内容安全实验报告

基于 ResNet 的不良图片识别模型

学　　院　：

专　　业　：

班　　级　：

课 程 名 称　：内容安全

指 导 教 师　：

学 生 学 号　：

学 生 姓 名　：

小 组 成 员　：

二〇二一年六月

目录

一、Abstract

随着现代移动互联网设备以及社交软件的迅速发展，网络用户每天都会上传大量的图片内容到互联网中，为保证网络空间的环境清明，通过各种手段来识别和过滤网络色情图片是很有必要的。针对实际应用中色情图片的复杂多样性问题，运用了基于 ResNet（深度残差网络）的不良图片识别模型，对图片进行判类、不良程度打分，从而实现真对不良图片的自动化判例。测试结果表明，该方案在时间效率、模型置信度以及识别准确率上具有良好的效果。

二、Introduction

2.1 研究背景

随着 Web 2.0 时代的到来，用户在浏览网络空间的信息的同时，也能自由地发表自己的看法、观点，越来越多的人们开始以图片、视频等形式在网络上分享自己的生活。然而，受利益驱动，一些不法分子会借此在论坛、贴吧、博客、微博等公共平台中上传一些不雅视频、照片以吸引流量或达到其他目的，这使得网络淫秽色情现象仍然较为严重。

一方面，随着越来越多的年轻用户、儿童群体接触互联网，网络空间中的不雅图片会对其造成巨大的伤害；另一方面，在公司办公场所和学校教育场所等地方， 应该过滤掉这些网络不良内容。基于此，2021 年 3 月，全国“扫黄打非”办公室作出安排部署：即日起至 11 月底开展“新风 2021”集中行动，“新风”行动将以“护苗 2021”“净网 2021”“秋风 2021”专项行动为开展平台，通过摸底排查、接收群众举报等方式对网络空间进行整治。

然而人工的成本是巨大的。因此，随着机器学习和人工智能的快速发展，越来越多的理论被提出，越来越多的技术开始代替人工、被应用于不良图片的鉴别。在这个过程中，如何通过技术手段更准确、更有效地识别和过滤网络色情图片，一直是一个重要的研究课题。国内外的许多学者对不良图片的识别技术做出了许多尝试，并取得了一些进展。

2.2 相关研究

常用的检测方式有三种：基于肤色检测的识别模式、基于多种特征结合检测的识别模式、基于深度学习的方法。

基于肤色检测的识别模式是最典型的识别方法，这种方法主要基于色情图片中往往包含有大量的裸露肤色信息这一事实，进而通过肤色检测来判断一张图片是否为不良图片。RGB 颜色空间、YCbCr 颜色空间[1]都曾被利用以检测肤色；有的理论使用多重颜色空间定义肤色模型，同时加入纹理检测以更准确地提取出肤色区域，进而结合裸露肤色区域的像素比例、肤色连通域的数目、位置、形状[2]等特征，将其作为 SVM 等分类器的输入并判断一张图片是否为不良图片。这类基于肤色特征和人体检测的方法误检率较高，因为其存在有如下的一些问题：现实中不同种族之间存在有肤色差异、相同的肤色在不同的光照条件下会呈现不同的结果、自然界中的一些与肤色类似的物体会对检测结果存在干扰。

基于多种特征结合检测的识别模式往往会结合不同的人体识别技术来提高人体检测的准确率，如人体动态姿势识别的多层联合模型[3]、在具有复杂背景的图像中结合图像区域检测技术[4]进行更准确的场景分割与识别等，通过引入其他的技术预先进行检测、随后再进行特征识别的方式，进一步提高了准确率。

随着深度学习理论的发展，基于深度学习的方法也逐渐被应用到这一领域中。这种发放是把特征提取与分类器放在一个模型里统一起来，模型通过对大量标注的训练数据进行自动学习，从而得到一套可用于分类的参数。从最简单的 CNN（卷积神经网络）模型，到基于 CNN 的 AlexNet、AGNet、VGGNet 和 GoogleNet 等模型，再到多种模型的联合使用，基于深度学习的不良图像检测方法已经成为了综合评价较高的一种方法，其应用最为广泛、成本相对较低、准确率相对较高。

在基于深度学习的检测方法中，就理论上而言，随着神经网络层数的加深，模型可以对更为复杂的特征进行提取，从而获得更好的分类结果。然而实验表明，实际上网络会出现退化问题（degradation problem）：刚开始随着网络深度的增加，训练效果可以得到一定的提升；然而在达到某一个阈值之后，随着网络深度的继续增加，网络的训练精度反而下降了，且这种退化并不是由过拟合引起的，同时在适当的深度模型上添加更多的层会导致更高的训练误差。

在本项目中，我们采用了基于深度学习的检测方法，通过预标注的数据集训

练一个神经网络模型，对不良图片进行分类检测。本项目中将采用单一网络对图片进行简单分类，并构建一个网页供用户自行上传图片进行检测。

考虑到不良图片的分类检测是一个很复杂的问题，为了保证在加深神经网络的同时不影响网络的性能，同时受限于时间和能力，在进行综合考量后，我们决定采用 ResNet（残差神经网络）[5]作为我们的分类模型。

三、The proposed scheme

3.1 数据获取及标注

由于目前网络上缺少一个公开的、具有一定标准的不良图片数据集，故本次实验采用自行获取数据并构建数据集并进行标注的方式。数据获取将结合 python opencv 库，将收集到的不良视频运用分帧的形式剪切为图片数据集。为保证数据的差异性，选图的决策为每个视频间隔固定时间选取 1 张的方式获取 50 张图片，即用总帧数除以 50 作为选取图片的时间间隔。将得到的图片数据保存到本地，然后以打分的形式人工对所有图片进行标注，所选分值为 0、1、2、3。其中 0 代表正常图片，1、2、3 代表色情图片，并根据图片色情程度确定图片最终分值，程度越重分值越高。

