



使用生成对抗网络合成咳嗽音频进行 COVID-19 检测

毕业设计报告

作者：Yahya Saleh

智能系统理学荣誉学士

2023年5月

目录

目录.i
图目录.iv
1 引言.1
1.1 项目背景.1
1.2 问题背景.1
1.3 研究依据.2
1.4 潜在益处.2
1.5 目标受众.2
1.6 范围与目标.3
1.6.1 目标.3
1.6.2 目标.3
1.7 交付成果.4
1.8 挑战的性质.4
1.9 调查报告概述.4
1.10 项目计划.5
2 文献综述.6
2.1 生成对抗网络（GAN）.6
2.2 通过咳嗽检测新冠肺炎.8
3 技术研究.10
3.1 编程语言.10
3.2 库.10
3.3 集成开发环境.11
3.4 所选算法.12
3.5 硬件.12
3.6 数据库.13
3.7 Web 框架.13
4 开发方法论.14

4.1 提出的深度学习开发方法论.14
4.2 深度学习开发方法的不足之处.15
4.3 深度学习的选定方法.16
5 反思.19
6 数据集.20
7 探索性数据分析.21
7.1 初始数据清理.22
7.2 数据分析.24
7.2.1 数据分布偏差.24
7.2.2 质量检查.25
7.3 专家标注.26
7.3.1 医生分布.28
7.3.2 咳嗽质量.28
7.3.3 咳嗽类型.29
7.3.4 咳嗽诊断和严重程度.29
7.3.5 专家状态标注与患者诊断.30
7.3.6 可听见的标注.32
8 音频预处理.33
8.1 音频数据预处理.33
8.2 用于数据增强的咳嗽分割.34
8.3 存储预处理后的咳嗽片段.37
9 特征提取.40
9.1 频谱图.40
9.1.1 短时傅里叶变换.40
9.1.2 梅尔频谱图.41
9.2 梅尔频率倒谱系数.42
9.3 色度特征.43
9.4 选定的特征表示算法.43
10 生成对抗网络实现.44
10.1 辅助分类器生成对抗网络（ACGAN）.44
10.2 判别器.44

10.2.1 结构.46
10.2.2 优化.46
10.3 生成器.48
10.3.1 类别标签.49
10.3.2 噪声.49
10.3.3 二维转置卷积.50
10.3.4 优化与损失.51
10.3.5 生成器架构.52
10.4 训练与评估.53
10.4.1 训练.58
10.4.2 验证.60
10.4.3 评估.61
11 系统实现.64
11.1 上传音频片段.64
11.2 查看已上传音频.65
11.3 音频样本分类.66
11.4 查看模型预测结果.67
参考文献.71
附录.错误！未定义书签。
项目提案草稿表（PPF）.错误！未定义书签。
项目规格说明表（PSF）.错误！未定义书签。
日志表.错误！未定义书签。
导师会议 1.错误！未定义书签。
导师会议 2.错误！未定义书签
导师会议3.错误！未定义书签
导师会议4.错误！未定义书签。
导师会议5.错误！未定义书签
导师会议6.错误！未定义书签。

图表目录

图2.1：原始生成对抗网络结构（van de Wolfshaar，2018年）.6
图2.2：图像到图像翻译（Zhu 等人，2017年）。给定两个域，生成对抗网络学习将图像从一个域转换到另一个域.7
图 3.1：所有开发者使用的首选集成开发环境（Stack Overflow 开发者调查，2021年）.11
图 3.2：生成逼真的人工人脸（Horev，2019年）.12
图4.1：流行方法的不足.16
图4.2：机器学习开发生命周期草案（Pinhasi，2021年）.17
图 7.1：加载 Coughvid 数据集的元数据.21
图 7.2：Coughvid 元数据列.21
图 7.3：评估元数据表中各列的空值百分比.22
图 7.4：将元数据中的缺失值填为“未知”.24
图 7.5：绘制 Coughvid 的数据分布.24
图 7.6：Coughvid 中患者状态、是否患有呼吸系统疾病、性别，以及是否发热和肌肉疼痛的数据分布.24
图 7.7：检测到咳嗽的随机变量分布.25
图 7.8：信噪比特征的取值分布.26
图 7.9：筛选检测到咳嗽概率不低于 0.5 且信噪比值不低于 5 的音频样本.26

图 7.10：筛选后的数据分布.26
图 7.11：按医生拆分函数，该函数用于移除 40 列专家标注，并将其转换为 11 列，其中 10 列表示医生的发现，1 列表示负责这些发现的医生.27

图 7.12：使用按医生拆分函数处理筛选后的数据集.28
图 7.13：右侧显示按医生统计的样本数量，左侧显示医生根据患者状态处理的样本.28

图 7.14：左图显示咳嗽性质的整体分布。右图显示咳嗽性质与患者状态的关系.29

图 7.15：左图：咳嗽类型的整体分布。右图：咳嗽类型相对于患者状态的分布.29

图 7.16：左图：咳嗽诊断。右图：咳嗽严重程度.30
图 7.17：医生根据患者病例的严重程度作出的诊断.30 图 7.18：左：医生对咳嗽的诊断。右：医生对咳嗽严重程度的评估.30

图 7.19：用于表示诊断与状态匹配的代码片段.31
图 7.20：诊断与状态匹配.31
图 7.21：数据集中可听到呼吸困难、喘鸣、窒息或充血的情况 .. 32 图 7.22：按咳嗽状态划分的数据集中可听到呼吸困难、喘鸣、窒息或充血的情况.32

图 8.1：从三个音频类别中分别随机选取的样本：健康、有症状和新冠肺炎.33

图 8.2：用于加载和预处理咳嗽音频样本的函数.33
图 8.3：用于生成咳嗽分割掩码以及分割后的咳嗽片段的函数.34

图 8.4：采样音频中的咳嗽掩码.36
图 8.5：图 8.4 所示音频样本生成的咳嗽片段.36
图 8.6：处理音频样本并生成该样本的咳嗽片段的代码片段.36

图8.7：从每个类别中选取相同数量的样本，其中所有COVID样本均被选中.37

图8.8：将预处理后的数据存储到 NumPy 文件中，以便快速处理和加载 ..... 38 图9.1：随机选取的音频样本.40
图9.2：图9.1 的短时傅里叶变换.40
图9.3：图9.1的梅尔频谱图.41
图9.4：图9.1的梅尔频谱图，带坐标轴和灰度缩放.41
图9.5：图9.1的梅尔频率倒谱系数.42
图9.6：图9.1的色度特征.43
图10.1：ACGAN架构。c 是类别标签，z 是随机噪声，Q 是判别器对类别标签的概率分布，y 是输入判别器的内容为真实数据的概率.44

图10.2：判别器架构代码.44
图10.3：判别器的优化器和损失选择.46
图10.4：判别器架构.47
图10.5：生成器架构代码.48
图 10.6：标签分支.49
图 10.7：噪声分支以及噪声分支与标签分支的拼接.49
图10.8：生成器的二维转置卷积.50
图 10.9：构建复合模型.51
图10.10：生成器架构.52
图 10.11：定义 ACGAN 的训练循环.53
图 10.12：训练判别器并记录真实和伪造数据的损失.58
图 10.13：训练生成器.59
图 10.14：验证生成器和判别器的性能.60
图 10.15：显示 ACGAN 的总损失并保存生成的样本 ..... 61 图 11.1：基于咳嗽的新冠肺炎检测平台首页.64
图 11.2：设置为仅接受音频文件的文件上传小组件.64
图 11.3：选择用于分类的音频文件.65
图 11.4：查看并播放上传的音频.65
图 11.5：预测咳嗽类别按钮.66

图 11.6：显示用户未上传音频片段的错误消息.66
图 11.7：加载旋转图标，表示模型正在预测输出.67
图 11.8：显示模型预测结果.67
图 0.1：原始生成对抗网络结构（van de Wolfshaar，2018年） ..... 错误！未定义书签 图 0.2：图像到图像翻译（Zhu 等人，2017年）。给定两个域，生成对抗网络学习将图像从一个域转换到另一个域.错误！未定义书签。

表格目录

表 6.1：Coughvid 中记录的咳嗽样本特征。这些特征构成元数据表的列.20

表 7.1：Coughvid 元数据的前 3 行.21
表 7.2：专家标注说明及可能范围。如前所述，以下各项特征在元数据表中均出现四次，每次后面都跟有负责评估的专家索引.22

表 7.3：元数据列中的空值百分比.23

1 引言

1.1 项目背景

医疗保健是当今影响力最大、最重要的行业领域之一，对人们的生活和健康至关重要。然而，在新冠疫情期间，行业的不足再次凸显：管理医疗保健的过时体系难以满足日益增长的医疗服务需求。

在人工智能领域，尤其是机器学习和深度学习领域，许多有望改变生活各个领域的发现已在交通运输（Khan，2022年；Mokayed，2021年；NG，2015）、文档与金融分析（Nikolaidou，2022年；Kanchi，2022年；Mokayed，2014年）、生物特征验证（Marzuki，2011）、自然语言（Alkhaled，2023年；T Adewumi，2022年）、智慧城市与安全（Mokayed，2022年；）、工业应用（Masri，2021年）、无人机与机器人技术（Mokayed，2021年）以及医疗保健（Voon，2022年）等领域取得进展。不同领域的许多此类有前景的发现，尤其是医疗保健领域的发现，仍停留在理论层面或未能取得理想成果，原因是医学数据集匮乏（S，2021年）。医疗数据的录制既困难又昂贵，因此医疗领域的数据十分稀缺，这降低了深度学习的效果，因为深度学习本质上需要大量数据，才能正确学习底层潜在特征并展现出可接受的性能。为了解决这一问题，一些研究人员开始转向深度生成模型进行图像合成，目标是构建生成对抗网络或类似模型，以生成属于该数据集领域的新合成医疗数据，从而有效扩大数据集规模。

