_true[:,7] * tf.log(y_pred[:,7]) + (1-y_true[:,7]) * tf.log(1-y_pred[:,7]))
    a_8=-(y_true[:,8] * tf.log(y_pred[:,8]) + (1-y_true[:,8]) * tf.log(1-y_pred[:,8]))
    a_9=-(y_true[:,9] * tf.log(y_pred[:,9]) + (1-y_true[:,9]) * tf.log(1-y_pred[:,9]))
    a_10=-(y_true[:,10] * tf.log(y_pred[:,10]) + (1-y_true[:,10]) * tf.log(1-y_pred[:,10]))

    haha= (1163/3331*a_0 + 154/3331*a_1 + 142/3331*a_2 + 313/3331*a_3 + 179/3331*a_4 + 195/3331*a_5 +
    339/3331*a_6 +163/3331*a_7 + 210/3331*a_8 + 141/3331*a_9 + 332/3331*a_10)
    newloss=tf.reduce_mean(haha)
    return newloss
```

最终直接在 model.complie 直接调用即可：

```python
model.compile(loss=new_loss,optimizer=adadelta,metrics=['accuracy'])
```

3.1.5 keras 调用主流的网络模型及其参数

Keras 的应用模块（keras.applications）提供了带有预训练权值的深度学习模型，这些模型可以用来进行预测、特征提取和微调（fine-tuning），当你初始化一个预训练模型时，会自动下载权值到 `~/.keras/models/` 目录下。

模型概览

模型	大小	Top-1 准确率	Top-5 准确率	参数数量	深度
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

Top-1 准确率和 Top-5 准确率都是在 ImageNet 验证集上的结果。

所有的这些模型（除了 Xception 和 MobileNet 外）都兼容 Theano 和 Tensorflow，并会自动按照位于 ~/.keras/keras.json 的配置文件中设置的图像数据格式来构建模型。举个例子，如果你设置 image_data_format=channels_last，则加载的模型将按照 TensorFlow 的维度顺序来构造，即“高度-宽度-深度”（Height-Width-Depth）的顺序。

Xception 模型仅适用于 TensorFlow，因为它依赖于 SeparableConvolution 层。MobileNet 模型仅适用于 TensorFlow，因为它依赖于 DepthwiseConvolution 层。

以上模型可以直接从 keras 的应用中导入，在比赛中，我们删除了最后的全连接层，加了全局平均池化和 2 层全连接分类器，为了防止过拟合，使用了 L2 正则化和 Dropout，同时使用了 imagenet 的参数。以 Xception 为例：

```
base_model = Xception(include_top=False,
                      weights='imagenet',
                      input_shape=(height, width, 3))
# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu',kernel_regularizer=keras.regularizers.l2(0.01))(x)
x = Dropout(0.5)(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(11, activation='softmax')(x)
# this is the model we will train
model = Model(input=base_model.input, output=predictions)
```

3.1.6 keras 优化器选择

优化器(optimizer)是编译 Keras 模型的所需的两个参数之一：

```
from keras import optimizers

model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('tanh'))
model.add(Activation('softmax'))

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

你可以先实例化一个优化器对象，然后将它传入 model.compile()，像

上述示例中一样， 或者你可以通过名称来调用优化器。在后一种情况下，将使用优化器的默认参数。

Keras 提供了足够多的优化器可供选择，你可根据具体问题选择合适的优化器，也可更改优化器的参数。比赛中我们采用的优化器如下：

```
adadelta=Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=1e-6)
```

3.1.7 keras ModelCheckpoint 实现断点续训功能

回调函数是一个函数的合集，会在训练的阶段中所使用。你可以使用回调函数来查看训练模型的内在状态和统计。你可以传递一个列表的回调函数（作为 callbacks 关键字参数）到 Sequential 或 Model 类型的 .fit() 方法。在训练时，相应的回调函数的方法就会被在各自的阶段被调用。Keras 提供了丰富的回调函数，见 https://keras.io/zh/callbacks/