3.2 数据预处理

根据我们的设计，模型有 2 分类检测模型和 4 分类评级模型，其中 2 分类检测模型用于检测输入图片是否含有不良信息，4 分类评级模型将用于对图片内容的不良程度进行评估。

根据检测模型与评级模型的区别设计了两种数据集：2 分类检测数据集和 4 分类评级数据集。其中 2 分类检测数据集有两个标签：0 表示图像不是不良图像，1 表示图像可能含有不良内容；4 分类评级数据集有 4 个标签，其中 0 表示不是不良图像，从 1 开始，数字越高表示不良内容越严重。因此，针对 2 分类检测数据集的数据标注比较好标，但是针对 4 分类评级数据集的数据标注会不可避免地受到人工判断的影响导致标注界限较为模糊，进而影响模型性能，这点我们会在

结果分析中进行详细说明。

我们获取到的图像的尺寸大多在 720 * 406 左右，在训练时要对其进行裁切和重置尺寸等操作。一般而言，在条件允许的情况下，用 ResNet 做图像分类时，输入图像的尺寸为 224 * 224 是比较好的，batch size 取值为 64、128 等值时是比较好的。然而，受限于硬件条件，我们只能将输入图像的尺寸重整为 56 * 56，并将 batch size 设置为 8，这将在一定程度上影响实验结果（尝试过将图像尺寸和 batch size 分别向大值调整，然而稍微大一点就会用尽内存）。

具体实现上，借助 torchvision 库中的 transforms、ImageFolder 以及 torch 库中的 DataLoader 方法划分训练集、验证集和测试集。其中训练集的 transform 设置如下图所示：

```
# 设置训练集
train_transform = transforms.Compose([
    transforms.RandomResizedCrop([h, w]),  # 重置分辨率为w * h大小
    transforms.RandomHorizontalFlip(),  # 随机水平翻转
    transforms.RandomRotation(degrees=15),  # 随机旋转
    transforms.ToTensor(),  # 转化成Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # 正则化
])
```

重设分辨率到 56 * 56， 进行随机水平翻转、随机旋转、正则化操作。

测试集和测试集的 transform 设置如下图所示：

```
test_transform = transforms.Compose([
    transforms.Resize([h, w]),  # resize到w * h大小
    transforms.ToTensor(),  # 转化成Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # 正则化
])
```

重设分辨率到 56 * 56， 并进行正则化操作。

划分训练集 : 测试集 : 验证集的比例为 8 : 1 : 1。

3.3 分类模型 – ResNet

本项目的分类核心是利用 ResNet（深度残差网络）构造一个图片分类器。不同层数的 ResNet 结构有所不同，具体构造如图 3.3.1 所示。本项目中构造了 18 层 ResNet 结构和 50 层 ResNet 结构，具体分类采用了 50 层的 ResNet 结构。

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 3.3.1 ResNet **结构**

实际上，ResNet 网络参考了 VGG19 网络，并在其基础上进行了修改，并通过短路机制加入了残差单元，如图 3.3.2 所示。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 设计原则之一是：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。

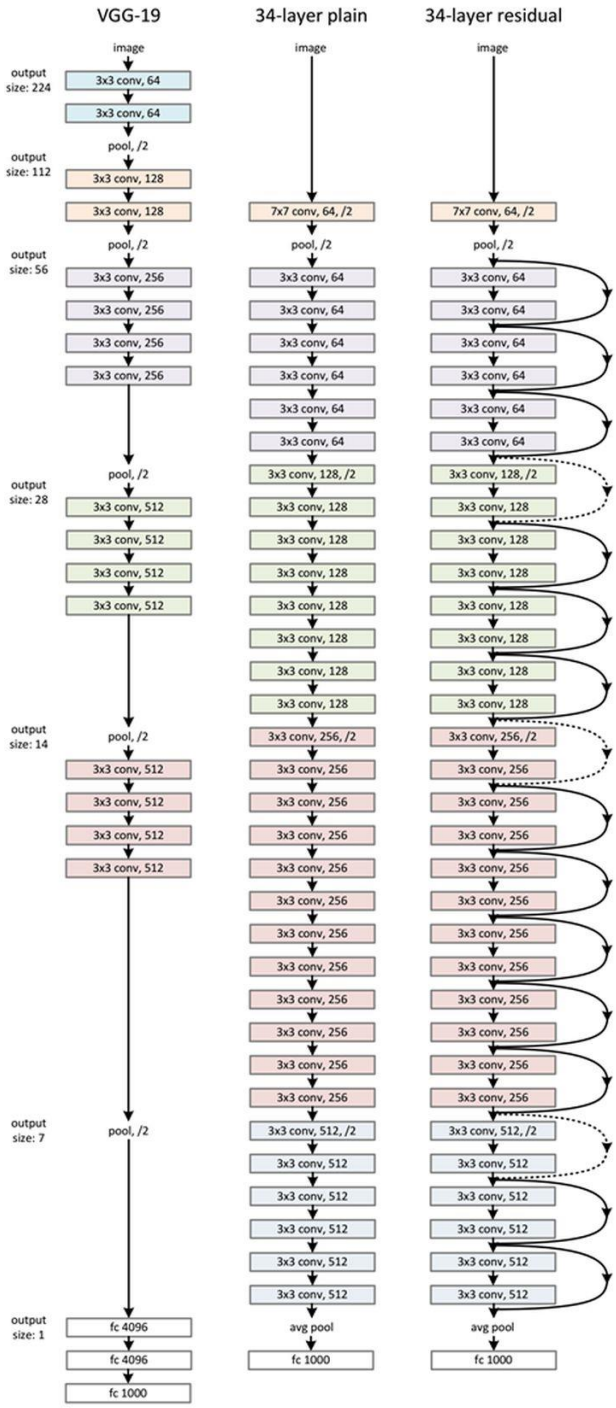


图 3.3.2 ResNet **网络结构图**

从图 3.3.2 中可以看到，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。图 3.3.2 展示的是一个 34-layer 的 ResNet，可以根据图 3.1.1 构建更深层次的 ResNet。