同样地，本毕业项目旨在为医疗保健领域的数据合成相关已发表研究成果作出贡献。本项目的最终成果是一个经过训练的模型，该模型能够生成合成图像，用于扩充医疗数据集（Pierre，2021年）。本项目所选领域为新冠咳嗽录音，事实证明，这是一种可用于检测新冠肺炎的可行数据源。尽管该领域具有极其重要的意义，但由于可用于这一任务的数据集有限，它仍然是一个研究不足的领域。模型开发完成后，将分别使用扩充数据集和未扩充数据集训练最先进的模型，并通过测量性能差异来展示该模型的影响。最后，将模型嵌入 Web 应用程序中，以展示其能力，从而完成整个项目。为实现上述目标，将对利用生成模型进行图像合成的最新创新开展广泛的文献综述。

1.2 问题背景

- 医疗数据稀缺：医疗数据十分稀缺，阻碍了该领域的发展（Maier，2020年）。

- 医疗数据成本高昂：由于缺乏获取医疗数据的便捷途径，收集这些数据的成本很高（Pierre，2021年）。• 离散医疗数据：医疗保健领域的另一项顾虑是，如何保护患者信息的隐私，避免其落入对手之手。虽然已有一些方法可用于隐藏患者医疗数据中的信息，但这些算法可能遭到利用。借助生成对抗网络，可以创建无法追溯到患者的离散医疗数据（Pierre，2021年）。• 缺乏专家：图像合成的另一项用途是，在数据生成后，可弥补标注数据的专家人数有限这一问题，从而帮助训练用于监督学习任务的模型（Yi 等，2019年）。

1.3 研究依据

了解项目背景及其问题情境后，拟议解决方案背后的研究依据便显而易见。医疗保健是技术最应加以改进和革新的重要领域之一，但由于该领域的数据集匮乏，基于人工智能的解决方案在其中一直未能取得理想成效。医疗保健领域的数据获取十分困难，因为成本高昂，而且容易引发诸多隐私问题。具体来看 COVID-19 领域，它不仅面临同样的限制，阻碍基于人工智能的解决方案发展，而且 COVID-19 还是一种尚未被充分认识的新型病毒，因此可用数据更为有限。

1.4 潜在益处

项目初期对图像合成文献进行的批判性分析，将使研究人员更深入地了解图像合成领域当前取得的突破。此外，用于合成 COVID-19 咳嗽音频的已开发生成对抗网络模型，将通过考察模型为合成这些数据而学习到的潜在特征，提供宝贵的见解。此外，利用合成数据提升通过咳嗽进行 COVID-19 分类的性能，将改善 COVID-19 筛查流程。总的来说，该项目有望为人工智能和医疗保健领域带来显著益处。

1.5 目标受众

本项目的目标受众是为医疗保健构建基于人工智能的解决方案的人工智能开发者和生物医学工程师，以及能够利用所提出的改进取得新发现、推动创新的人工智能专家和开发者。此外，合成数据可以帮助人们深入了解新冠肺炎的特征，并帮助专家更好地理解这种病毒及其影响。

1.6 范围与目标

1.6.1 目的

构建一个可扩展的图像合成生成对抗网络，为新冠肺炎咳嗽音频领域生成合成医疗数据提供一种方法，以推动人工智能在医疗保健领域的发展，并激发各个领域的创新。

1.6.2 目标

本项目的目的可分解为以下各项目标，每个目标都有一组需要完成的子任务。

- 分析生成对抗网络和新冠检测领域的现有文献

需要对图像合成领域近期的创新成果进行文献综述，以了解用于相关任务的最新算法，从而确保针对所选任务构建性能更优的生成对抗网络结构。随后，对用于协助应对新冠疫情的最新模型和人工智能解决方案进行文献综述。最后，了解专家如何分析新冠咳嗽音频，以及此类音频中存在的特征。

- 实现生成对抗网络

探索生成对抗网络的强大能力及其生成合成数据的能力。随后，构建一个能够在通用数据集上合成新图像的生成对抗网络。这种通用方法有助于完善生成对抗网络的性能，无需担心数据集或领域对性能的影响，从而能够可靠地衡量模型的健康状况。构建的生成对抗网络结构将用于在所选领域内合成图像。在查看生成对抗网络的结果后，下一步是对模型进行去偏，以获得更具代表性的输出。

- 展示合成数据的影响

通过提升最先进的模型的性能，并评估利用扩展数据集所实现的改进程度，来展示该模型的影响。

- 将最终模型嵌入 Web 应用程序，以展示所提出的神经网络的强大性能。

完成工作并整理成文档后，将通过 Web 应用程序展示其影响力和强大性能。

1.7 交付成果

以下是该项目预期的交付成果：

- 分析有关生成对抗网络和新冠检测的现有文献• 实现一个用于咳嗽数据合成的生成对抗网络，以捕捉音频中新冠的潜在特征。• 提升模型通过咳嗽进行新冠检测的性能• 开发一个利用所提出流程的 Web 应用程序

1.8 挑战的性质

对于这个项目以及生成对抗网络而言，最大的挑战是确保判别器不会对数据过拟合，并确保生成的新数据具有差异性。

这个项目的主要局限在于，生成的模型将针对特定数据集进行构建；考虑到医疗保健领域任务的难度，这并不能保证模型架构具备可扩展性。尽管如此，该模型的结构很可能会启发新的可扩展架构，或启发能够适配医疗保健领域新领域的架构。

这个项目的主要学习目标是理解深度生成模型如何生成潜在特征，并利用这些特征构建属于同一领域、但略有差异的图像。理解这一过程的运作方式，以及将分布引入模型而非假设参数所带来的随机性，是构建生成对抗网络的关键。

1.9 调查报告概述

本调查报告旨在开展启动该项目所需的初步研究。本报告将通过各章节记录为准确评估项目的可行性及其影响而必须了解的技术和理论方面内容。

本报告分为 7 章。第一章为引言，阐述了本项目的动机、潜在收益以及项目目标。第二章对相关文献进行了详细分析，作者讨论了所选研究领域中的各种创新，以及所提出算法的应用。

本报告所研究的领域是新冠肺炎咳嗽音频记录；所综述的各种创新则属于医疗保健领域的图像合成方向。

第三章介绍技术研究，并记录所选用的工具。第四章记录项目所采用的方法论。第五章和第六章围绕需求验证展开；对于本项目而言，这意味着在领域专家的帮助下评估合成图像的准确率。最后，末尾部分总结所记录的发现，并对项目构想进行探讨，包括其可行性和影响。本报告的结论旨在确定这个项目是否值得继续投入努力。

1.10 项目计划

项目启动的第一步是获取领域和技术知识。2022年1月17th 至同月30th 将用于领域选择与理解，以及数据集选择和文档整理。随后，从2月 6th 到20th，将对生成对抗网络及其在数据集扩充方面的最新创新进行深入研究。

在掌握领域知识和技术知识后，下一步就是开始编程并构建模型架构。为了将流程拆分开来，专注于分别构建每个模块，首先要从头开始实现一个用于数据集扩充的生成对抗网络，并将其应用于通用数据集。这样一来，唯一需要关注的就是让模型正常运行，而无需担心领域的影响以及数据集的性质。当然，如前所述，还会测试该技术的有效性。整个任务将于2月中旬开始，持续至3月中旬。

生成对抗网络架构能够正常运行后，下一步就是在领域数据集上实现模型并衡量其性能。既然已经实现了可运行的生成对抗网络，人们或许会认为这项任务很简单，但它其实最为困难，因为模型需要消除偏差，而且在医疗保健领域合成数据与在通用领域合成数据大不相同。这项任务将历时一个月，从3月中旬持续到4月中旬。模型消除偏差并正常运行后，下一步就是衡量其对所选领域的影响并记录结果。由于期间会进行各种实验和测试，这项任务还将再持续一个月，直至5月中旬。

一旦基于所展示的工作完成一篇关于生成对抗网络对医疗保健领域数据集扩充影响的论文，下一步就是通过提出对生成对抗网络架构的改进方案，并在严格测试后记录发现，来进一步提升结果。此任务将持续一个月，直到6月中旬。

最后，待工作内容完成文档整理并准备发布后，我们将把最终模型嵌入 Web 应用程序，以展示其全部功能。这项任务将于6月中旬开始，持续至7月第一周。

<escape>利用生成对抗网络（GAN）合成咳嗽音频，用于 COVID-19 检测
</escape>

上述任务集遵循 DevOps 部署方法的管理结构，其中各项任务彼此解耦，并分别独立完成和集成。

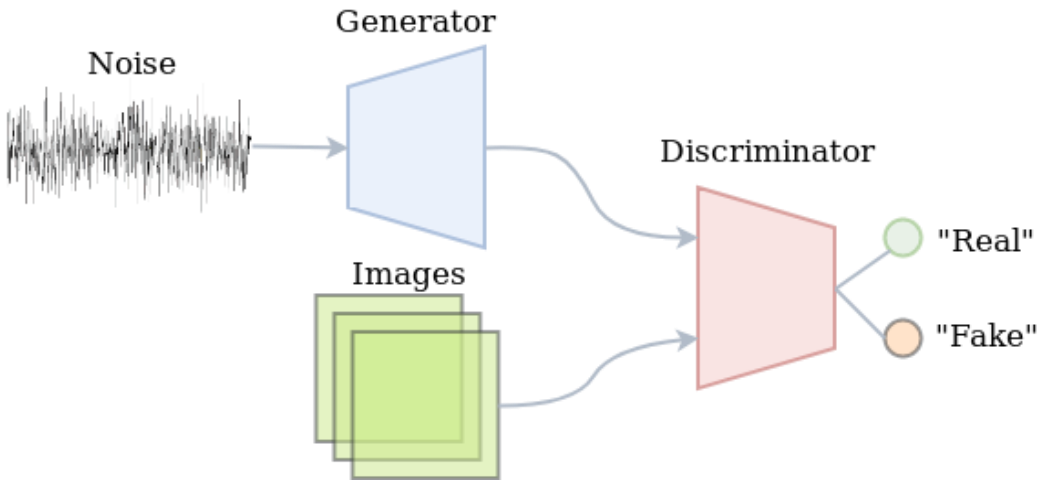
2 文献综述

2.1 生成对抗网络（GAN）

Goodfellow 等人（2014）首次定义的生成对抗网络是一组用于生成建模的两个神经网络，其中网络以隐式方式从数据分布中抽取样本。隐式抽取样本意味着，模型并非显式映射底层分布，而是通过训练来生成与数据分布相匹配的样本。这是通过使用两个神经网络实现的：生成器 G 和判别器 D。