在比赛中，我们采用的回调函数主要是 ModelCheckpoint 和 ReduceLROnPlateau 和一个自定义的 history 用于训练完毕画出训练集和验证集的 loss 和 acc 曲线，为误差分析提供帮助

```
checkpointer_1=ModelCheckpoint(filepath=Xception_models/model.h5, monitor='val_loss', verbose=0,
    save_best_only=True, save_weights_only=False, mode='auto', period=1)
checkpointer=ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=0, mode='auto',
    epsilon=0.0001, cooldown=0, min_lr=0)
#model.fit(X_train, Y_train, epochs = 1000, batch_size = 64,callbacks=[checkpointer,checkpointer_1,hi
model.fit_generator(
        train_generator,
        epochs=80,
        verbose=1,
        validation_data=validation_generator,
        class_weight='auto',callbacks=[checkpointer,checkpointer_1,history],workers=4,shuffle=True)
#绘制acc-loss曲线
history.loss_plot('epoch')
```

ModelCheckpoint 主要用于在每个训练期之后保存模型，monitor 设置的为需要在每轮训练完毕后检测的量，在比赛中，我们设置的为 val_loss，即如果下一代验证集的 loss 小于上一代，则会保存新的一代模型。其中 filepath 可以包括命名格式选项，可以由 epoch 的值和 logs 的键（由 on_epoch_end 参数传递）来填充。

例如：如果 filepath 是 weights.{epoch:02d}-{val_loss:.2f}.hdf5，那么模型被保存的的文件名就会有训练轮数和验证损失。用此种方式会保存多个模型，有利于当遭遇停电时，模型后续可重新被加载用于训练。

ReduceLROnPlateau 用于当标准评估已经停止时，降低学习速率，当学习停止时，模型总是会受益于降低 2-10 倍的学习速率。 这个回调函数监测一个数据并且当这个数据在一定「有耐心」的训练轮之后还没有进步， 那么学习速率就会被降低。

3.1.8 keras 可视化训练过程

在比赛中，我们采用自己定义的类，在 callback 中调用，最后实现训练过程每轮训练集和验证集的 acc，loss 的可视化。

```
class LossHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.losses = {'batch':[], 'epoch':[]}
        self.accuracy = {'batch':[], 'epoch':[]}
        self.val_loss = {'batch':[], 'epoch':[]}
        self.val_acc = {'batch':[], 'epoch':[]}

    def on_batch_end(self, batch, logs={}):
        self.losses['batch'].append(logs.get('loss'))
        self.accuracy['batch'].append(logs.get('acc'))
        self.val_loss['batch'].append(logs.get('val_loss'))
        self.val_acc['batch'].append(logs.get('val_acc'))

    def on_epoch_end(self, batch, logs={}):
        self.losses['epoch'].append(logs.get('loss'))
        self.accuracy['epoch'].append(logs.get('acc'))
        self.val_loss['epoch'].append(logs.get('val_loss'))
        self.val_acc['epoch'].append(logs.get('val_acc'))

    def loss_plot(self, loss_type):
        iters = range(len(self.losses[loss_type]))
        plt.figure()
        # acc
        plt.plot(iters, self.accuracy[loss_type], 'r', label='train acc')
        # loss
        plt.plot(iters, self.losses[loss_type], 'g', label='train loss')
        if loss_type == 'epoch':
            # val_acc
            plt.plot(iters, self.val_acc[loss_type], 'b', label='val acc')
            # val_loss
            plt.plot(iters, self.val_loss[loss_type], 'k', label='val loss')
        plt.grid(True)
        plt.xlabel(loss_type)
        plt.ylabel('acc-loss')
        plt.legend(loc="upper right")
        plt.show()
```

```
model.fit_generator(
        train_generator,
        epochs=80,
        verbose=1,
        validation_data=validation_generator,
        class_weight='auto',callbacks=[checkpointer,checkpointer_1,history],workers=4,shuffle=True)
#绘制acc-loss曲线
history.loss_plot('epoch')
```