对于 18-layer 和 34-layer 的 ResNet，其进行的两层间的残差学习；当网络更深时（比如本项目中的 50-layer ResNet），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。需要注意的是，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。ResNet 使用的两种残差单元如图 3.3.3 所示，其中图左对应的是浅层网络，即每两层进行一次残差学习；图右对应的是深层网络，即每三层进行一次残差学习。

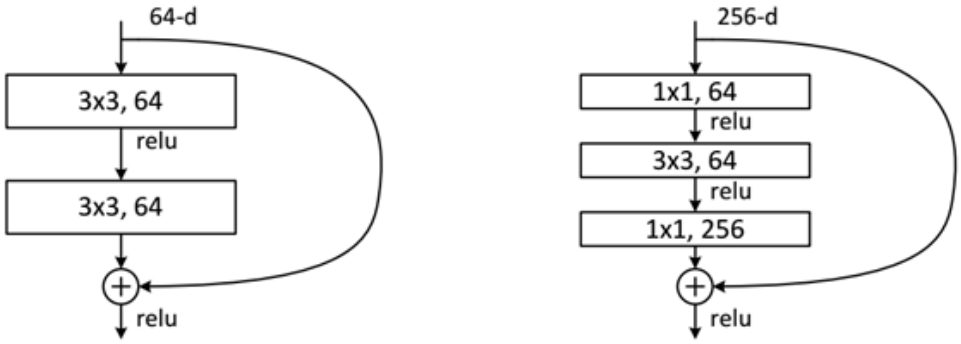


图 3.3.3　不同的残差单元

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；当维度不一致时（对应的是维度增加一倍），此时有两种策略：

1、采用 zero-padding 增加维度，此时一般要先做一个降采样，可以采用 strde=2 的池化，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，但同时也会增加计算量。

本项目中采取的是策略 1。

具体实现上，18 层 ResNet 的残差单元如下图 3.3.4 所示：

```python
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(nn.Conv2d(in_planes, self.expansion * planes,
                                                    kernel_size=1, stride=stride, bias=False),
                                          nn.BatchNorm2d(self.expansion * planes))

    def forward(self, x):
        # print(x.shape)
        out = F.relu(self.bn1(self.conv1(x)))
        # print(out.shape)
        out = self.bn2(self.conv2(out))
        # print(out.shape)
        out += self.shortcut(x)
        # print(out.shape)
        out = F.relu(out)
        return out
```

图 3.3.4　18 层 ResNet 的残差块

50 层 ResNet 的残差单元如下图 3.3.5 所示：

```python
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion * planes, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion * planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion * planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion * planes))

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

图 3.3.5　50 层 ResNet 的残差块

ResNet 的基本构造如图 3.3.6 所示（未展示前向传播模块）：

```python
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, in_channels=3, num_classes=4):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(in_channels, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)

        self.linear = nn.Linear(512 * block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)
```

图 3.3.6 ResNet 模型设计

ResNet 的前向传播如图 3.3.7 所示：

```python
def forward(self, x):
    # print(x.shape)
    out = F.relu(self.bn1(self.conv1(x)))
    # print(out.shape)
    out = self.layer1(out)
    # 特征图个数没变，输入时64， 输出也是64， 在shortcut中不需要调整
    # print(out.shape)
    out = self.layer2(out)
    # print(out.shape)
    out = self.layer3(out)
    # print(out.shape)
    out = self.layer4(out)
    # print(out.shape)
    out = F.avg_pool2d(out, 4)
    # avgpool = nn.AdaptiveAvgPool2d((1, 1))
    # out = avgpool(out)
    # print(out.shape)
    out = out.view(out.size(0), -1)
    # out = out.reshape(out.shape[0], -1)
    # print(out.shape)
    out = self.linear(out)
    # print(out.shape)
    return out
```

图 3.3.7 前向传播模块

18-layer 的 ResNet 和 50-layer 的实现如图 3.3.8 所示：

```python
def ResNet18(in_channels, num_classes):
    return ResNet(BasicBlock, [2, 2, 2, 2], in_channels=in_channels, num_classes=num_classes)