如上图所示，生成器神经网络以从先验分布中采样得到的随机噪声 z 作为输入，该先验分布通常为高斯分布，记作 $p(z)$，并输出与真实样本 x_r 相似的数据 x_g。真实样本 x_r 则取自真实数据分布 $p_r(x)$ （ Yi 等人，2019 ）。以图像数据为例，生成器会接收随机噪声作为输入，并输出一张在视觉上与数据集中图像相似的图像。

图2.1：原始生成对抗网络结构（van de Wolfshaar，2018年）



另一方面，判别器接收来自所选领域的数据，例如图像。这些数据可能由生成器生成，也可能来自数据集本身。判别器的任务是确定给定图像是真实的（即属于数据集），还是虚假的。因此，判别器的输出是一个单一值，表示输入是真实还是虚假的概率。

这两个模型之间存在对抗关系：生成器试图欺骗判别器，而判别器则试图识别由生成器生成的数据。这种结构的独特性和强大之处在于，判别器能够指导生成器，使其不断改进生成数据的性质。

当梯度信息从判别器反向传播至生成器时（Yi 等，2019），便会发生上述过程。以下函数分别表示这两个模型的训练目标：

公式 1：生成器和判别器模型的训练目标

$$\mathcal{L}_D^{GAN} = \max_D \mathbb{E}_{x_r \sim p_r(x)}[\log D(x_r)] + \mathbb{E}_{x_g \sim p_g(x)}[\log(1 - D(x_g))]$$

$$\mathcal{L}_G^{GAN} = \min_G \mathbb{E}_{x_g \sim p_g(x)}[\log(1 - D(x_g))]$$

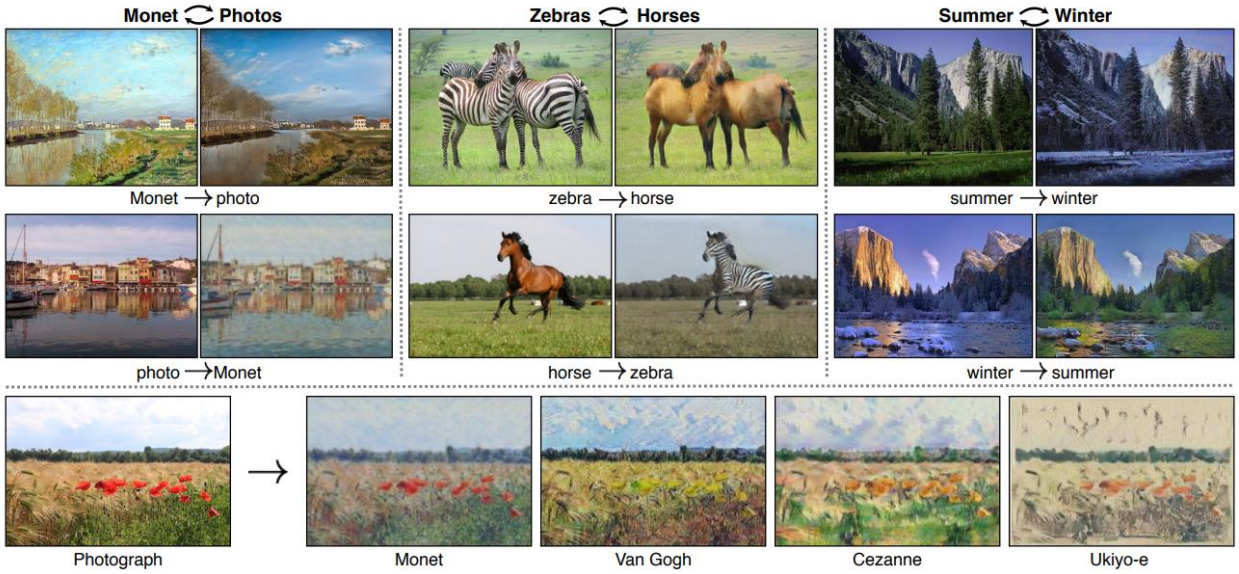
该公式表明，判别器会最大化输入数据集中的数据时返回的概率，即 $\log D(x_r)$；而在输入生成图像时，则会最小化返回的概率，这相当于最大化 1 与输入生成图像时返回的概率之间的差值，即 $\log(1 - D(x_g))$。这使判别器成为一个采用最大似然目标的二分类器。另一方面，生成器会最小化其生成的数据为假的概率，即 $\log(1 - D(x_g))$。

Goodfellow 等人（2014）指出，如果在生成器更新权重之前对判别器进行最优训练，那么最小化生成器的损失函数 L_G^{GAN} 将会缩小生成数据分布 $p_g(x)$ 与真实样本分布 $p_r(x)$ 之间的 Jensen-Shannon散度。

尽管 Yadav 等人（2018）指出，最优训练生成对抗网络模型面临诸多挑战，但该算法的强大能力和潜力使其成为图像合成领域的核心。许多研究提出了生成对抗网络架构的变体，以增强其鲁棒性并拓展其潜力。例如，人们提出了不同的判别器损失函数，以稳定训练过程，因为训练生成对抗网络时，处理判别器是最大的挑战之一。其中最值得关注的是 Wasserstein距离，因为它最为常用（Arjovsky 等人，2017年；Gulrajani 等人，2017年）。

其他作者对生成器进行了改进，使生成对抗网络能够执行图像转换等新任务。实现这一能力的方法是向生成器提供额外信息 c，以引导其生成具有一系列所需属性的输出 $x_g = G(z, c)$。这种生成对抗网络称为条件生成对抗网络（Mirza & Osindero, 2014）。这种改进的一种常见应用是前文提到的图像到图像翻译，其首个框架由 Isola 等人（2016）提出。下图展示了 Zhu 等人（2017）的论文中的一些图像到图像翻译示例；该论文提出了循环生成对抗网络的架构，这是一种非常流行且强大的变体：

图2.2：图像到图像翻译（Zhu 等人，2017年）。给定两个域，生成对抗网络学会将图像从一个域转换到另一个域。



2.2 通过咳嗽进行新冠肺炎检测

据我们所知，目前尚无已发表的论文研究用于 COVID-19 检测的咳嗽音频合成；然而，许多已发表的研究表明，这类研究确有必要。新冠疫情暴发后，许多研究人员探讨了机器学习在实现高效筛查方面能够发挥的作用。其中许多研究人员提出了基于机器学习的解决方案，通过一个人的咳嗽来检测病毒。

Darici 等人（2022年）的研究测试了多个模型，以利用咳嗽音频检测新冠肺炎。研究发现，受数据集有限的影响，模型可能存在偏差，并得出结论：需要更多数据来提升模型的性能。Lella和PJA（2021年）的另一项研究则将一维卷积神经网络（CNN）用于同一问题。不过值得注意的是，作者为这项研究收集了约五千个样本，其中仅有300个样本被标记为新冠肺炎阳性患者。

从这些发现可以清楚地看出，该领域的数据极为匮乏，许多实体都采取了行动来缓解这一局限。DiCOVA挑战赛于2021年首次举办（Muguli等人，2021年），并于2021年举办了第二届（Sharma 等，2021年），旨在鼓励研究人员分析感染新冠肺炎和未感染新冠肺炎者的录音。然而，与之前的数据集一样，比赛各赛道采用的数据集存在相当大的偏差：第一赛道的数据集仅有965个样本，其中只有172个新冠肺炎阳性样本；第二赛道有471个样本，其中只有71个新冠肺炎阳性样本（Sharma 等，2021年）。事实上，这种偏差也反映在参赛队伍的性能上。正如他们在论文中所述（Sharma 等，2022年），排名前四的队伍均未达到世界卫生组织提出的≥ 70%标准（特异度为95%）。尽管将这些队伍开发的模型所得分数简单地结合起来

便可达到既定目标，但作者仍指出：“未来仍有进一步发展的空间”（Sharma 等，2021年）。

通过音频为社区提供适用于 COVID-19 筛查的数据集的努力仍在继续，其中最引人注目的是 Orlandic 等人（2021）的工作。他们提出了 COUGHVID众包数据集，这是一个用于咳嗽分析的大规模数据集。该数据集包含 20,000 个样本，其中 1,010 个是“声称患有新冠肺炎”的人员样本（Orlandic 等，2021）。数据集规模的扩大无疑将提升模型的性能；然而，针对新冠肺炎患者的偏差仍然存在。事实证明，Darici 等人（2022）最先开展的研究使用了 COUGHVID 和 Coswara 数据集（Sharma 等人，2020），但仍报告了模型性能中的偏差。

3 技术研究

3.1 编程语言

深度学习可以使用各种编程语言，而为任务选择合适的编程语言对于确保开发和推进顺利进行至关重要。了解某一领域最佳编程语言的一个好方法，是听取该领域专家的意见。在机器学习和深度神经网络领域，Python 占据主导地位：57%的数据科学家和机器学习专家使用 Python 进行开发，33%的人将其列为首选（Nation，2019）。Python 之所以如此受欢迎，是因为它为机器学习开发者提供了诸多优势，包括丰富的库生态系统、平台独立性，以及庞大的社区和支持（Great Learning Team，2021）。

