**人工智能与机器学习报告**

1. **口罩佩戴检测**

3220101111 洪晨辉

1. **实验介绍**
   1. 实验背景

本次实验力图建立一个目标检测的模型，可以识别图中的人是否佩戴了口罩。此实验在新冠疫情中编写，具有浓厚的时代特色。

* 1. 实验要求

1）建立深度学习模型，检测出图中的人是否佩戴了口罩，并将其尽可能调整到最佳状态。

2）学习经典的模型 MTCNN 和 MobileNet 的结构。

3）学习训练时的方法。

* 1. 实验环境

可以使用基于 Python 的 OpenCV 、PIL 库进行图像相关处理，使用 Numpy 库进行相关数值运算，使用 Pytorch 等深度学习框架训练模型。

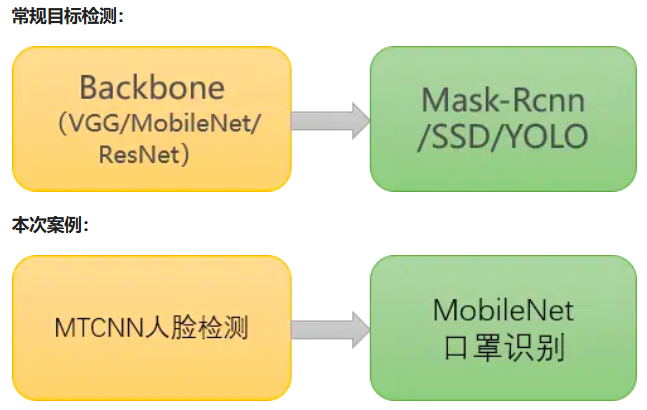
* 1. 实验思路

针对目标检测的任务，可以分为两个部分：目标识别和位置检测。

通常情况下，特征提取需要由特有的特征提取神经网络来完成，如 VGG、MobileNet、ResNet 等，这些特征提取网络往往被称为 Backbone 。而在 BackBone 后面接全连接层(FC)就可以执行分类任务。

但 FC 对目标的位置识别乏力。经过算法的发展，当前主要以特定的功能网络来代替 FC 的作用，如 Mask-Rcnn、SSD、YOLO 等。

我们选择充分使用已有的人脸检测的模型，再训练一个识别口罩的模型，从而提高训练的开支、增强模型的准确率。



1. **实验操作**
   1. 数据集

主要是一些拥有口罩和没有口罩的人的图片样本。一个典型的正负样本如下所示：





共计约600多个样本。

* 1. 制作数据集

数据集的预处理主要采用尺寸调整和图像增强的方法。代码如下：

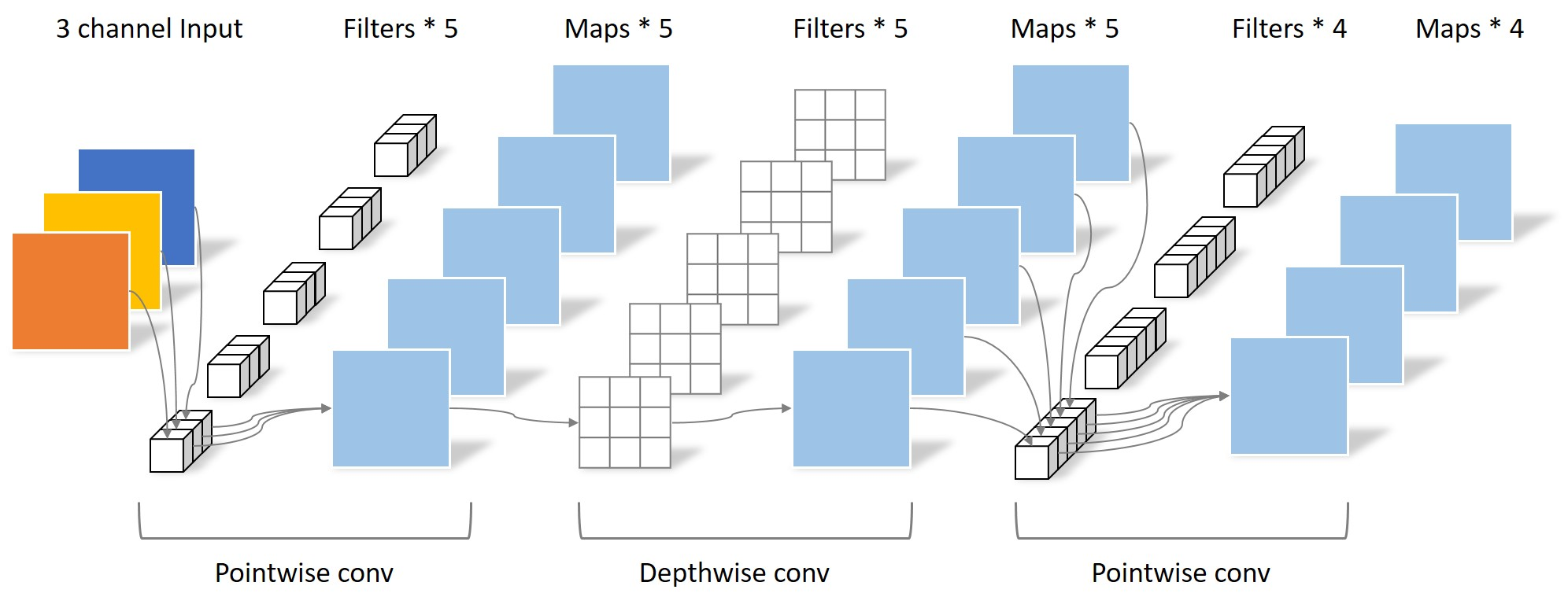
1. *# Training data generator*
2. 2 train\_data = ImageDataGenerator (
3. 3 rescale =1. / 255 , shear\_range =0.1 ,
4. zoom\_range =0.1 , width\_shift\_range =0.1 ,
5. 4 height\_shift\_range =0.1 , horizontal\_flip =
6. True , vertical\_flip = True ,
7. 5 validation\_split = test\_split
8. 6 )
9. 7 *# Testing data generator*
10. 8 test\_data = ImageDataGenerator ( rescale =1. /
11. 255 , validation\_split = test\_split )
    1. 模型选用