通过这种方式就可以简单方便的可视化训练过程。

第四章 主要思路

4.1 初赛

对于初赛，题目要求只需要将所有的图片分类为：正常 or 瑕疵就行，刚开始的思路就很简单，直接把正常图片放在一个文件夹，瑕疵图片放在另一个文件夹，（首先对所有图片进行 resize，原图大小为 2560*1920，GPU 也跑不动，因此 resize 到 320*320）然后选一个模型，用 keras 训练走了一遍……然后发现效果巨烂（acc 大概 0.8 左右）。

然后分析了一下数据集，发现了很多问题：

1. 数据集不平衡，正常图片的数量大约是瑕疵图片的两倍；
2. 瑕疵在整张图片中非常小，resize 更是损失了很多信息，但是不经过 resize 又不能训练；
3. 模型怎么选合适？优化器怎么选合适？……

经过试验和摸索，选择了优化数据集的方式如下：

（1） 原图 resize 到 499*499 大小；

（2） 对于有瑕疵的图片，对瑕疵部位进行多次采样，将图片中与瑕疵所在的标注框的交并比大于一定阈值的图像区域 resize 到 499*499 大小，也作为瑕疵样本参与训练，如图 3.1 所示；

（3） 分别测试 VGG 系列模型以及 Resnet、Inception 等多种模型，选出最合适的模型构架；

（4） 尝试不同的优化器、加 dropout、GAP 层、BN 等调参过程优化。

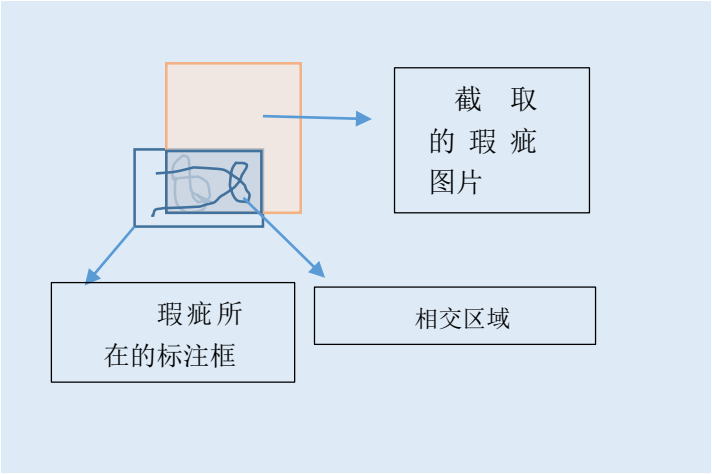


图 4.1 从瑕疵图片中采样示意图

其中第 1,2 步处理的依据为：对于有些瑕疵，其具有全局特征，比如边扎洞等，需要用整张图像进行训练；但是对于毛洞等局部小瑕疵，resize 并不能很好地保留特征，因此需要对瑕疵部位进行重点采样并训练，确保模型可以识别瑕疵特征。同时，对瑕疵的采样过程也充实了瑕疵样本，这样可以让瑕疵样本和正常样本数量达到平衡。

最后选用 VGG-13 模型为基础，在最终的参数调整之后（调参就是不断地尝试，当然也有一些经验以及不同的数据集会有不同的调整策略，但是还是具体情况具体实施，这里的经验不一定适用，就不总结了），准确率达 0.91。

对于很多组都是用 3-4 个模型在融合的情况（初赛排第九的一个队伍三个 0.91 的模型融合之后就是 0.94 左右的成绩了，模型融合也只是简单的将几个模型对同一个样本的预测结果加权平均，不过好处就是可以评测好几个模型的预测结果，一定程度上提高准确度），我们这样的单模型训练效果也算是不错的了。