许多开发者在选择机器学习和深度学习语言时也会考虑 R；然而，事实证明，R 在这一领域并不十分流行，只有 31% 的开发者使用它，仅有 5% 的开发者将其列为首选（Nation，2019年）。虽然这可能会让一些开发者感到惊讶，但这些结果与该语言的设计目的十分吻合。R 是为具有数学背景的数据分析师和科学家设计的，而非为具有编程或计算机科学背景的人士设计的（r-project，未注明日期）。由此来看，R 不会针对深度学习或类似用途进行定制，也不太适合这些用途，这是合乎情理的，因为它们超出了 R 的设计用途范围。

对开发者而言，更合适的选择是 C/C++ ，但它仍远不及 Python。虽然 C++ 对于深度学习这类计算密集型任务似乎更具吸引力，但 Python 更容易学习，实际使用起来也更简单（Rohner 和 Netguru，2021年）。此外，机器学习领域的大多数已发表论文和研究成果都使用 Python 来实现模型（Rohner 和 Netguru，2021年），这使 Python 更具优势。

因此，本项目选择 Python 作为编程语言。Python 是一种非常流行的编程语言，拥有强大的深度学习支持和庞大的社区，可提供协助。使用 Python，开发者可以轻松实现并改进其他已有成果，因为文献中实现的模型很可能都是使用 Python 开发的。

3.2 库

如上一小节所述，Python 拥有丰富的库生态系统，使开发者能够在开发的各个阶段利用预先编写的代码。本项目将使用的主要库是 TensorFlow，这是一个功能强大的生成式深度学习库，它使用 Keras 库作为各种机器学习任务的 API端点。

在开发的各个阶段，也可能会使用其他库。例如，在对数据进行预处理时，可以使用 OpenCV，然后将预处理后的数据存储在

用于分析的 pandas 数据框。此外，与 Python 内置的列表数据类型相比，NumPy 库提供的 NumPy 数组以效率更高、性能更快而著称。

3.3 集成开发环境

本项目不选择集成开发环境，而是使用 Visual Studio Code（一款代码编辑器）。如下方所示，根据 2021 年 Stack Overflow 开发者调查（Stack Overflow，2021年），Visual Studio Code 是最受欢迎的代码编辑器，71% 的开发者将其选作主要代码编辑器。

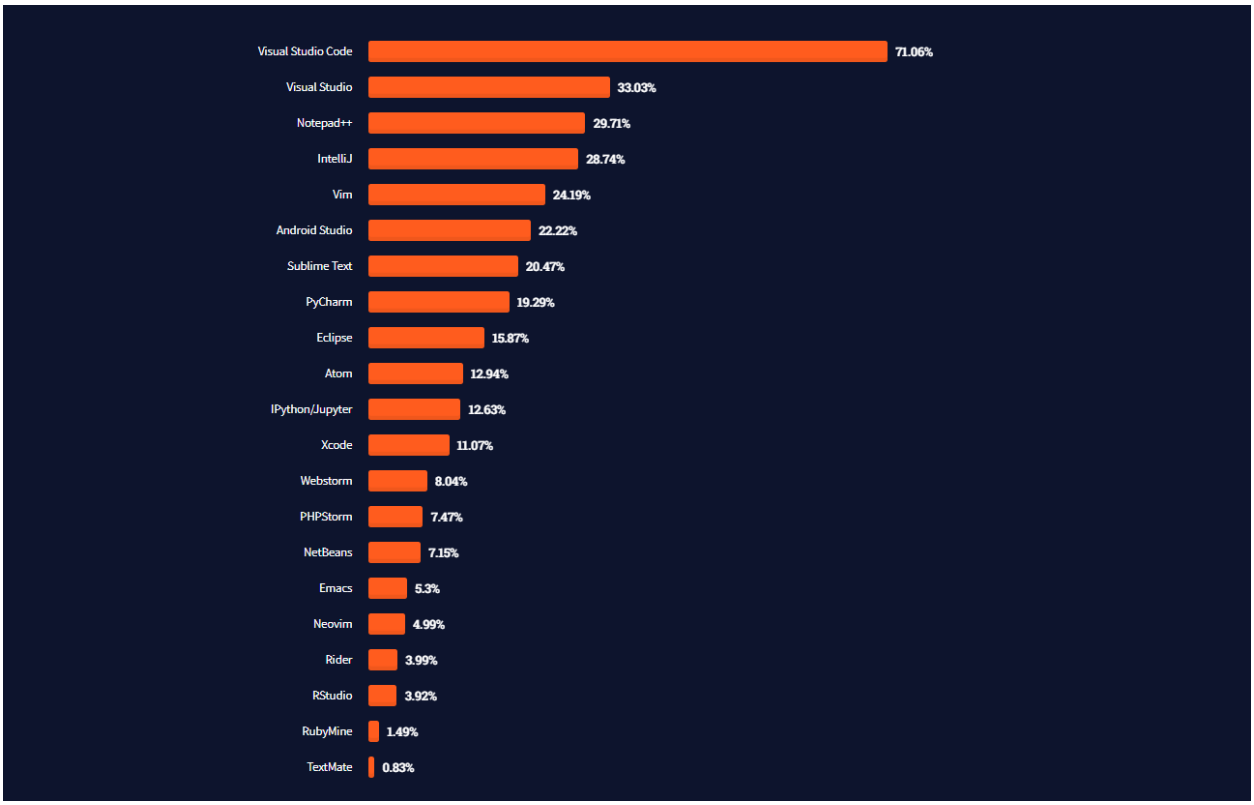


图 3.1：所有开发者首选 IDE 的 Stack Overflow 开发者调查（Stack Overflow，20 *21)*

VS Code 凭借轻量灵活的架构深受开发者欢迎，它将版本控制等功能集成在一个编辑器中。VS Code 拥有许多由社区开发并维护的扩展，借助代码片段、代码高亮和 IntelliSense 等功能，大幅提高了开发效率。Python 及其支持扩展是安装量最高的扩展，充分体现了 Python 在 VS Code 中的受欢迎程度。此外，VS Code 对 Web 开发的支持堪称最佳，因此非常适合用于完成项目，并进一步开发 Web 应用程序来展示成果。

3.4 所选算法

正如本调查报告中所讨论的，图像合成所选用的算法是生成对抗网络（GAN）。GAN 采用 Ian Goodfellow 首创的一种新颖算法，由两个模型组成，能够生成极为逼真、细节丰富的合成图像，使其看起来确实属于所选领域（Goodfellow 等人，2014）。GAN 算法从输入图像或视频中学习其不规则之处或模式，从而生成看起来像是来自该数据集的新图像或视频（Brownlee，2019）。例如，下图使用 GAN 模型，根据两张输入图像生成此前不存在的人工人脸，并将这两张图像分别标记为源图像和目标图像：



图 3.2：生成逼真的人工人脸（Horev，2019）

生成对抗网络通过训练两个深度神经网络来运作：一个生成模型和一个判别模型。前者使用输入的数据来训练，以生成新的样本；后者则经过训练，用于辨别并判断给定的数据是否来自该数据集。这两个模型会同时训练，直到判别模型错误地认为生成模型生成的数据中至少有一半是真实数据（Brownlee，2019）。

3.5 硬件

本项目的硬件要求经过专门调整，以满足深度学习所需的硬件规格，因为深度学习的计算需求极高，也是本项目的核心。首先要考虑的，也是最重要的因素，是图形处理器（GPU）。相比中央处理器，图形处理器更受欢迎，也更常用于深度学习，因为它们处理并行计算的能力更强。以下是适合深度学习的优质图形处理器的要求：

- 内存带宽• 核心数量• 处理能力• 显存容量

对于深度学习，至少配备 8 GB 显存的图形处理器较为理想；理想的选择包括 3080/3080ti/3090，这些显卡均属于英伟达。其近期推出的显卡支持人工智能功能，并兼容常见的深度学习框架（Ben-Zvi，2022年）。第二个需要考虑的因素是随机存取存储器（RAM）：RAM 越大，系统能够处理的数据量就越多（Zindi，2020）。通常，训练深度学习模型需要 8 GB RAM，但强烈建议配备 16 GB 或更高容量的 RAM。最后一个因素是中央处理器（CPU）。能够确保性能稳定的最低配置是英特尔酷睿 i7 处理器。处理能力相当或更高的处理器都足以满足需求（Zindi，2020）。

这些要求很高，普通计算机都无法满足。要满足这些要求，有几种选择：第一，使用 Google Colab 笔记本，通过谷歌的硬件进行计算。第二，向亚马逊租用工作空间，该空间提供一台虚拟机，可通过任何设备访问，并配备符合要求的硬件规格。

3.6 数据库

此项目不需要集成数据库；不过，如果 Web 应用程序需要存储数据以供日后使用，那么 Fire Cloud 将是首选数据库。它是一种免费的、高度灵活的云存储，采用 NoSQL 结构来存储数据。

3.7 Web 框架

本项目将使用 Flask 作为 Web 应用程序后端的 Web 框架，以展示项目成果。Flask 是基于 Python 的 Web 开发框架，与 Django 很相似，但在简单项目中颇受欢迎。这是因为与其他框架相比，它所需的配置少得多，也无需担心 Web 项目包含多个应用，因此使用起来简单方便。

由于 Flask 框架基于 Python，因此非常适合用于呈现这个以 Python 开发的项目，训练好的模型可以轻松集成其中，无需使用另一种语言编写模型处理代码。

最后，如果 Web 应用程序的前端需要任何编程逻辑，将使用原生 JavaScript 进行开发。

4 开发方法论

正如本调查报告中所述，该项目的主要目标是通过提出对生成对抗网络架构的改进，为生成建模和医疗保健领域作出贡献，以合成医疗数据、扩充医学数据集，并进一步推动人工智能在医疗保健行业的发展与融合。因此，该项目需要一种独特的开发方法论，更侧重于开发和优化生成模型，并衡量其影响。

4.1 深度学习的拟议开发方法论

围绕神经网络开发及其在软件中的集成，已有多项研究并提出了不同的开发方法论。一篇题为《组织内人工智能软件开发方法论》的研究论文（Rago，2018年）讨论了采用系统开发方法论的重要性，以便在系统中集成人工智能并将其作为嵌入式组件加以使用。所提出的方法论包含系统开发生命周期中已知的各个阶段，依次为：

