《机器学习》结课报告：活体检测

高博钰: 21838031　种琳: 11812073

2020 年 12 月 14 日

摘要

活体检测任务属于对图像进行二分类，整体网络框架基于卷积神经网络。由于总数据量巨大且单张图像尺寸也较大，考虑到计算资源有限，本报告将研究重点聚焦于如何在抽样获得的小样本数据集下进行数据处理与网络框架设计，尽可能提高训练网络对测试数据集的预测精度。本报告主要内容包括：数据集重采样、数据预处理与数据增强、网络框架设计、模型调优策略这四个方面。另外，本报告在最后一节列举我们小组认为可进一步提高模型精度的思路，但受限于时间和计算资源有限，这些思路并未付之实践。

关键字：数据重采样；数据集预处理；数据增强；网络框架设计；模型调优策略

1 前言

人脸活体检测（Face Liveness Detection，FLD）是一种典型的图像逻辑回归问题（二分类）：我们需要用算法对真实人脸图像与照片人脸图像进行区分。现实项目中。人脸活体检测常使用时间序列图像数据进行综合分析，即要求镜头前真人人脸按要求完成一系列固定动作；但本项目规定数据单位是单张图像，不能使用时间序列信息。基于此，我们需要充分挖掘“真人拍照数据”与“照片拍照数据”的区别，对用算法去放大这些区别。首先，我们先对比观察这两种数据来判定该项目的可行性。

如图 1 所示：两种数据最大的区别在于“光线”。左 1 为真人拍照数据：光线更加自然，就像是我们用裸眼看到的景象一样。其余照片拍照数据明显光线都不够自然：背景光源被锐化（左 2），要么被虚化（左 3、左 4）。我们认为这种“虚和实”在照片拍照数据中是必然存在的，因此我们用手机拍照中，常会点击屏幕中的某块区域让镜头进行“聚焦”，且这种“聚焦”常常在人脸部位（左 2）！因此，没有被聚焦的背景区域，常常会被虚化或锐化（取决于背景光的强弱）。为什么我们会推断这些照片数据一定是在拍照过程中有“聚焦”操作？因为普通手机的防抖性能很差，人手的不可靠抖动必定会导致拍照模糊，因此我们拍照时常会进行这种“默认”的取舍：让镜头聚焦于部分关键区域，一定程度舍弃背景区域的清晰度。这种“虚与实”的光效效果不同反应在图像中就是区域内的像素值不同。

综上分析，我们推断“真人拍照数据”与“照片拍照数据”之间一定存在肉眼（人的经验）

可辨别的信息，即这项人脸活体检测任务是具有可行性的。卷积神经网络（Convolutional Neural Network，CNN）的本质一个复杂的“滤波器”，它利用一个个卷积核与图像进行卷积计算来完成特征提取（滤波）；这意味着卷积核越小，滤波操作就越细腻，关注到的特征就越细节。因此，本文将以卷积神经网络为基础搭建整个网络框架。

图 1. 真人拍照图有照片拍照图的对比（左 1 为真人拍照数据；其余均为照片拍照数据）

2 数据重采样

本项目的数据量庞大，为保证我们的计算机能够正常完成网络训练，对数据集必须要有筛选。注意到一点：原始数据为视频文件，但由于不让使用时域信息，故将视频文件以帧为单位进行分解。这意味着：来自同一视频文件的分帧图像数据，它们之间的相关性是非常高的！甚至彼此之间几乎没有差别，即不能补充大量有效信息。因此在小样本条件下，本文的数据集重采样策略如下：

本项目原始数据命令规律是：文件夹名称以“_1”结尾的都是“真人拍照数据”（以下统称为“真数据”），标签为 1；以其他结尾的都是“照片拍照数据”（以下统称为“假数据”），标签为 -1。训练数据集（trainset）中一共 1800 个文件夹，真假数据文件夹比例为 $1:4$。此外，每个文件夹内的图像个数也不是一定的。为保证重采样后的真假数据个数比例为 $1:1$，本文从每个“真数据文件夹”中随机采 4 张图片，从每个“假数据文件夹”中随机采 1 张图片。最终从训练数据集中抽取 2880 张图片：其中真、假数据各 1440 张。