4.2 复赛

虽然 0.91 的准确率让我们顺利进了复赛，但是复赛的难度也有所加大。

复赛要求将瑕疵分为 10 类，加上正常图片，需要将所有图片进行 11 分

类，但是数据集并没有明显扩充（只是将初赛的两个测试集给了 label，可以用于训练集的扩充，但是对于 11 分类问题，有的类别的样本依然是非常少，甚至只有几十张图片）。初赛时候的简单思路显然已经行不通，首要面对的问题就是如何扩充数据集。

受到初赛一个队伍的处理思路（源代码的 github 链接：https://github.com/lightfate/XueLang-YOLOhasst）的启发，我们开始采用滑动窗的方式处理图片，最后的处理流程为：

（1） 将初赛的测试数据转换为训练数据所需的格式，即同名瑕疵图片和 xml 标签文件在同一路径下，便于后续一起处理；

（2） 将所有图片分为 11 类，并保存在'./data/raw'路径下的 11 个子文件夹中；

（3） 将所有图片进行滑动窗分割，正常图片滑动步长为 256，滑动窗大小为 512*512；而对于瑕疵样本，由于瑕疵所在的区域比较小，如果采用全局滑动窗裁剪，得到的图片里面有大部分图片不包含瑕疵，因此在裁剪结果中仅保留与瑕疵所在的标注框交并比大于某一个阈值的滑动窗作为得到的瑕疵图；由于瑕疵图片的样本数量不平衡，因此每一类瑕疵图片的滑动窗的滑动步长可以有所不同，以得到理想数量的瑕疵训练样本；同时由于不同的瑕疵的特点，对于小瑕疵可以设置交并比大一些的图进行保留，大瑕疵可以设置交并比小一些的图进行保留，以保留瑕疵的重要特征。

（4） 滑动窗裁剪之后得到的正常样本数量会远远超出瑕疵训练样本的数量，以一定比例随机剔除正常样本，使得每一类样本的数量相对平衡；

（5） 以一定的比例划分训练集合验证集；

（6） 利用 keras 的数据 ImageDataGenerator 对训练数据进行数据增强；

（7） 使用 keras 的现有模型 InceptionResNetV2 及其 imagenet 参数进行模型的初始化，修改全连接层，改为 11 分类输出，自定义 11 分类损失函数，并调整其他网络参数，进行训练。

（8） 由于训练数据使用的是原图裁剪得到的 512*512 小图，因此在预测的时候也要使用 512 的滑动窗对待预测图片进行滑动裁剪，滑动步长为 256，裁剪得到((2560-512+256)/256)* ((1920-512+256)/256)=63 张小图，每一张图都得到一个预测结果。最后，判断 63 个预测结果中有无瑕疵类的预测结果，如果没有，则该图为正常图片；如果有，则判断哪一类瑕疵的预测概率值最大，取最大值的瑕疵概率即为该图的瑕疵类别。

以上思路经过调参测试发现模型对瑕疵的分类准确率较高，但是对正常图片的识别率较低，容易将正常图片中的折叠、花纹等识别为瑕疵。考虑到最后的得分计算公式：

AUC*0.7+mAP*0.3，其中 mAP 为 Mean Average Precision

其中 AUC 是二分类模型的重要评估指标，也就是说，最后成绩的 70%来源于模型能不能将正常样本和瑕疵样本区分清楚，而瑕疵内部的分类权重并不算很高。因此，最后的策略为：

（1） 选用一个较好的二分类模型，使得 2 分类的 AUC 达到 0.93 以上；

（2） 对于待遇测图片，先用二分类进行预测，如果二分类判断其为正常图片，则不再进行瑕疵预测；如果二分类判断其为瑕疵图片，将该图片送给 11 分类模型，预测得到该瑕疵的类别。

最终，模型得到 0.73 左右的评分，位于复赛 20 左右。

第五章 心得体会

经过此次比赛，总结了一些参加此类比赛的经验：

（1） 关注开源社区，多看看相关问题别人的解决思路，会很有启发，不要自己一上来就闷着头做，很容易走进死胡同；

（2） 要打好基础，机器学习以及深度神经网络的基础知识要扎实，虽然现在的 keras 等平台非常的容易上手，但是真正的好的思路都是要从根源去想的，有扎实的基础不管是开阔思路还是对编程验证都有很大的帮助；