1. 规划——“初始活动的目的是分析企业文化并进行高层次的流程分析”（Rago，2018年）；a. 能力管理 2. 设计——“接下来是流程模型的宏观层级定义任务。这样便可以进行差距分析，从而确定后续活动的指导方针。”（Rago，2018年）；3. 构建——“构建阶段包括实现软件，并从公司的大数据中开发知识。”；4. 运行——实施完整系统。

这种方法论更侧重于软件开发本身，并未将神经网络的集成视为主要组件。从提出的阶段来看，显然，神经网络方面被视为次要组件，可以作为系统生命周期的一部分来完成。

另一篇题为“神经网络软件工程方法”的研究论文（Senyard 等，2004）介绍了一些方法，帮助人工智能开发者摆脱临时拼凑的方法（即试错法），以有条理的方式构建模型，使结果可重复且稳定。该研究的作者将开发神经网络的常见方法定义为包含以下阶段：

1. 选择问题表示并获取相关数据集；2. 设定模型的性能预期；3. 训练神经网络，直至达到预期性能；4. 更新模型的参数以改进模型。

随后，他们提出应为开发流程引入一定的秩序：不应只定义一种问题表示并配以一个数据集，而应根据开发者的判断确定多种问题表示。然后，按照可靠性对每种问题表示及其对应的数据集进行排序，并分别用它们训练模型，直至达成目标。对于深度学习专家而言，本文对现有开发流程的描述以及提出的流程可能显得不妥且不正确。

4.2 深度学习开发方法的不足之处

大多数研究论文，包括前面讨论过的那些论文，在神经网络开发方面都未能切中要点。这可能是因为提出这些开发方法的开发人员具有软件工程背景，因此缺乏开发神经网络所需的详细知识和经验。神经网络始终处于不断变化和演进之中，而他们却试图将神经网络所要解决的问题归结为一个系统开发问题；这种做法即使从最宽容的角度来看，也并非一种有效的方法。

神经网络开发并非如《组织内人工智能软件开发方法论》（Rago，2018年）一文所暗示的那样，是一个与系统相互关联、可作为系统生命周期一部分来开发的组件。相反，它是一个独立组件，需要经过大量迭代和测试，才能部署可靠的模型。经过训练且完整的模型随后才会成为一个组件，可集成到系统中或用于商业用途。

另一个难以找到能够完美定义神经网络开发过程的方法论的原因是，神经网络，或者更具体地说深度学习，是一个庞大的领域，其中使用了各种模型和架构来实现不同的应用，例如图像分类、分辨率提升、自动驾驶汽车、图像合成、参考学习等。这些不同的应用会处理各种数据类型和记录，例如常规数值数据、图像数据、Sequential 数据和生成式数据。

然而，仍然需要一种有条理的开发方法论，以加快开发进程并为深度学习提供路线图。许多机器学习专家会指出这一问题，并强调开发任何人工智能模型都需要结构化方法论（Pinhasi，2021年）；不过，由于他们拥有人工智能开发经验，他们也会指出，即便是最敏捷的方法论，在用于机器学习开发时仍有不足之处。以下总结了一些常见的开发方法论，以及它们为何并不完全适用于机器学习，更不用说深度学习了：

方法论	结论	原因
敏捷	无法直接照搬使用	敏捷方法 白小姐 实践 were 已开发 for 工程方面，并且对人工智能的泛化能力也不够好 (Pinhasi，2021年)

科学研究 方法论	Not a 完整 解决方案	优惠彩票开户 有用的技术来处理 with 存在不确定性，但该方法论并非如此 为商业发布而设计（Pinhasi, 2021年)

图4.1：流行方法的不足

4.3 深度学习的选定方法

如图所示，机器学习问题通常介于研究与部署之间，因为要选择合适的算法并加以改进，需要深入理解问题。同时，人工智能用于解决关键问题，因此开发过程必须稳健可靠。由于深度学习可用于解决各种各样的问题，因此没有一种具体的开发方法论可以直接套用。相反，专家会依靠自己的判断制定计划，并很可能随着模型结果和发现逐渐明朗而对计划进行调整。

尽管如此，机器学习专家仍在尝试提出一种通用的机器学习工作流程，如下方流程图所示。然而，这些工作流程可能并不适用于深度学习，因为深度学习利用神经网络来实现各种技术，其阶段和问题结构都截然不同。

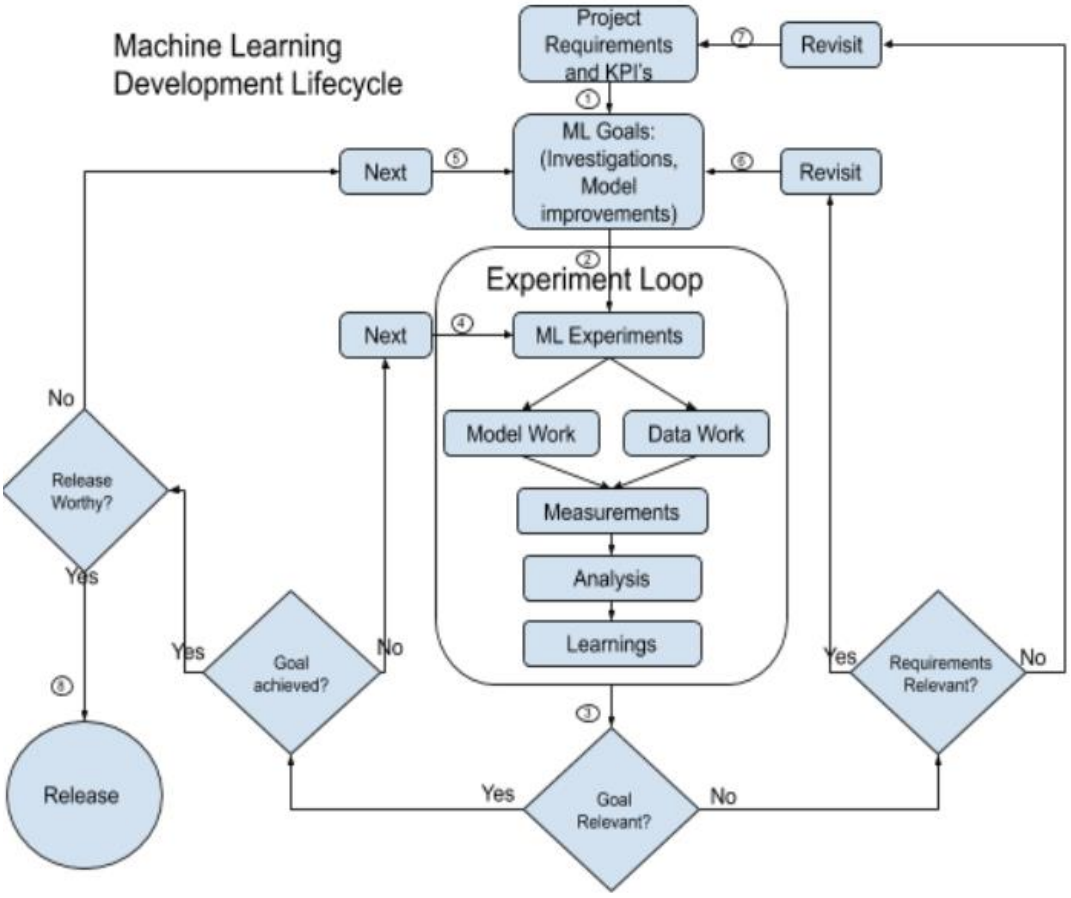


图 4.2：机器学习开发生命周期草案（Pinhasi，2021年）

因此，该系统的开发方法论将采用一种定制方法论，遵循一套简单的阶段，这些阶段与机器学习开发的阶段类似。各阶段如项目计划所示。

- 领域与数据集选择

此阶段涉及在医疗保健领域中选择最佳的关注领域（目前为新冠肺炎胸部X光片），并选择适合合成的数据。

- 数据集预处理与分析

此阶段的工作包括加载数据、对其进行预处理，并了解数据的潜在特征，这将有助于开发人员评估训练所得生成对抗网络所生成合成数据的性质。

- 文献综述

此阶段将进行批判性分析，以了解生成对抗网络领域的最新创新，以及这些创新和改进如何应用于医疗保健领域。这项批判性综述将为开发生成对抗网络以及了解特定领域中的不足提供有价值的见解。

- 构建通用生成对抗网络

此阶段旨在了解生成对抗网络模型背后的算法，以便更好地改进所提出的方案，并找到最适合当前任务的模型调整方式。

- 构建图像数据合成模型

在这个过程中，各项内容被整合起来，对模型进行训练，从而为指定领域生成合成数据。

- 模型评估

在这一阶段，开发人员会严格评估模型的性能，确保合成了恰当的特征，并且生成的数据具有实用价值。这是最为严谨的过程，期间会对模型进行各种改进，并请专家审查结果。

- 用于数据集扩充的模型

随后将使用该模型扩充给定的数据集，并在最擅长识别和利用数据潜在特征的最先进模型上，对比评估扩充前后的模型性能。

- 模型部署

模型完成后，最后一步是将模型部署用于商业用途，并构建前文提到的 Web 应用程序，以展示模型的能力和效能。

5 反思

总体而言，使用 GAN 合成咳嗽音频以检测 COVID-19 是一项极具前景、尚待探索的研究。这个想法蕴藏着巨大的潜力，因此需要投入大量工作，但最终成果必定会非常出色。

在本项目调查阶段结束时，研究构想和研究主题已经敲定，选定了合适的数据集，并针对该构想的技术和领域层面开展了充分研究。随着开发者进一步探索这一构想及其面临的挑战，计划仍有调整和改进的空间，但总体而言，计划已经确定，将继续推进。

6 数据集

用于咳嗽数据增强的所选数据集是 Coughvid 数据集（Orlandic 等，2021），这是一个大型新冠肺炎音频数据集。Coughvid 数据集包含来自不同患者的 25,000 个咳嗽样本，这些患者的年龄、性别和地理位置各不相同，其中 1,155 个样本来自新冠肺炎阳性患者。此外，作为额外的验证环节，作者请四位专家评估了部分数据集。专家共评估了 2,800 个样本。表 6.1 定义了每个咳嗽样本的记录特征，如（Orlandic 等，2021）所示。

表 6.1：记　　录的 Coughvid 咳嗽样本特征。这些特征构成元数据表的各列　　.