def ResNet50(in_channels, num_classes):
    return ResNet(Bottleneck, [3, 4, 6, 3], in_channels=in_channels, num_classes=num_classes)
```

图 3.3.8 网络的具体实现

3.4 网页展示

为了使我们的项目有一个更直观的展示，也为了我们自己能够更好地调试，在本次项目中我们定制了一个呈现于浏览器上的人机交互页面。我们使用一款轻量级的机器学习可视化工具 streamlit 来搭建我们的前端展示页面，streamlit 支持使用纯 python 的语言来生成一个可交互式的站点，无需编写任何 HTML、CSS 或 JS 代码就可以生成界面不错的页面。

如上面几个部分所述，在本次项目中我们分别实现了基于 CNN 和 ResNet 两种模型的不良图像检测系统，虽然前者的正确率不尽人意，但也正突出体现了后者性能的优越性，所以在前端展示页面中我们同时向用户提供了基于两种模型的检测模式。根据实验的需求，交互界面应包含以下两个功能：

（1）提供一个友好的界面来操纵模型的训练和测试（“友好”的具体体现为在交互界面上可以完成各种参数的调整，以及对整个训练过程的监控，将训练产生的数据进行可视化等）。

（2）提供一个友好的用户使用界面。我们项目的宗旨在于实现对不良图像的检测，所以前端页面应该支持用户对选定的图像进行不良性检测。

需要说明的是，使用 streamlit 搭建的交互式网页需要使用自己的机器作为服务器，所以在使用展示页面之前，需要在自己的机器上下载相关的代码、数据，并配置好相应的依赖环境。

四、Experiment

4.1 实验配置

(1)处理器：

Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz

(2)操作系统环境：

Windows 10 家庭中文版 x64 21H1 19043.1052

(3)编程语言：

Python 3.8

(4)其他环境：

16 GB 机带 RAM，

NVIDIA GeForce RTX 2060，

6G 显存

(5)IDE 及包管理器：

JetBrains PyCharm 2020.1 x64，

anaconda 3 for Windows（conda 4.9.0）

(6)使用的第三方库：

datetime， matplotlib.pyplot， numpy， os， pickle， PIL，sklearn.metrics， torch， torch.nn， torch.nn.functional，torch.optim ， torch.utils.data ， torchvision.datasets ，torchvision.transforms，tqdm

(7)数据集大小：

总共 2511 张，其中训练集 2008 张、测试集 251 张、验证集 252 张，训练集 ： 测试集 ： 验证集的比例约为 8 ： 1 ： 1。

4.2 训练情况

2 分类的训练日志（部分）如图 4.2.1 所示：

```
 96%|█████████| 96/100 [1:22:24<03:28, 52.08s/it]dev AUC: 0.84965 ACC: 0.85141
train AUC: 0.83793 ACC: 0.83815
test AUC: 0.86995 ACC: 0.86853
 97%|█████████▋| 97/100 [1:23:16<02:35, 51.99s/it]dev AUC: 0.82653 ACC: 0.82731
train AUC: 0.82758 ACC: 0.84064
test AUC: 0.83025 ACC: 0.84861
 98%|█████████▊| 98/100 [1:24:09<01:44, 52.21s/it]dev AUC: 0.82704 ACC: 0.77912
train AUC: 0.8169 ACC: 0.83715
test AUC: 0.87582 ACC: 0.85259
dev AUC: 0.8204 ACC: 0.80321
 99%|█████████▉| 99/100 [1:24:59<00:51, 51.83s/it]train AUC: 0.82384 ACC: 0.81723
test AUC: 0.761 ACC: 0.77689
dev AUC: 0.7596 ACC: 0.78715
Finish at:  2021-06-09 16:43:22
epoch 12 is the best model with auc 0.91861 and acc 0.82869
==> Testing model...
100%|██████████| 100/100 [1:25:51<00:00, 51.52s/it]

Process finished with exit code 0
```

图 4.2.1 2 分类的训练日志

2 分类训练的 AUC 随训练轮次（epoch）的变化折线如图 4.2.2 所示：

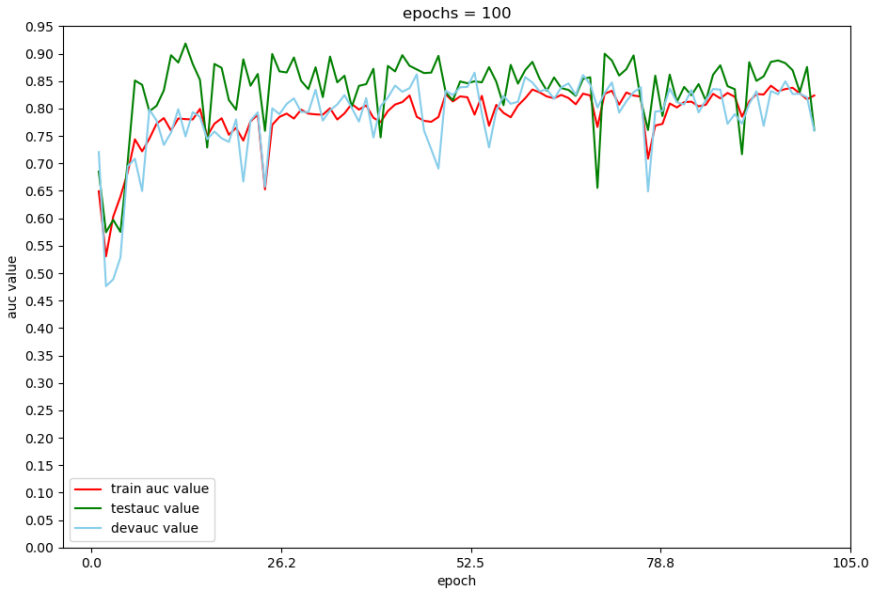


图 4.2.2　2 分类模型的 auc-epoch 图像

2 分类训练的 AACCUC 随训练轮次（epoch）的变化折线如图 4.2.3 所示：

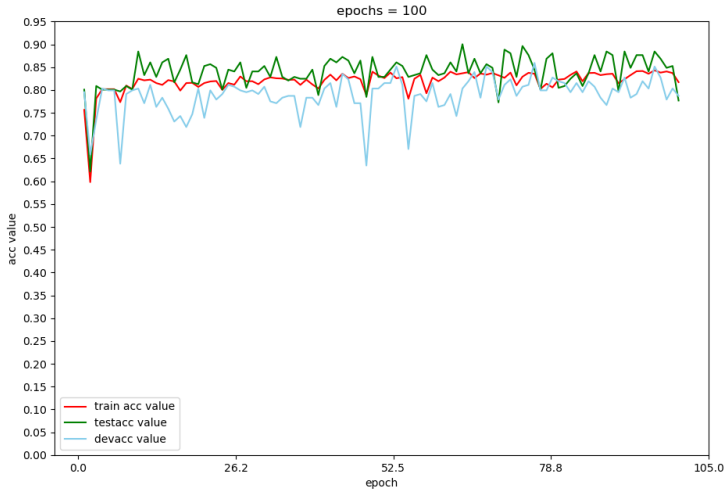


图 4.2.3　2 分类模型的 acc-epoch 图像

4 分类的训练日志（部分）如图 4.2.4 所示：