（3） 要分析数据特点，做好数据预处理工作，将数据特点“最大化”地展现给模型，不能将全部问题抛给模型；

（4） 调参是一个需要不断尝试的过程，要有足够的耐心。

首次参加此类比赛，虽不是为了名次，但也算留有遗憾，不过收获的知识和经验也已经值得一个月有余的精力投入了，整理好自己的收获，以期更好地征程！

附录 程序使用的源代码

1 源程序

程序使用的所有源代码都将被放在我的 github 上，链接：https://github.com/WenfeiGe/-AI----

2 AUC

（参考博客：https://www.cnblogs.com/van19/p/5494908.html）

从二分类说起，假设我们的样本全集里，所有样本的真实标签（label）为 0 或 1，其中 1 表示正样本，0 表示负样本，如果我们有一个分类模型，利用它对样本进行了标注，那边我们可以得到下面的划分

		truth	
		1	0
predictor	1	TP	FP
	0	FN	TN

TP(true positive)：表示正确的肯定

TN(true negative)：表示正确的否定

FP(false positive)：表示错误的肯定

FN (false negative)：表示错误的否定

简记：第一个字母表示最终结果 true or false（正确或错误），第二个字母表示预测标签 positive or negative（肯定或否定），从左往右读。例如 FP，表示错误地肯定，即预测是 1，而真实是 0 个样本个数。

由此，我们能得到 TPR 和 FPR 的定义

TPR(true positive rate)：TPR = TP / T = TP / (TP + FN)，表示预测为正的正确结果 TP 在所有正样本 T 中的占比，显然 TPR 越大，模型的预估能力更好。

FPR(false positive rate)：FPR = FP / F = FP / (FP + TN) ，表示预测为正的错误结果 FP 在所有负样本 F 中的占比，显然 FPR 越大，模型的预估能力越差。

ROC 曲线（receiver operating characteristic curve），由 FPR 为 X 轴坐标，TPR 为 Y 轴坐标的曲线。下面介绍如何绘制 ROC 曲线

假设模型 M 对样本进行标注，当预测值大于某个阈值 r 时，我们用下面的函数来预测最后结果

$$f(x)=\{1\ \ x\geqslant r0\ \ x<rf(x)=\{1\ \ x\geqslant r0\ \ x<r$$

然后，对于每一个 r，模型对所有样本标注一遍，统计得到所有的 TP 和 FP，就可以计算(fpr,tpr)，对所有的点在 ROC 图上进行标注就可以得到 ROC 曲线，而本文的主题 AUC 就是指