Name	必填	可能取值范围 值	描述
uuid	Yes	字符串值	每个咳嗽样本的唯一标识符
日期时间	Yes	UTC 日期和 ISO 8601 时间 格式	接收时间戳 录音。
咳嗽_已检测到	Yes	浮点数 在区间 [0, 1] 内	概率　　that　　the　　录音 包含咳嗽声，根据 自动检测算法 如方法部分所述。
SNR	Yes	浮点数位于 区间 [0, ∞)	咳嗽录音的信噪比估计值。 咳嗽录音的比例。
纬度	No	浮点数 值	自我报告的纬度地理定位 以降低的精度进行坐标计算。
经度	No	浮点数 值	自报的地理定位经度 坐标，精度已降低。
age	No	整数值	自报年龄值。
性别	No	{女性, 男性, 其他}	自报性别。
呼吸系统_状况	No	{真, 假}	患者有其他呼吸系统 （自行报告的）状况。
发热_肌肉_疼痛	No	{真, 假}	患者有发热或肌肉 疼痛（自行报告）。
状态	No	{新冠肺炎, 有症状. 健康}	患者自述已 被诊断为新冠肺炎 （新冠肺炎），有症状，但 无诊断（有症状），或该 是健康（健康）。
专家_标签_{1,2,3,4}	否	JSON 字典 使用手动方式 标签来自 专家 1、2、3 或 4	专家标注变量如下 如表 4 所述。

7 探索性数据分析

本节分析音频样本的元数据和结构，以验证其分布并探索其中的模式。最终，该项目旨在通过咳嗽来辅助和改进新冠肺炎检测。因此，探索性数据分析的目标是了解数据集中的趋势并识别潜在偏差。系统加载元数据，并显示数据的前 3 行，如图 7.1 所示，以便检查元数据；相关内容如表 7.1 所示。

图 7.1：加载 Coughvid 数据集的元数据

```
# 加载元数据信息data_raw = pd.read_csv(os.path.join(ROOT, "
```

表 7.1：Coughvid 元数据的前 3 行

uuid	datetime	cough_detected	SNR	latitude	longitude	age	gender	respiratory_condition	fever_muscle_pain	...	quality_4
00039425-7f3a-42aa-ac13-834aaa2b6b92	2020-04-13T21:30:59.801831+00:00	0.9609	16.151433	31.3	34.8	15.0	male	False	False	...	NaN
0007c6f1-5441-40e6-9aaf-a761d8f2da3b	2020-10-18T15:38:38.205870+00:00	0.1643	16.217201	NaN	NaN	46.0	female	False	False	...	NaN
0009eb28-d8be-4dc1-92bb-907e53bc5c7a	2020-04-12T04:02:18.159383+00:00	0.9301	20.146058	40.0	-75.1	34.0	male	True	False	...	NaN

ws × 51 columns

元数据包含 51 列，由于列数较多，难以全部纳入表格。图 6.2 展示了元数据的所有列。

图 7.2：Coughvid 元数据列

```
索引(['通用唯一识别码',    '日期时间'.    '检测到咳嗽_'          '信噪比',    '纬度'.          '经度'.
'年龄'.    '性别'.          '呼吸系统_状况'.                '发热_肌肉_疼痛'.          '状态'.
'性质_1', '咳嗽_类型_1', '呼吸困难_1', '喘鸣_1', '喉鸣_1', '窒息_1',
'充血_1'.                '无_1'.              '诊断_1'.              '严重程度_1'.        '性质_2'.
'咳嗽_类型_2'.          '呼吸困难_2'.        '喘鸣_2'.              '喉鸣_2'.            '窒息_2'.
'充血_2'.                '无_2'.              '诊断_2'.              '严重程度_2'.        '性质_3'.
'咳嗽_类型_3'.          '呼吸困难_3'.        '喘鸣_3'.              '喉鸣_3'.            '窒息_3'.
'充血_3'.                无_3                 '诊断_3'.              严重程度_3           '性质_4'.
'咳嗽_类型_4'.          '呼吸困难_4'.        '喘鸣_4'.              '喉鸣_4'.            '窒息_4'.
'充血_4', '无_4', '诊断_4', '严重程度_4'],  dtype='对象')
```

检查各列可以发现，这些列有所区别：前几列是与音频相关的元数据，例如其标识符、日期时间、状况等。这些列及其意义汇总于表 5.1。接下来是一组重复列，每列后面都带有一个 1 到 4 之间的数字，例如诊断这一列在各列中出现了四次'诊断_1'、'诊断_2'、'诊断_3'和 '诊断_4'。这些

列代表四位专家各自对所记录特征的反馈之一。例如，`诊断_1'` 和 `'严重程度_1'` 分别表示[redacted]的诊断和[redacted]这些列中的某一行若为空值，意味着相应专家未被分配该样本。表 7.2 汇总并描述了专家记录的各项特征。根据数据集文档，四位专家各自负责检查一组独特的音频样本；然而，四位专家并未覆盖整个数据集。因此，在修改元数据结构以更好地呈现专家反馈之前，先对与专家无关的特征或列进行探索性数据分析。

表 7.2：专家标注说明及可能取值范围。如前所述，以下每项特征在元数据表中均出现四次，每次后面都跟有负责评估的专家的索引。

Name	可能取值范围	说明
性质	{good, ok, poor, no_咳嗽}	录制的咳嗽声的性质。
咳嗽_类型	{湿咳, 干咳, 未知}	咳嗽类型。
呼吸困难	{真, 假}	可听见的呼吸困难。
喘鸣	{真, 假}	可听见的喘鸣。
喉鸣	{真, 假}	可听见的喉鸣。
窒息	{真, 假}	可听见的窒息。
充血	{真, 假}	可听见的鼻塞。
无	{真, 假}	没有听到任何明显的声音。
诊断	{upper_感染, lower_感染, 阻塞性_疾病, 新冠肺炎, 健康_咳嗽}	专家对病情的印象的患者。它可能是上呼吸道或下呼吸道感染、阻塞性疾病（哮喘、慢性阻塞性肺疾病，等）、新冠肺炎或健康咳嗽。
严重程度	{假咳嗽，轻度，重度，未知}	专家对严重程度的印象咳嗽。它可能是来自健康患者的假咳嗽，也可能是患病患者的轻度或重度咳嗽，如果专家无法判断，则为未知。

7.1 初始数据清理

探索性数据分析的第一步是导出每列空值的百分比。这有助于对数据进行有依据的处理，并建立一组预期值。图 7.3 用于生成表 7.3。

图 7.3：评估元数据表中各列的空值百分比

```
# 检查列级别的空值百分比 # 检查每列的数据类型 temp_df = pd.DataFrame((data_
```

表 7.3：元数据列的空值百分比

	空值百分比	
uuid	0.000000	对象
日期时间	0.000000	对象
咳嗽_检测到	0.000000	64 位浮点数
SNR	0.000000	64 位浮点数
纬度	41.620385	64 位浮点数
经度	41.620385	float64
age	44.760245	float64
性别	41.108570	对象
呼吸系统_状况	41.108570	对象
发热_肌肉_疼痛	41.108570	对象
状态	41.108570	对象
性质_1	97.023485	对象
咳嗽_类型_1	97.088824	对象
呼吸困难_1	97.023485	对象
喘鸣_1	97.023485	对象
喉鸣 1_	97.023485	对象
窒息_1	97.023485	对象
充血_1	97.023485	对象
无_1	97.023485	对象
诊断_1	97.088824	对象
严重程度_1	97.088824	对象
性质_2	97.023485	对象
咳嗽_类型_2	97.092453	对象
呼吸困难_2	97.023485	对象
喘鸣_2	97.023485	对象
喉鸣 2_	97.023485	对象
窒息_2	97.023485	对象
充血_2	97.023485	对象
无_2	97.023485	对象
诊断_2	97.088824	对象
严重程度_2	97.088824	对象
性质_3	97.023485	对象
咳嗽_类型_3	97.114233	对象
呼吸困难_3	97.023485	对象
喘鸣_3	97.023485	对象
喉鸣 3_	97.023485	对象
窒息_3	97.023485	对象
充血_3	97.023485	对象
无_3	97.023485	对象
诊断_3	97.125123	对象
严重程度_3	97.114233	对象
性质_4	97.023485	对象
咳嗽_类型_4	97.092453	对象
呼吸困难_4	97.023485	对象
喘鸣_4	97.023485	对象
喉鸣 4_	97.023485	对象

窒息_4	97.023485	对象
充血_4	97.023485	对象
无_4	97.023485	对象
诊断_4	97.136012	对象
严重程度_4	97.099713	对象

检查表 7.3 后发现，只有通用唯一识别码、日期时间、咳嗽_检测和信噪比没有空值。这些列在分析中是可靠的。此外，其余非专家特征——纬度、经度、年龄、性别、呼吸系统_状况、发热_肌肉_疼痛和状态——约有 41% 的空值，可能会影响评估和发现；不过，由于探索性数据分析的目标是了解数据并发现潜在偏差，这一损失比例可以接受。空值将替换为“未知”，以便在使用图 7.4 所示代码片段生成的图表中呈现。最后，专家特征的空值百分比为 97%，这符合预期，因为每位专家评估的样本仅占全部 25,000 个样本中的 2,800 个。总的来说，专家评估了数据集的 12%，这意味着他们的评估结果无法推广；若要使用这些结果，则必须大幅缩减数据集。