```
train AUC: 0.69591 ACC: 0.4791
test AUC: 0.68213 ACC: 0.44841
 96%|          | 96/100 [1:22:41<03:29, 52.33s/it]dev AUC: 0.67889 ACC: 0.43145
train AUC: 0.68908 ACC: 0.45174
test AUC: 0.67744 ACC: 0.43254
 97%|          | 97/100 [1:23:34<02:37, 52.40s/it]dev AUC: 0.63827 ACC: 0.36694
train AUC: 0.68782 ACC: 0.47015
test AUC: 0.69879 ACC: 0.42063
 98%|          | 98/100 [1:24:28<01:45, 52.78s/it]dev AUC: 0.64591 ACC: 0.41935
train AUC: 0.69072 ACC: 0.46517
test AUC: 0.70606 ACC: 0.42063
 99%|          | 99/100 [1:25:19<00:52, 52.44s/it]dev AUC: 0.66777 ACC: 0.37097
train AUC: 0.68698 ACC: 0.45821
test AUC: 0.76992 ACC: 0.54762
100%|          | 100/100 [1:26:11<00:00, 51.71s/it]
dev AUC: 0.64724 ACC: 0.3629
Finish at:  2021-06-09 19:10:48
epoch 99 is the best model with auc: 0.76992 and acc: 0.54762
==> Testing model...