ROC 曲线下方的面积（Area under the Curve of ROC）。

下面我们来看一个实际的例子

下面表中有 10 个样本，其中 5 个正样本，5 个负样本，即 T=5，F=5。

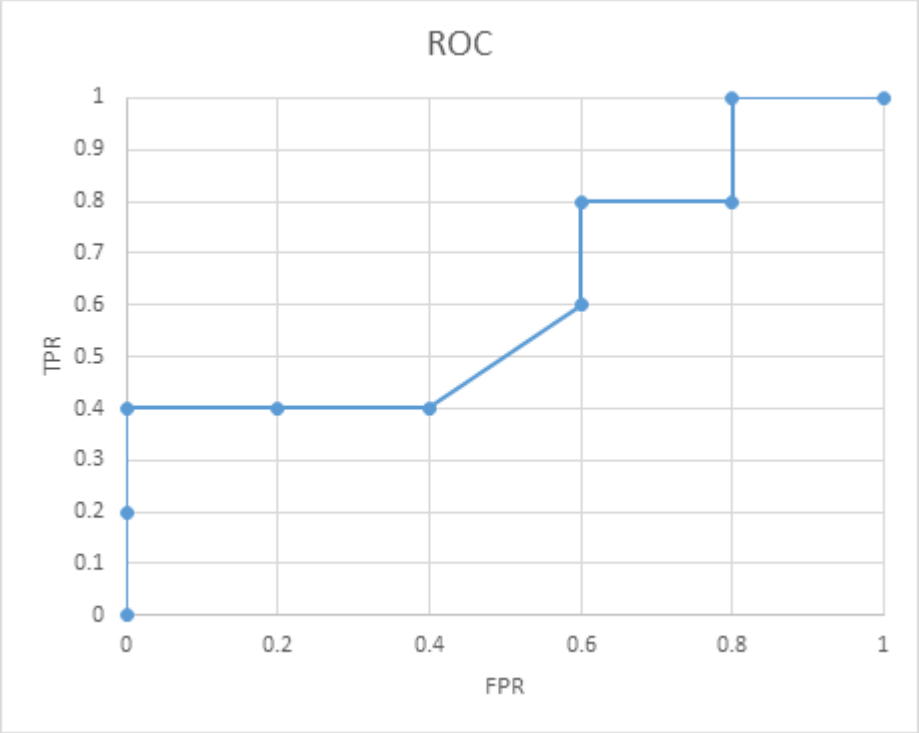
score	label
0.15	1
0.12	1
0.11	0
0.1	0

0.04	0
0.04	1
0.03	1
0.02	0
0.012	1
0.01	0

我们让 r 从+∞逐渐较小至-∞，对每一个区间的 r 计算点(fpr,tpr)，如下图

r	fp	tp	fpr	tpr
(0.15,+∞)	0	0	0	0
(0.12,0.15]	0	1	0	0.2
(0.11,0.12]	0	2	0	0.4
(0.1,0.11]	1	2	0.2	0.4
(0.04,0.1]	2	2	0.4	0.4
(0.03,0.04]	3	3	0.6	0.6
(0.02,0.03]	3	4	0.6	0.8
(0.012,0.02]	4	4	0.8	0.8
(0.01,0.012]	4	5	0.8	1
(-∞,0.01)	5	5	1	1

由此描绘 ROC 曲线



计算 auc

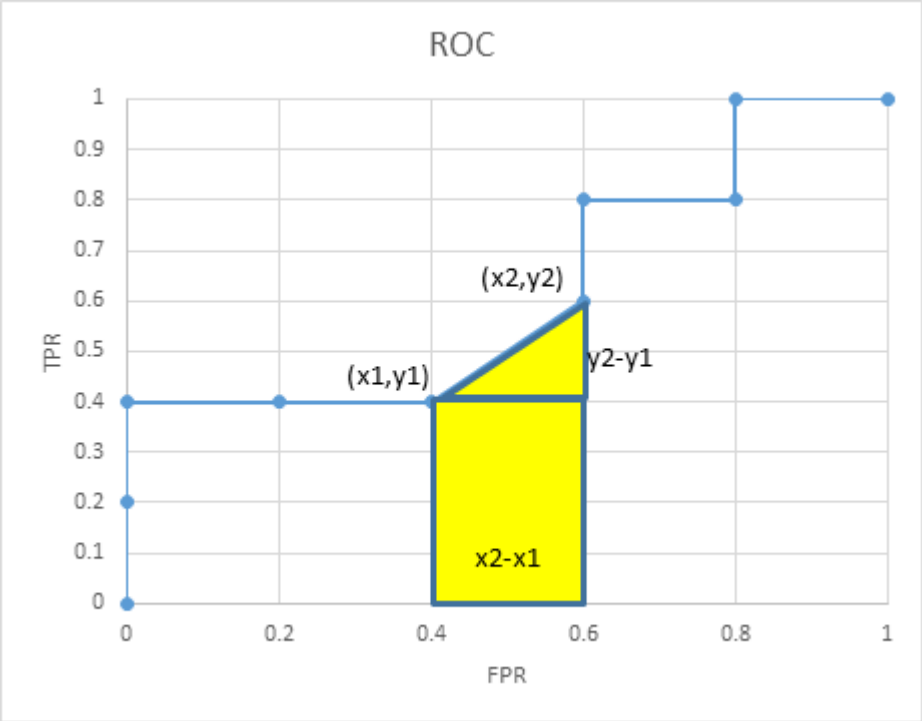
auc 即为 ROC 曲线下方的面积，按梯形法（连续两点构成的梯形）计算面积得到上面例子的 auc 为 0.62。