MTCNN仍然相同。但令人伤心的是，MobileNetV1的识别效果较差。因此，本报告选用的是微软更新后的MobileNetV2。值得注意的是，该模型也有一定历史了——其提出时间为2018年。

该模型仍然遵循经典的步骤，先进行模型预处理（已经存在ProcessingData.py中，未改），再进行模型训练。模型部分（存在MobileNetV2.py中）如下：

1. import torch
2. import torch.nn as nn
3. import torchvision
4. def Conv3x3BNReLU(in\_channels, out\_channels, stride, groups):
5. return nn.Sequential(
6. *# stride=2 wh减半，stride=1 wh不变*
7. nn.Conv2d(in\_channels=in\_channels, out\_channels=out\_channels,
8. kernel\_size=3, stride=stride, padding=1, groups=groups),
9. nn.BatchNorm2d(out\_channels),
10. nn.ReLU6(inplace=True)
11. )
12. *# PW卷积*
13. def Conv1x1BNReLU(in\_channels, out\_channels):
14. return nn.Sequential(
15. nn.Conv2d(in\_channels=in\_channels, out\_channels=out\_channels,
16. kernel\_size=1, stride=1),
17. nn.BatchNorm2d(out\_channels),
18. nn.ReLU6(inplace=True)
19. )
20. *# # PW卷积(Linear) 没有使用激活函数*
21. def Conv1x1BN(in\_channels, out\_channels):
22. return nn.Sequential(
23. nn.Conv2d(in\_channels=in\_channels, out\_channels=out\_channels,
24. kernel\_size=1, stride=1),
25. nn.BatchNorm2d(out\_channels)
26. )
27. class InvertedResidual(nn.Module):
28. *# t = expansion\_factor,也就是扩展因子，文章中取6*
29. def \_\_init\_\_(self, in\_channels, out\_channels, expansion\_factor, stride):
30. super(InvertedResidual, self).\_\_init\_\_()
31. self.stride = stride
32. self.in\_channels = in\_channels
33. self.out\_channels = out\_channels
34. mid\_channels = (in\_channels \* expansion\_factor)
35. *# print("expansion\_factor:", expansion\_factor)*
36. *# print("mid\_channels:",mid\_channels)*
37. *# 先1x1卷积升维，再1x1卷积降维*
38. self.bottleneck = nn.Sequential(
39. *# 升维操作: 扩充维度是 in\_channels \* expansion\_factor (6倍)*
40. Conv1x1BNReLU(in\_channels, mid\_channels),
41. *# DW卷积,降低参数量*
42. Conv3x3BNReLU(mid\_channels, mid\_channels,
43. stride, groups=mid\_channels),
44. *# 降维操作: 降维度 in\_channels \* expansion\_factor(6倍) 降维到指定 out\_channels 维度*
45. Conv1x1BN(mid\_channels, out\_channels)
46. )
47. *# 第一种: stride=1 才有shortcut 此方法让原本不相同的channels相同*
48. if self.stride == 1:
49. self.shortcut = Conv1x1BN(in\_channels, out\_channels)
50. *# 第二种: stride=1 切 in\_channels=out\_channels 才有 shortcut*
51. *# if self.stride == 1 and in\_channels == out\_channels:*
52. *#     self.shortcut = ()*
53. def forward(self, x):
54. out = self.bottleneck(x)
55. *# 第一种:*
56. out = (out + self.shortcut(x)) if self.stride == 1 else out
57. *# 第二种:*
58. *# out = (out + x) if self.stride == 1 and self.in\_channels == self.out\_channels else out*
59. return out
60. class MobileNetV2(nn.Module):
61. *# classes为分类个数, t为扩充因子*
62. def \_\_init\_\_(self, classes=2, t=6):
63. super(MobileNetV2, self).\_\_init\_\_()
64. *# 3 -> 32 groups=1 不是组卷积 单纯的卷积操作*
65. self.first\_conv = Conv3x3BNReLU(3, 32, 2, groups=1)
66. *# 32 -> 16 stride=1 wh不变*
67. self.layer1 = self.make\_layer(
68. in\_channels=32, out\_channels=16, stride=1, factor=1, block\_num=1)