Process finished with exit code 0
```

图 4.2.4　4 分类的训练日志

4 分类训练的 AUC 值随训练轮次（epoch）的变化折线如图 4.2.5 所示：

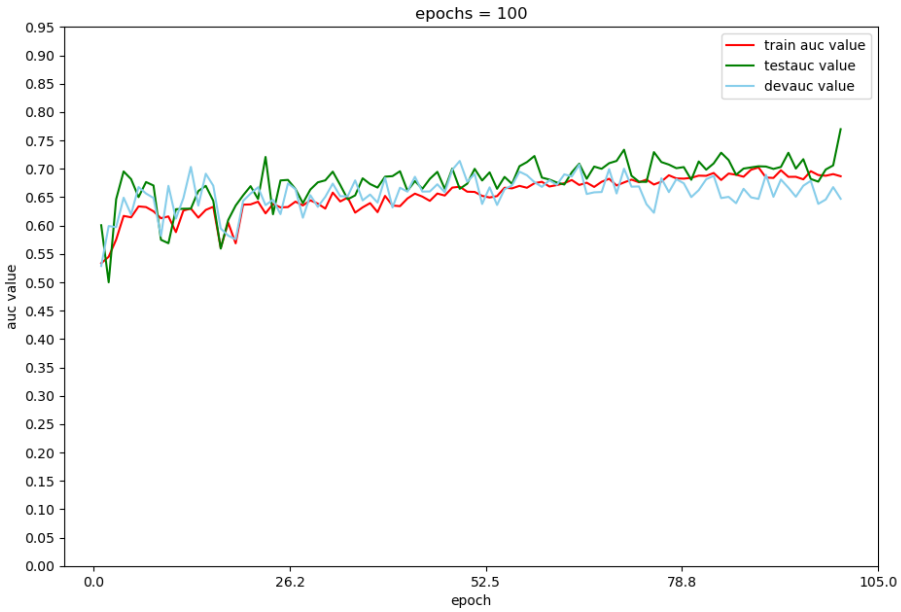


图 4.2.5　4 分类评级模型的 auc-epoch 图像

4 分类训练的 ACC 值随训练轮次（epoch）的变化折线如图 4.2.6 所示：

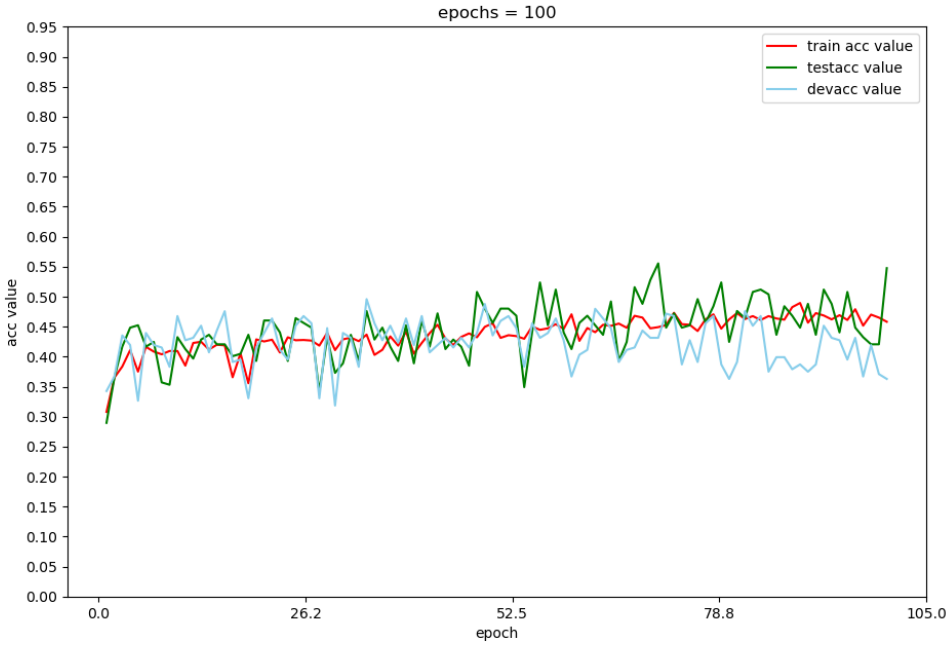


图 4.2.6 4 分类评级模型的 acc-epoch 图像

4.3 模型结果分析

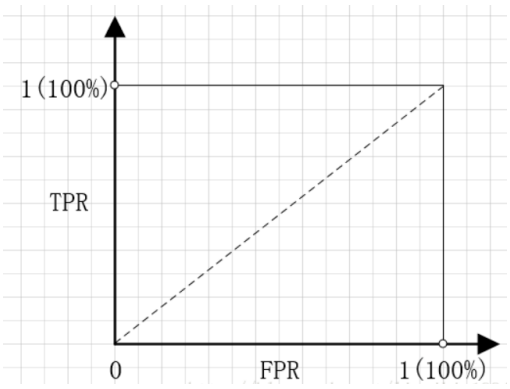
对于模型结果采用了 AUC 和 ACC 两种值进行评估，其中 ACC 指的是预测正确的样本个数占待检测样本个数的比例；AUC 的全称是 AreaUnderRoc 即 Roc 曲线与坐标轴形成的面积，取值范围 [0, 1]。

Roc (Receiver operating characteristic) 曲线是一种二元分类模型分类效果的分析工具。Roc 空间将伪阳性率(FPR)定义为 X 轴，真阳性率(TPR)定义为 Y 轴，FPR 和 TPR 的计算公式如下：

TPR: 在所有实际为阳性的样本中，被正确地判断为阳性之比率 TPR = TP/P = TP/(TP+FN)；

FPR: 在所有实际为阴性的样本中，被错误地判定为阳性之比率 FPR = FP/N = FP/(FP + TN)。

如果以 FPR 为横坐标，TPR 为纵坐标，就可以得到下面的坐标系：



点(0,1)，即 FPR=0、TPR=1，代表没有假正例、没有假反例，是最完美的情况；点(1,0)，即 FPR=1、TPR=0 则代表最糟糕的情况；点(0,0)和点(1,1)分别代表全预测为反例和全预测为正例的情况，分别代表两个极端。因此，如果一个点越接近左上角，则说明模型的预测效果越好。

模型在预测时需要设定阈值才能将概率转换为类别，进而才能得到 FPR 和 TPR。而选定不同的阈值会得到不同的 FPR 和 TPR。当我们不断改变阈值，就会得到不同的 FPR 和 TPR；如果将得到的(FPR, TPR)连接起来，就可以得到 ROC 曲线。如果想用 ROC 来评估分类器的分类质量，可以通过计算 AUC 来评估，因此 AUC 表示的是正例排在负例前面的概率。即，AUC 的值越高，模型的置信度越高。因此，AUC 最高的模型将被保留下来，作为训练好的预测模型。

从结果图中可以看到，无论是置信度还是准确率，2 分类检测模型都显著高于 4 分类评级模型的准确性。这是因为数据集和训练参数的原因，一方面是因为数据量不够大；另一方面也是因为四分类评级不同级别之间的区别并不明显，且存在有一定的人工偏差；此外，硬件条件也在一定程度上限制了模型训练时送入神经网络的数据大小，进而影响了模型在进行特征提取时的准确性。

4.4 前端展示

前端界面的整体布局如图 4.4.1 所示

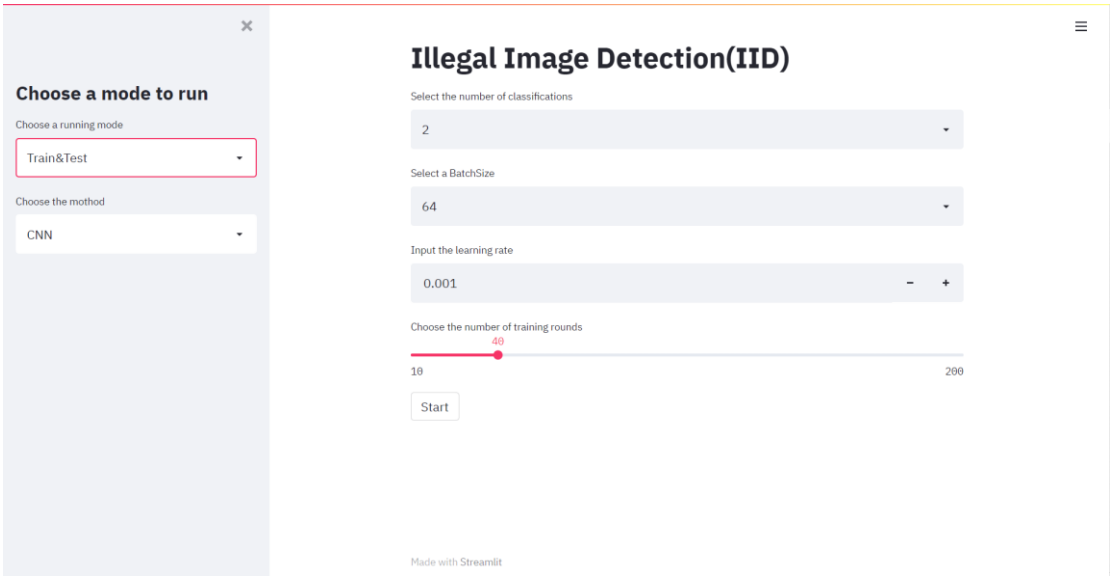


图 4.4.1　前端页面整体布局

可以看到整个页面分为两个部分——左边是功能选择部分，可以选择运行的模式（训练、测试以及对单张图片的检测）和所使用的模型（CNN、ResNet）；右边是选择了运行模式之后各种参数以及数据的输入和调试窗口，以及数据展示页面。

图 4.4.2　CNN **模型的训练和测试**

图 4.4.2 展示的是 CNN 模型的训练以及测试过程（因为 ResNet 训练时间过长，在这里不方便进行展示）。上面的参数调试窗口可以选择使用的数据、学习率、每一批次训练的图片数量以及训练的轮次，设定这些必要的参数之后点击 Start 按钮开始训练和测试。可以看到在前端中依次出现了“Loading data...”、“Training...”、“Testing...”等提示信息，这对应着模型的“数据加载 -> 训练 -> 测试”的过程。训练结束之后会展示一个 loss-epoch 曲线图（ResNet 展示的是 acc-epoch 和 auc-epoch）反映训练的情况，测试结束之后则是会打印相