我们知道：提升模型对未知数据的预测精度（泛化能力）的最好办法就是增加其训练数据集，让模型见更多的数据。基于此思想，我们把原本作为训练过程中检验模型精度的“验证集”也融入到训练数据集中，具体操作如下：原始验证数据集（devset）中一共 1350 个文件夹，真假数据文件夹比例也为 $1:4$。我们最终决定先将原始训练数据集（trainset）与原始验证数据集（devset）合并在一起（文件夹不会重名）形成一个原始总数据集，一共有 3150 个文件夹（真假数据文件夹比例依旧是 $1:4$）；然后按照上面的思路，从每个“真数据文件夹”中随机抽取 4 张图片，从每个“假数据文件夹“中随机抽取 1 张图片。最终原始数据集中有 5040 张图片，真假数据各一半。在后面模型训练前，对这个原始数据集再按照 $8:2$ 划分为训练数据集（4032）与验证数据集（1008）。

使用上面的重采样策略，我们的训练数据集样本量大大扩充。训练集扩充导致验证数据集样

本数量减少，但我们要认清验证集的作用到底是什么：验证集只是模型训练过程中用来评估的数据。它其实并没有为模型训练提供有效信息，它只为模型提供“精度评估报告”而已。因此，验证集样本数量削减不会对模型训练带来大的负面影响。很明显，训练集样本的边际效应递增，验证集样本的边际效应递减。

基于“训练数据越多越好”的真理，我们认为仅从已有数据中“抢占”训练数据是远远不够的，还需要数据增强技术。

3 数据预处理与数据增强

训练数据尺寸为 1080×1920，这对于我们的小服务器来说负担太重。因此在数据预处理阶段，我们选择将所有数据统一到 256×256 大小。注意：我们这里选择的是“等比例缩小”方式，而不是直接裁剪！等比例缩放可以保存关键像素点彼此间的相对位置，而直接裁剪丢弃数据太过直接。

对于训练数据的预处理我们采用了“数据增强”技术来扩大训练样本量。训练集与验证集、测试集的预处理有所不同：训练集不是直接等比例缩小到 256×256，而是先等比例缩小到 300×300，之后在数据增强过程中被随机裁剪到 256×256。数据增强技术实现如下：训练样本在每一轮训练开始前，都会被随机地上下、左右翻转，并并随机裁剪到 256×256 大小（从 300×300 被随机裁剪为 256×256）。由于关键信息（例如图像中虚实分界处）只与其所在图像中的相对位置有关，图像倒转与翻转不会破坏图像中的有效信息；但“每一轮”随机翻转和裁剪（尤其是随机裁剪）后的样本对于网络模型来说是新的数据!。因此数据增强技术可以在不实际增加样本的同时，大量增加训练样本。但需要注意的是：利用数据增强技术创造出来的样本彼此之间有很大的相关性，其价值要比实实在在增加一张新图片低很多！因此，数据增强技术必须在已拥有一定数量的高质量样本条件下才有效果。

验证集与测试集无需进行数据增强，因为它们一个用来验证，一个用来测试，读需要重复进行多次！故要求前后两次图像必须完全一致才有可比性，因此这两个数据集不能进行数据增强。

不论是哪种数据集，预处理阶段都需要进行数据格式转换和归一化操作，其目的显而易见。整个数据集预处理操作都已集成到 Tensorflow2.x 平台，直接调用相关函数即可。下面展示训练数据集、验证数据集、测试数据集的预处理函数写法：