图 7.4：使用“未知”填充元数据中的缺失值

```
# 处理缺失值数据_完整 = 数据_原始。填充
```

7.2 数据分析

数据清理完成后，下一步是探索数据分布，并识别咳嗽中可能存在的偏差。探索性数据分析的发现将指导流程的选择和预处理阶段。

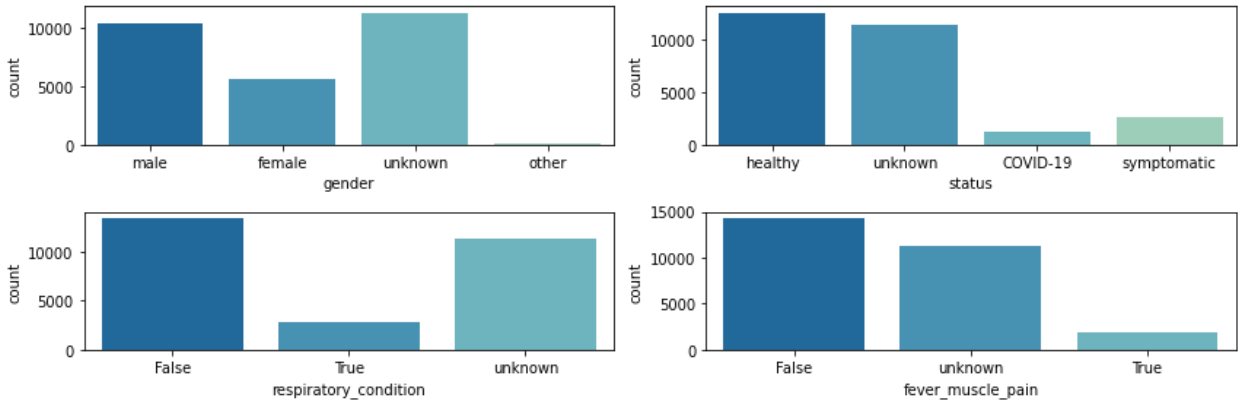
7.2.1 数据分布偏差

图 7.5 绘制了数据分布，涉及患者状态、是否存在呼吸系统疾病、性别，以及是否发热或存在肌肉疼痛。

图 7.5：绘制 Coughvid 的数据分布

```
f, axs = 2, 2, = 12, 4 _ = _ [4,
```

图 7.6：Coughvid 中患者状态、呼吸系统疾病、性别以及发热和肌肉疼痛情况的数据分布

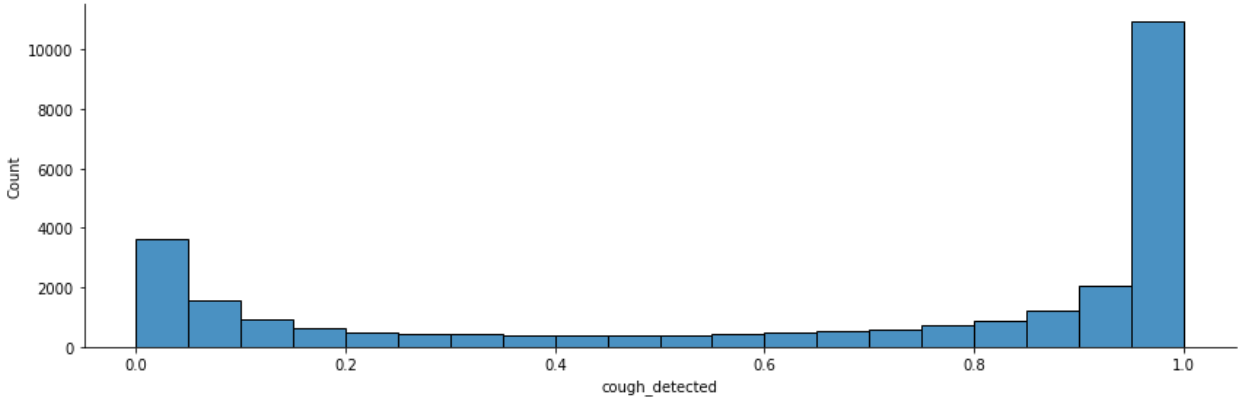


数据分布总体上显示，许多参与者选择不填写许多可选字段，这是开放式自愿参与的结果。查看第一张图可以发现明显的男性偏差，男性参与者人数是女性的两倍。第二张图显示了患者的状态分布，凸显了 COVID 样本的稀缺：只有约 1000 个样本，而其他状态的样本数量约有数万。呼吸系统状况显示，大多数参与者没有呼吸系统疾病。同样，大多数参与者没有发热或肌肉疼痛。

7.2.2 质量检查

通过检查两列来评估各个音频样本的性质：检测到咳嗽_和信噪比。前者表示音频样本包含咳嗽的概率；检查该特征的分布后发现，非咳嗽音频很可能存在，如图 7.7 所示。

图 7.7：检测到咳嗽的随机变量分布



因此，如图 7.9 所示，只有检测到咳嗽值达到 0.5 或更高的音频样本才会被选用于此任务，以减少无咳嗽音频的影响。信噪比（Signal to Noise Ratio）是对咳嗽录音信噪比的估算。图 7.8 展示了信噪比特征的数值分布。

图 7.8：信噪比特征的数值分布

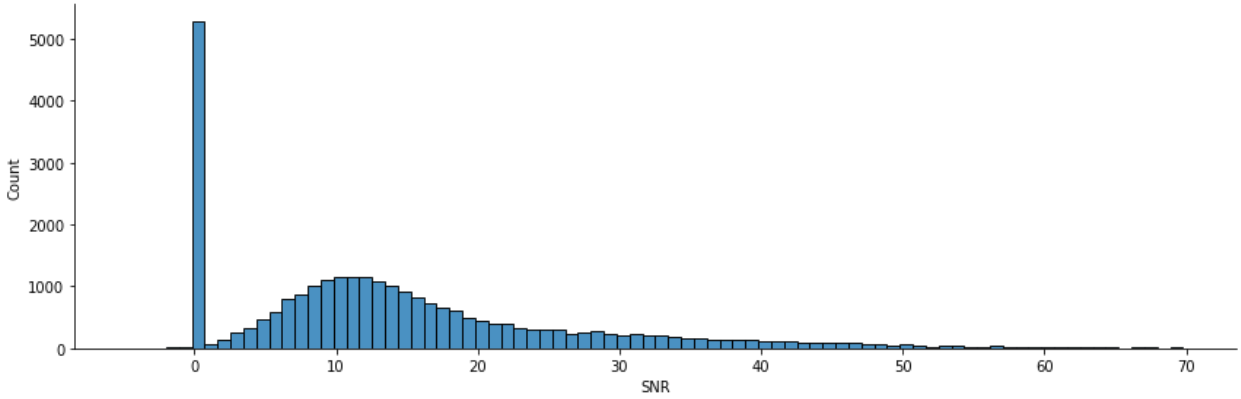


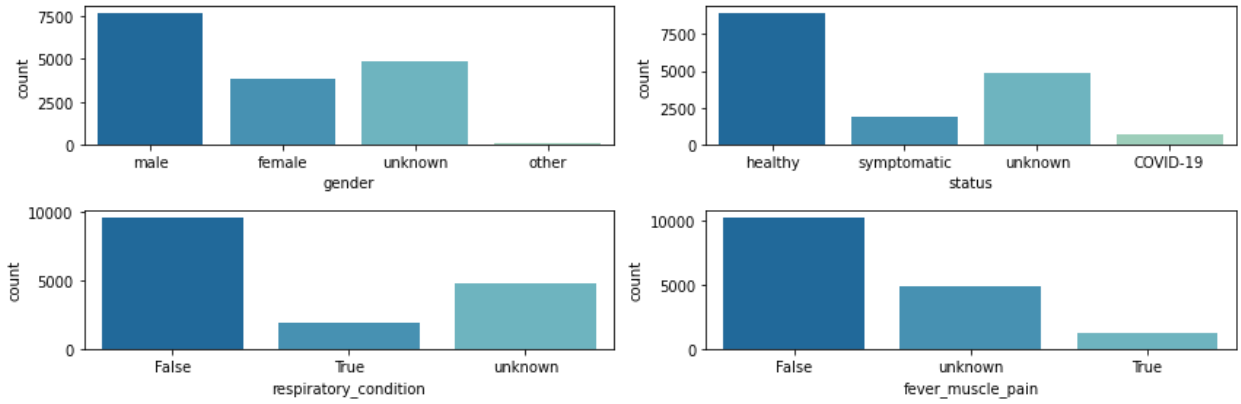
图 7.8 显示，低信噪比的数值出现了一个巨大的峰值，随后是合理分布的信噪比数值。为确保噪声不会破坏训练结果，只选择信噪比大于 5 的样本，如图 7.9 所示。

图 7.9：　选择检测到的咳嗽概率为 0.5 或更高且信噪比为 5 或更高的音频样本

```
已筛选 _ = _ [ _ [ _ ] > 0 5 _ [ ] 数据 完整数据(完整数据"
```

过滤数据后的分布如图 7.10 所示。

图 7.10：过滤后的数据分布



7.3 专家标注

虽然仅为 2,800 个样本提供了专家标注，但仍值得检查这些发现，以了解这些样本。图 7.11 中定义的函数旨在接收一个数据框，删除 40 列（其中每 10 列代表表 7.2 所示的一位专家的发现），并将这 40 列汇总为 10 列，同时添加一个医师列，用于标识