关的正确率。

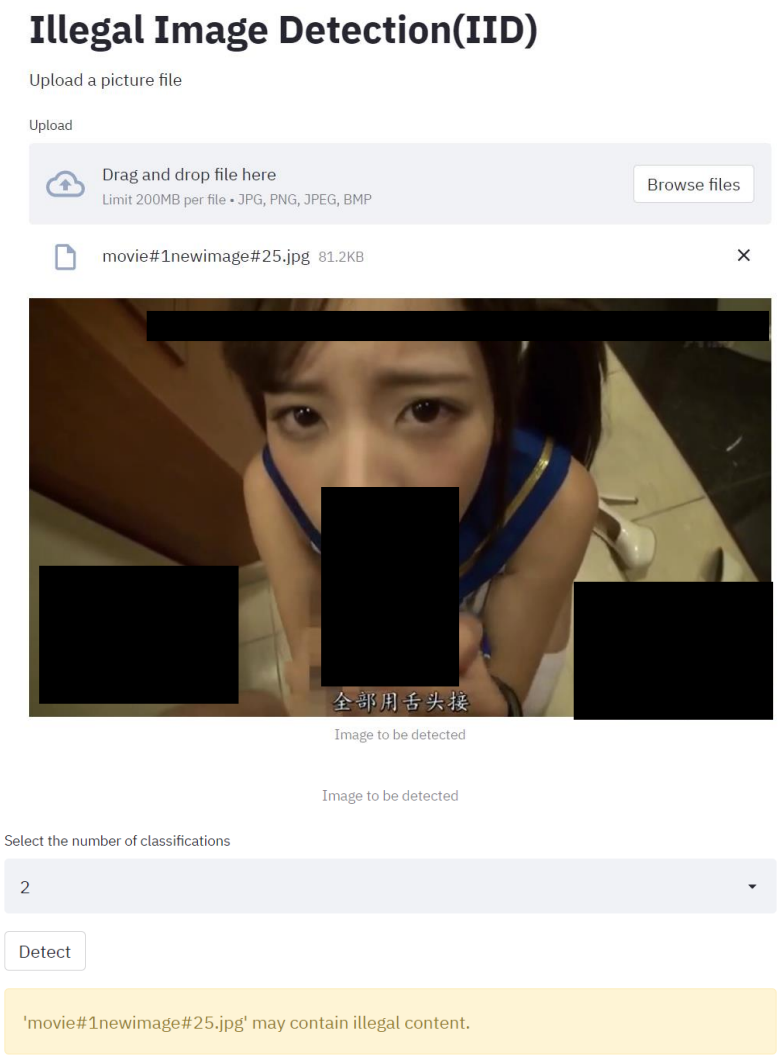


图 4.4.3　对单张图片进行检测

图 4.4.3 展示的是前端页面的检测功能，点击“browse”按钮选择保存在本地待检测图像，图像会加载到系统中，然后使用已经训练好的模型进行检测，最后检测结果以文本的形式展现。

五、Conclusion

1、对项目的理解

在存在于网络空间中的众多不良多媒体信息中，不良图片、不良文本、不良

视频是不良信息的主要载体。随着相关理论技术的发展，神经网络已经越来越多地被应用于不良信息的识别检测工作，并取得了一定的成效。因此，如何提高神经网络在这个过程中的准确率是一个重要课题。

本项目中只是简单地对图像进行二分类和四分类，实际上是不完善的。对于一个不良图片检测项目而言，其要判别的图片必然是多样的，可能包含有人、可能不含有人，含有的色情内容可能是儿童色情、也可能是成人色情、有也可能是性暗示等等，不能单一地用一套模型进行判别。因此，一个成熟的项目应该至少包括如下几个流程：场景识别、人体检测、形体检测、内容检测、严重程度评级。

本项目仅仅采用 ResNet 对图像进行分类，其实是有一定前提的，即必须假定输入的图像是可能存在有敏感信息的。这个前提与本项目的定位是有关系的，本项目的定位是在内容检测和严重程度评级步骤，即已经通过 CNN 或其他神经网络模型分拣出含有人体的图片，随后通过本项目对其进行检查、评级工作。同时在实际工作时应当考虑到，不同的不良图片的不良内容可能分布在图片的不同位置，所以不能简单地通过裁切和重整大小来对数据进行预处理。比较合理的方式是将图片切块或进行其他操作（如关键部位检测）再送入神经网络。因此，本项目还存在有一定的提升空间。

对于一个成熟、完善的项目而言，一般都需要采用多模块、多技术相结合的方法，才能极大地提高项目的准确性。因此，如果后续有条件的话，我们会考虑对项目进行进一步的完善工作。同时，不良图片的甄别也是具有重要的实用意义和广阔的前景空间，因此我觉得我们的工作尽管存在有瑕疵，但仍然是具有一定的意义的。

2、承担的工作

在此次项目中，我主要承担的任务是数据的预处理、ResNet 的搭建以及模型的训练、测试、评估部分。

具体来讲，在数据的预处理过程中，我负责的是针对已经标注好的数据集，分别构建 transformer 并将其按照 8：1：1 的划分训练集、测试集和验证集。其中在 transformer 的过程中涉及到图像的调整问题，这里我简单地采取将图片缩放为 56 * 56 大小并引入随机旋转的方式。

一方面， 在将图片缩放为56 * 56的过程中，图像会受到一定程度上的破坏，很多信息并不会被保留下来，这在一定程度上会影响分类的准确性。

另外一方面，对transformer的简单而粗暴的操作是存在一定缺陷的。一般而言，一张图片中所包含的不良内容往往是存在于图像的某个“块”而非图像本身，因此对图像做整体检测实际上不如做分块检查。

预处理之后就是将分好的数据集分batch送入搭建好的ResNet训练指定轮次，然后保留auc最高的模型作为训练好的模型。关于ResNet，代码中其实搭建了有以下几种的ResNet模型：18-layer、34-layer、50-layer、101-layer和152-layer，但实际训练时考虑到机器的性能，就只训练了50-layer的模型。

训练好后会在测试集和验证集上分别做auc和acc的计算，在验证集上的效果最好的模型将被保留下来，其将被用于检测。训练过程中的loss图像以及验证过程中的acc和auc图像将被保存下来。