</> CODE 1: 训练数据集预处理与数据增强

```
# 对训练数据及标签的预处理：先等比缩放，再裁剪
# 原始都是1080 x 1920 —— 先缩放到300 x 300 —— 再随机裁剪到256 x
    256 —— 注：512x512情况下计算太慢了!
def preprocessing_train(path, label):
    # 数据/图像读入：.jpg 有R、G、B三通道
    image = tf.io.read_file( path )                          # 文件读取
    image = tf.image.decode_jpeg( image, channels = 3 )    # 文件解码
    # 数据增强：resize + crop
    # 等比例缩放：不用写通道数
    image = tf.image.resize( image, [300,300], method = tf.image.
    ResizeMethod.NEAREST_NEIGHBOR )
```

```
    # 随机裁剪：裁剪时要写最后一维的通道数
    image = tf.image.random_crop( image, size = [256,256,3] )
    image = tf.image.random_flip_left_right( image )  # 随机左右翻转
    image = tf.image.random_flip_up_down( image )     # 随机上下翻转
    image = tf.cast( image, tf.float32 )              # 转浮点类型
    image = image / 255                               # 归一化
    # 标签：label转为嵌套list
    label = tf.reshape( label, [1] )
    # 返回结果：
    return image, label
```

</> CODE 2: 验证集数据预处理

```
# 对训练数据及标签的预处理：只等比缩放到同样的尺寸
def preprocessing_val(path, label):
    # 数据读入：image
    image = tf.io.read_file( path )                          # 文件读取
    image = tf.image.decode_jpeg( image, channels = 3 )      # 文件解码
    # 等比缩放：resize
    image = tf.image.resize( image, [256,256] )              # 等比缩放
    image = tf.cast( image, tf.float32 )                     # 转浮点型
    image = image / 255                                      # 归一化
    # 标签：label转为嵌套list
    label = tf.reshape( label, [1] )
    # 返回结果：
    return image, label
```

</> CODE 3: 测试集数据预处理（同验证集）

```
# 对测试数据及标签的预处理：只等比缩放到同样的尺寸
def preprocessing_test(path, label):
    # 数据读入：image
    image = tf.io.read_file( path )                          # 文件读取
    image = tf.image.decode_jpeg( image, channels = 3 )      # 文件解码
    # 等比缩放：resize
    image = tf.image.resize( image, [256,256] )              # 等比缩放
    image = tf.cast( image, tf.float32 )                     # 转浮点型
    image = image / 255                                      # 归一化
    # 标签：label转为嵌套list
    label = tf.reshape( label, [1] )
    # 返回结果：
    return image, label
```

4 网络框架设计

本文并未采用 MTCNN 网络。本文网络框架设计源于对 U-net 结构的思考：U-net 网络是语义分割任务中的常用的像素级的模型，其结构整体类似于 U 型：网络左半部分为特征提取，右半部分为图像尺寸恢复和标签预测，用条件连接操作（skip connection）将左、右两部分连接起来。U-net 框架的核心思想是“先特征提取，后恢复预测”，基于此思想，我们设计了一个“**陀螺型**”的网络结构：该结构两头窄，中间粗，如图 2 所示：