69. *# 16 -> 24 stride=2 wh减半*
70. self.layer2 = self.make\_layer(
71. in\_channels=16, out\_channels=24, stride=2, factor=t, block\_num=2)
72. *# 24 -> 32 stride=2 wh减半*
73. self.layer3 = self.make\_layer(
74. in\_channels=24, out\_channels=32, stride=2, factor=t, block\_num=3)
75. *# 32 -> 64 stride=2 wh减半*
76. self.layer4 = self.make\_layer(
77. in\_channels=32, out\_channels=64, stride=2, factor=t, block\_num=4)
78. *# 64 -> 96 stride=1 wh不变*
79. self.layer5 = self.make\_layer(
80. in\_channels=64, out\_channels=96, stride=1, factor=t, block\_num=3)
81. *# 96 -> 160 stride=2 wh减半*
82. self.layer6 = self.make\_layer(
83. in\_channels=96, out\_channels=160, stride=2, factor=t, block\_num=3)
84. *# 160 -> 320 stride=1 wh不变*
85. self.layer7 = self.make\_layer(
86. in\_channels=160, out\_channels=320, stride=1, factor=t, block\_num=1)
87. *# 320 -> 1280 单纯的升维操作*
88. self.last\_conv = Conv1x1BNReLU(320, 1280)
89. self.avgpool = nn.AvgPool2d(kernel\_size=7, stride=1)
90. self.dropout = nn.Dropout(p=0.2)
91. self.linear = nn.Linear(in\_features=1280, out\_features=classes)
92. self.init\_params()
93. def make\_layer(self, in\_channels, out\_channels, stride, factor, block\_num):
94. layers = []
95. *# 与ResNet类似，每层Bottleneck单独处理，指定stride。此层外的stride均为1*
96. layers.append(InvertedResidual(
97. in\_channels, out\_channels, factor, stride))
98. *# 这些叠加层stride均为1，in\_channels = out\_channels, 其中 block\_num-1 为重复次数*
99. for i in range(1, block\_num):
100. layers.append(InvertedResidual(
101. out\_channels, out\_channels, factor, 1))
102. return nn.Sequential(\*layers)
103. *# 初始化权重操作*
104. def init\_params(self):
105. for m in self.modules():
106. if isinstance(m, nn.Conv2d):
107. nn.init.kaiming\_normal\_(m.weight)
108. nn.init.constant\_(m.bias, 0)
109. elif isinstance(m, nn.Linear) or isinstance(m, nn.BatchNorm2d):
110. nn.init.constant\_(m.weight, 1)
111. nn.init.constant\_(m.bias, 0)
112. def forward(self, x):
113. x = self.first\_conv(x)  *# torch.Size([1, 32, 112, 112])*
114. x = self.layer1(x)      *# torch.Size([1, 16, 112, 112])*
115. x = self.layer2(x)      *# torch.Size([1, 24, 56, 56])*
116. x = self.layer3(x)      *# torch.Size([1, 32, 28, 28])*
117. x = self.layer4(x)      *# torch.Size([1, 64, 14, 14])*
118. x = self.layer5(x)      *# torch.Size([1, 96, 14, 14])*
119. x = self.layer6(x)      *# torch.Size([1, 160, 7, 7])*
120. x = self.layer7(x)      *# torch.Size([1, 320, 7, 7])*
121. x = self.last\_conv(x)   *# torch.Size([1, 1280, 7, 7])*
122. x = self.avgpool(x)     *# torch.Size([1, 1280, 1, 1])*
123. x = x.view(x.size(0), -1)    *# torch.Size([1, 1280])*
124. x = self.dropout(x)
125. x = self.linear(x)      *# torch.Size([1, 5])*
126. return x

其结构图示如下：



训练得到的模型最终交由FaceRec2.py进行分类。

1. **结果分析**



模型已足够优秀，不必再升级至更新的MobileNetV3或是V4。

1. **总结与讨论**

当然，如果足够自信，本例还可以采用MobileNetV1。但是相对的，其结果并不是那么理想：