对于被保存下来的模型，用户向模型输入一个图片，模型会进行预测并返回一个结果，如果模型认为图片是正常的，就会在控制台输出’Image is considered no problem.’，如果模型认为图片可能含有不良内容，就会在控制台输出’Warning! Image may contain illegal content.’，如果模型确定图片含有不良内容，则会在控制台输出’FBI WARNING! Image contains illegal content!’。

3、体会

1、硬件条件对实验性能的影响是巨大的。

我们将图片缩放为56 *56后再输入模型进行训练，这样的调整其实是我们向硬件妥协的结果，因为在实验中本机能支持的最大数据量就是这样，再大的话将会报错Out of Memory。在我进行调查时发现，很多效果比较好的图像分类模型，其一般是将图像缩放为224 * 224后再进行训练，其训练用的GPU都是12G的。

2、对整张图像进行判别的操作有些粗糙。

实验中我们直接对图像进行分类，这样的处理其实是有些粗糙的。比较好的操作方式是，先将图片切块并分类标注，然后训练某个神经网络模型，最终得到

一个能够检测一个图像块中是否含有不良内容的分类模型；然后引入一个滑动窗口，让这个分类模型对窗口所提取到的图像进行内容检测；最后再根据检测到的不良内容的类别进行打分，得到这张图片对应的分数，这个分数反映了这张图片的严重程度。

3、数据集的好坏程度会极大地影响模型的效果。

从理论上来讲，ResNet 凭借其更深的网络层数、引入的残差学习的特性，在分类问题上应该能够达到更好的分类效果才对，如参考文献[5]中作者搭建的 152-layer ResNet 在图像分类领域的 ImageNet 测试集上仅有 3.57%的错误率，而在我们的模型中错误率却达到了近 18%（2 分类）以及近 46%(4 分类)，错误率之高、2 分类与 4 分类之间的差别之大，这种现象显然是不合常理的。唯一的解释就是，人工标注对数据集的可信性产生了较大影响，尤其是对于涉及不良内容的图片的评级，标准的模糊将极大地影响不同类别之间的特征界限，从而影响模型的效果。

以上几点体会同时也是我认为的我们的项目可改进的方向，如果不是受限于硬件条件、数据条件以及时间等客观因素的话，我认为我们的项目应该能够达到一个非常良好的应用效果。

六、References

[1]. 裴向杰, 唐红昇, 陈鹏. 融合 YCbCr 肤色分割的不良图像检测算法研究. 计算机技术与发展, 2015, 25(12): 80－84,90.

[2]. 黄杰, 史啸. 一种基于人体裸露皮肤形状的不良图像过滤系统. 东南大学学报 (自然科学版), 2014, 44(6): 1111－1115.

[3]. Ding M, Fan GL. Multi-layer joint gait-pose manifold for human motion modeling. IEEE Transactions on Cybernetics, 2015, 45(11): 2413－2424. [doi: 10.1109/TCYB.2014.2373393]

[4]. Shao H, Yu TS, Xu MJ, et al. Image region duplication detection based on circular window expansion and phase correlation. Forensic Science International, 2012, 222(1－3):

71－82. [doi: 10.1016/j.forsciint.2012.05.002]

[5]. He KM, Zhang XY, Ren SQ, et al. Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV, USA. 2016.770－778. [doi:10.1109/CVPR.2016.90]