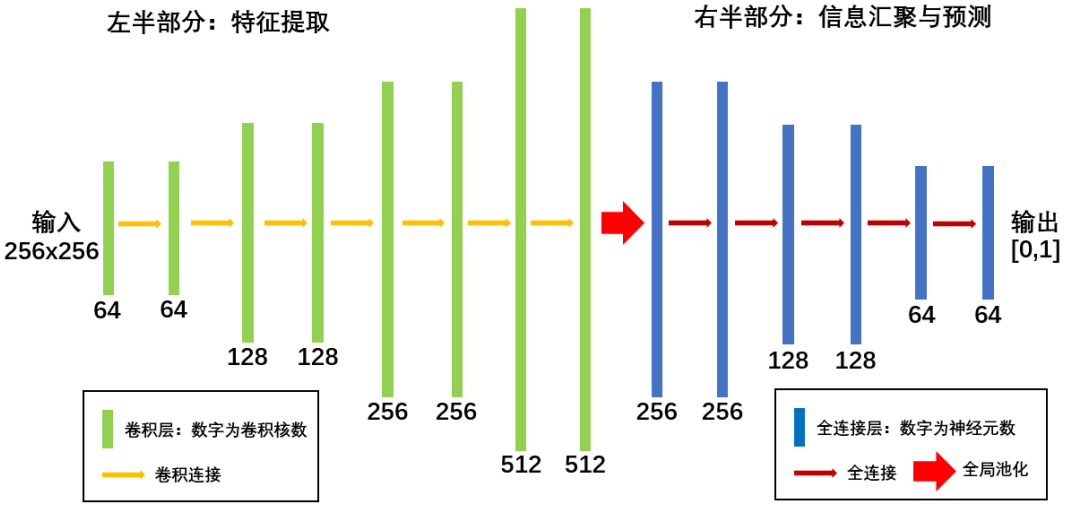

图 2. 网络结构

标签重新设定为 1（真数据）和 0（假数据），而并非原定的 1 和 −1。因为最后为二分类问题，当标签为 0 和 1 时，使用 sigmoid 激活函数时的网络预测结果会落在 [0,1] 之间，这样可以之间根据数值大小来判断模型预测为哪个标签。

由于计算资源实在有限，最终选择图 2 的简易结构（比该网络再大将超出计算机内存）。

5 模型训练

我们认为最终训练好的模型效果非常好，其中关键之一的操作就是在模型训练过程中使用了“**回调函数**”来监控整个模型的训练。经过多次测试发现：网络在训练过程中经常发生精度值的“跳变“：虽然模型对训练集的精度稳步攀升，但是在验证集上经常出现较大尺度的精度值跌落和回弹！此时“回调函数”的使用尤为关键：编写程序时，回调函数监控“验证集精度”这个指标，并实时保存使得该指标获得最大值的模型。

由图 3 可以看出：在该数据集下训练，模型在验证集上的数值跳变现象非常明显！上一轮训练的精度可以在 98% 以上，下一轮训练可以就会降到不到 70%。本文使用的网络一共训练了 100 个 epoch，并在第 62 轮训练由回调函数自动保存了一个最佳模型：该模型在训练集、验证集上的精度都超过了 99.5%。因此，模型训练过程中回调函数的使用必不可少。

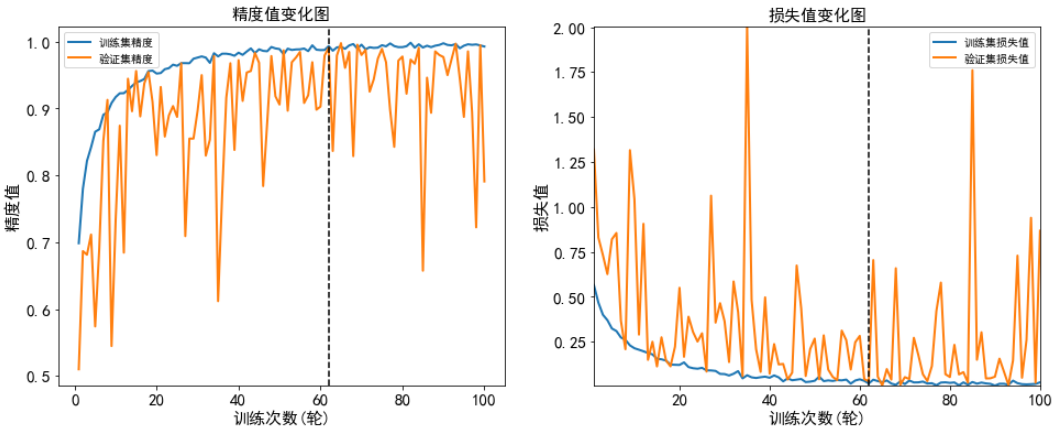

图 3. 模型训练过程中的精度、损失值变化图

6 模型调优策略

本报告使用“**迁移学习**”策略作为模型参数调优策略。操作如下：导入由回调函数保存得到的历史最佳模型，冻结前面所有层的参数，使其不可在之后的训练中改变，仅保留靠近输出层的最后一个全连接层，并在更小的学习速率下重启训练。经测试，该策略总是有效的，可再次小幅提升模型在验证集上的预测精度。但是，由于前面用回调函数保存得到的模型效果已经非常好，且训练过程中没有对数据集进行“**交叉验证**”！所有虽然迁移学习使得精度提升了不到 0.2%，但我们最终决定不采用微调后的结果！