AUC 表示，随机抽取一个正样本和一个负样本，分类器正确给出正样本的 score 高于负样本的概率。在所有标注样本中，正样本共 T 个，负样本共 F 个，如果随机抽一个正样本和负样本，共有 T*F 个 pair，其中如果有 s 个 pair 的正负样本 pair 满足于正样本的 score 高于负样本（权重为 1），v 个 pair 正负样本 score 相同（权重为 0.5），则 auc 就等于(s+0.5v)/(T*F)。本文的例子中，s=15，v=1，则 auc=(15+1*0.5)/(5*5)=0.62，和梯形法计算面积的结果是一致。整体上对于某一个正负样本 pair：<i,j>的权重 Wij 计算如下

$$W(i,j)=\begin{cases}1 & s_i>s_j \\ 0.5 & s_i=s_j \\ 0 & s_i<s_j\end{cases}$$

si 和 sj 分别表示正样本 i 和负样本 j 的 score。

下面我们对这个进行证明



我们用梯形法，求摸一个梯形的面积时，如上图

图中黄色填充区域的面积是

$S=(x_2-x_1)*(y_2+y_1)/2=(x_2-x_1)*y_1+0.5*(x_2-x_1)*(y_2-y_1)$S=(x2−x1)∗(y2+y1)/2=(x2−x1)∗y1+0.5∗(x2−x1)∗(y2−y1)

对 S 的分子分母同时乘以 T*F，由于 x*F=FP，y*T=TP，所以有

$S=(FP_2-FP_1)*TP_1+0.5*(FP_2-FP_1)*(TP_2-TP_1)T*F=\Delta FP*TP_1+0.5*\Delta FP*\Delta TPT*F$S=(FP2−FP1)∗TP1+0.5∗(FP2−FP1)∗(TP2−TP1)T∗F=ΔFP∗TP1+0.5∗ΔFP∗ΔTPT∗F

其中 ΔFP=FP2-FP1>=0，表示随着 r 降低，本次计算 roc 点时新增的负样本数目； ΔTP=TP2-TP1>=0，表示随着 r 的降低，本次计算 roc 点时新增的正样本数目。

由上式可以看出，分母 T*F 表示所有正负样本的 pair 总数。

分子第一部分 ΔFP*TP1 表示新增正负样本 pair 数目，因为前一个的正样本数 TP1 和本次新增的负样本数 ΔFP 组成的 pair 都满足正样本的 score 高于负样本，这部分 pair 权重为 1

分子第二部分 ΔFP*ΔTP 表示此次新增的同 score 的正负样本 pair 数目，这部分权重为 0.5。

由此可知每一个梯形面积都表示此时正负样本 pair 满足正样本 score 大于等于负样本的加权计数值占全体正负样本 pair 的占比。从 S0 累计到最后一个 Sn 整体表示样本整体满足条件的正负样本的占比，此时等于 AUC 的面积计算值。

到此为止，我们给出了计算 auc 的两种方法

1 根据定义，由梯形法计算 ROC 曲线下的面积，求 auc

2 遍历全部样本，对正负 pair 的数目计数，求 auc

在单机上，这两种算法的复杂度比较高，我们对方法 2 稍作改进，得到方法 3

3 我们可以将所有样本按升序排好后，通过样本的序号（或者说排序位置，从 1 开始）rank 来计算，我们先给出公式

AUC=((所有的正样本 rank 相加)-T*(T+1)/2)/(T*F)

对相等 score 的样本，需要赋予相同的 rank，即把所有这些 score 相等的样本的 rank 和取平均。

本文的例子（表 1），按升序排好后如下表

rank	score	label
1	0.01	0
2	0.012	1

3	0.02	0
4	0.03	1
5	0.04	0
6	0.04	1
7	0.1	0
8	0.11	0
9	0.12	1
10	0.15	1

所有的正样本 rank 相加=10+9+(5+6)/2+4+2=30.5，注意 5 和 6 由于 score 相同，需要均分 rank 和。

auc=(30.5-5*6/2)/(5*5)=0.62

下面我们证明一下这个计算公式和方法 2 计算是一致的

首先是分母部分，T*F 表示所有正负样本 pair 数，对于分子部分

设每个正样本 i 的 rank 值为 Ri，其 score 为 si，

若 Ri 的 score 唯一，表示 si 大于 Ri-1 个样本，样本 i 和这 Ri-1 个样本组成的 pair 权值为 1，所有此类正样本的 rank 之和表示和之前的所有 Ri 个样本组成的 pair 数（包括和自己的 pair 以及之前的正样本 pair 也算在内）

如 Ri 的 score 不唯一，不妨设此时有 p 个正样本和 q 个负样本 score 和 i 相同，那么此时有 p*q 个 pair 权值为 0.5。

假设这连续 p+q 个样本中的第一个 rank 为 t，则第 p+q 个样本的 rank 为 t+p+q-1，根据方法 3 所述，这 p+q 个样本的 rank 值用平均 rank 来代替，为(t+t+p+q-1)/2，则 p 个正样本的 ranker 和为(t+t+p+q-1)*p*0.5=p*(t+t+p-1)/2+0.5*p*q。

我们看到第二项就是 p*q 个正负样本 pair 的加权和，而第一项是 rank 从 r,r+1,…,r+p-1 这 p 个正样本的 rank 和,它表示这 p 个正样本和 r 前面的样本组成的 pair

数（这 p 个样本和自己的 pair 和之前的正样本 pair 也算在内）。

从 1 和 2 分析看，所有正样本 rank 和会把正样本自己（共 T 个 pair）和自己之前的所有正样本组成的 pair（共 T*(T-1)/2 个 pair）都计算一遍，并且权重为 1，因此最后要去掉这个计数，这个计数就是 T*(T+1)/2 个 pair，因此方法 3 的公式的分子算出来是正确正负样本 pair 的加权和。

总结：

1 auc 只反应模型对正负样本排序能力强弱，对 score 的大小和精度没有要求

2 auc 越高模型的排序能力越强，理论上，当模型把所有正样本排在负样本之前时，auc 为 1.0，是理论最大值。

3 mAP

Precision和Recall的计算公式分别为：

$$Precision=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

$$Recall=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

Precision-recall 曲线（PR 曲线）与 ROC 曲线的区别是横轴和纵轴不同，PR 曲线的横轴 Recall 也就是 TPR，反映了分类器对正例的覆盖能力。而纵轴 Precision 的分母是识别为正例的数目，而不是实际正例数目。Precision 反映了分类器预测正例的准确程度。那么，Precision-recall 曲线反映了分类器对正例的识别准确程度和对正例的覆盖能力之间的权衡。对于随机分类器而言，其 Precision 固定的等于样本中正例的比例，不随 recall 的变化而变化。

与 AUC 相似，AP 就是 PR 曲线与 X 轴围成的图形面积，

对于连续的 PR 曲线，有：

AP=∫10PRdrAP=∫01PRdr

对于离散的 PR 曲线，有：

$$AP=\sum k=1nP(k)\Delta r(k)AP=\sum k=1nP(k)\Delta r(k)$$

此外，对于网页排序场景，还需要引入 MAP（Mean Average Precision），MAP 是所有查询结果排序的 AP 平均。

公式表示为：

$$MAP=\sum Qq=1AP(q)QMAP=\sum q=1QAP(q)Q$$

其中，Q 为查询的总次数。