7 训练结果

最后的模型以.h5 的格式保存，整体大小为 19M。预测精度为：

1. 训练集：99.92%
2. 验证集：99.88%
3. 测试集：99.76%
4. B_test 测试集：没有验证。因为感觉数据集标签有问题。

8 改进策略

首选，我们想尝试 U-net 的思路来测试预测效果。虽然 U-net 模型是对每个像素点都进行标签，但只要我们将其最后一层之后添加一个全连接层，就可以改造成为一个可以用来做二分类任务的网络。我们非常看重 U-net 网络的设计思路，但可惜其计算资源消耗过大，我们无法运行起来。

其次，可以对图像进行频谱分析。图像中的“虚化与锐化”，表现在频谱图像中是“低频和高频”。往往在图像域（时域）中差不大的区别，在频率域会将这种差别放大。因此，可以使用：二

维离散傅里叶变换、二维离散希尔伯特-黄变换、2 维 EEMD-CCA 等算法挖掘数据集中关于“**频谱**”的新特征。

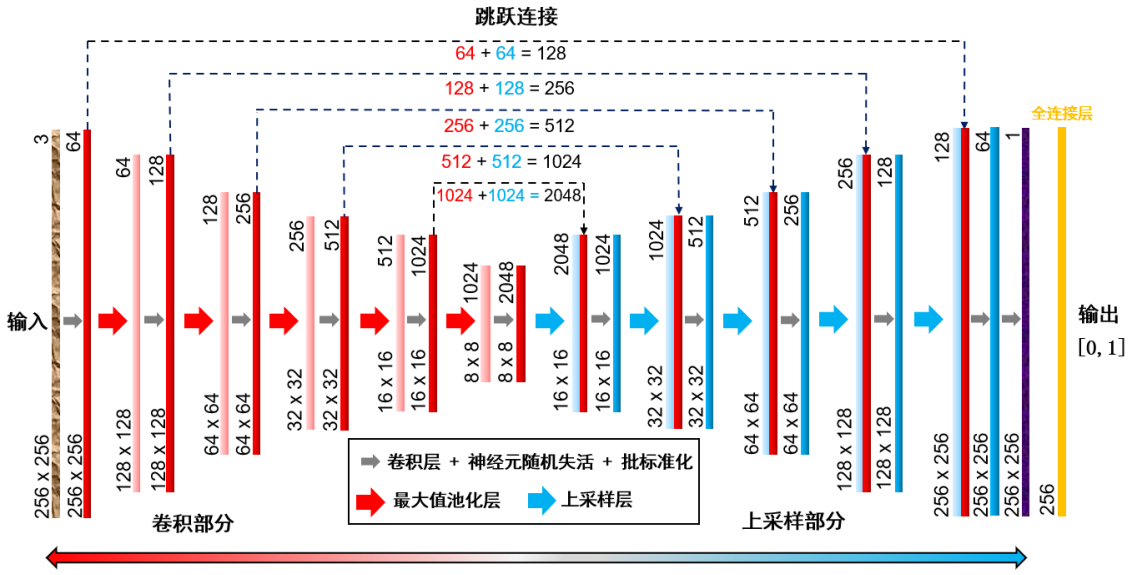

图 4. 想尝试的 Unet 框架进行二分类预测

最后一种想法：在资源勘探领域，常会使用“**向上延拓**”和“**向下延拓**”这两种操作处理采集得到的数据。野外数据集采集时，采集点一般都在地表。“向上延拓”是假设数据的采集点在空中，即从理论上将采集点上移！这样操作的带来的好处是压制采集数据中的“局部特征”，凸显“全局/背景特征”。反之，“向下延拓”是假设数据的采集点在地下，这样操作的带来的好处是压制采集数据中的“全局/背景特征”，凸显“局部特征”。