负责这些发现的专家。随后，图 7.12 使用该函数处理筛选后的数据。

图 7.11：按医生拆分，用于移除 40 列专家标注并将其转换为 11 列的函数，其中 10 列表示医生的发现，1 列表示负责这些发现的医生。

```
def split_by_医生(df):
    列_名称 = [
        通用唯一识别码 ,
        "日期时间" ,
        "咳嗽 已检测到",
        "信噪比" ,
        "纬度"     ,
        "经度",
        "age",
        "性别",
        "呼吸系统_状况",
        "发热 _m肌肉_疼痛 in",
        "状态"   ,
        "性质",
        "咳嗽_类型",
        "呼吸困难",
        "wheezing",
        "喉鸣"    ,
        "窄息",
        "充血",
        "无",
        "诊断",
        "严重程度",
    ]
    医生_01 = df.iloc[:, 0:21]
    医生_01 =
医生_01[医生_01.性质 1.notna()].reset_index(drop=真)
    医生_01.columns = 列_names
    医生_01["医生"] = "P01"

    医生_02 = pd.concat([df.iloc[:, 0:11], df.iloc[:, 21:31]], axis=1)
    医生_02 =
医生_02[医生_02.性质 2.notna()].reset_index(drop=真)
    医生_02.columns = 列_名称
    医生_02["医生"] = "P02"

    医生_03 = pd.concat([df.iloc[:, 0:11], df.iloc[:, 31:41]], axis=1)
    医生_03 =
医生_03[医生_03.性质 3.notna()].重置_索引(删除=真)
    医生_03.列 = 列_名称
```

```
    医生_03["医生"] = "P03"

    医生_04 = pd.concat([df.iloc[:, 0:11], df.iloc[:, 41:51]], axis=1)

    医生_04 =
医生_04[医生_04.性质_4.notna()].reset_index(drop=真)
    医生_04.columns = 列_名称
    医生_04["医生"] = "P04"
    返回 医生_01, 医生_02, 医生_03, 医生_04
```

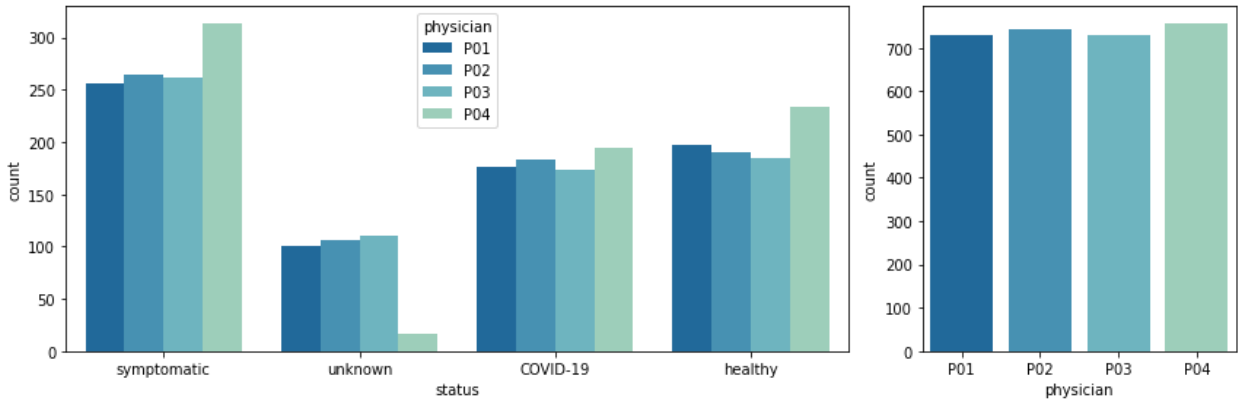
图 7.12：使用按医生拆分函数处理筛选后的数据集

```
医生_01, 医生_02, 医生_03, 医生_04 = 拆分_依据_医生(筛选后的_数据) 已标注_的df = pd.concat( [医生_01, 医
```

7.3.1 医生分布

下图显示了按状态划分的医生分布。总体而言，四位医生分配到的样本数量相同，但状态各不相同。医生处理的未知状态数量很少，其中第四位医生的数量明显更少。

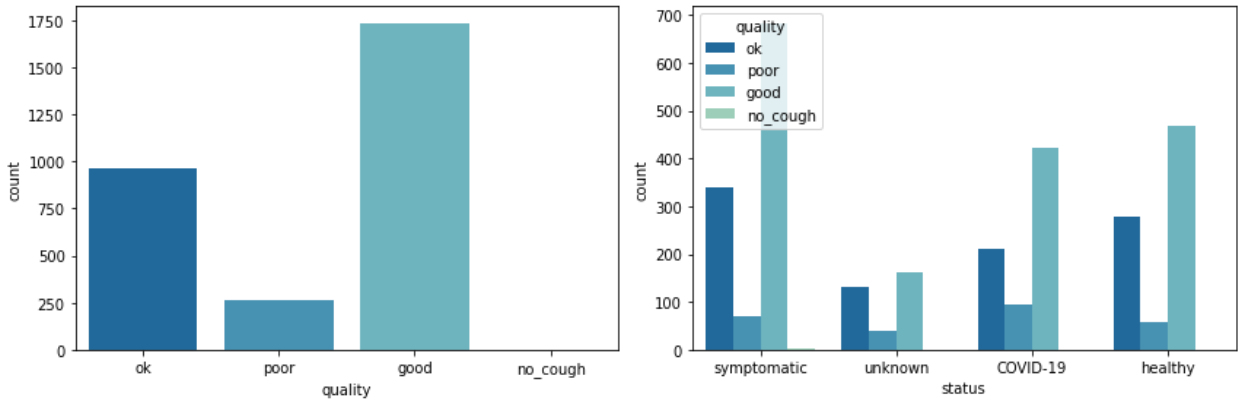
图 7.13：右侧装置 显示医生记录的样本数量，左侧则显示医生根据患者的状态 *ed* 。



7.3.2 咳嗽质量

医生评估了分配给他们的样本的咳嗽性质。结果如图 7.14 所示。总体而言，大多数样本的咳嗽质量良好；按照患者状态检查咳嗽性质后发现，不存在针对特定状态的咳嗽质量偏差，如图 7.14 所示。

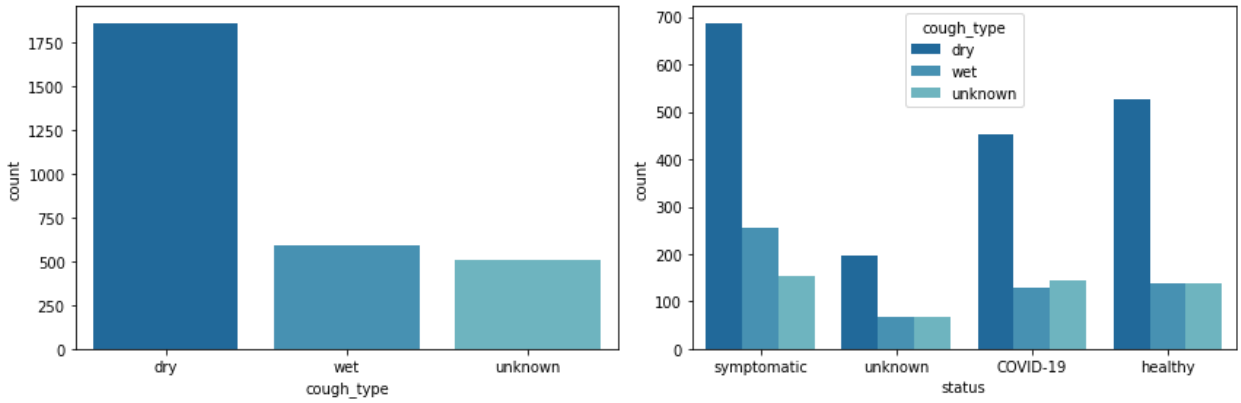
图 7.14：左图显示咳嗽质量的总体分布。右图显示咳嗽质量与患者状态的关系



7.3.3 咳嗽类型

医生还评估了每个样本的咳嗽类型，并将其标记为干咳、湿咳或未知。图 7.15 展示了咳嗽类型的总体分布，以及按患者状态划分的分布。最常见的咳嗽类型是干咳，这也是新冠肺炎感染患者常见的咳嗽类型（Diwan，2020）。

图 7.15：左图：咳嗽类型的总体分布。右图：咳嗽类型与患者状态的关系



7.3.4 咳嗽诊断和严重程度

医生还分析了咳嗽样本，并按照图 7.16 所示，对每个样本作出诊断，同时评估咳嗽的严重程度。虽然所有数据样本都被标记为健康、有症状、新冠肺炎感染或未知，但医生将咳嗽标记为健康、下呼吸道感染、新冠肺炎感染性阻塞性疾病、上呼吸道感染或未知。患者病例的总体严重程度为轻度；如图 7.17 所示，按诊断划分的严重程度没有异常。

图 7.16：左：咳嗽诊断。右：咳嗽严重程度

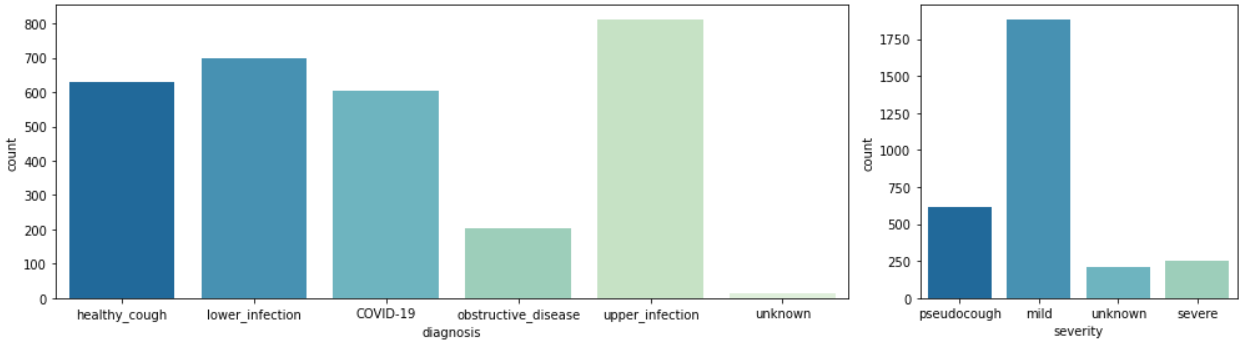
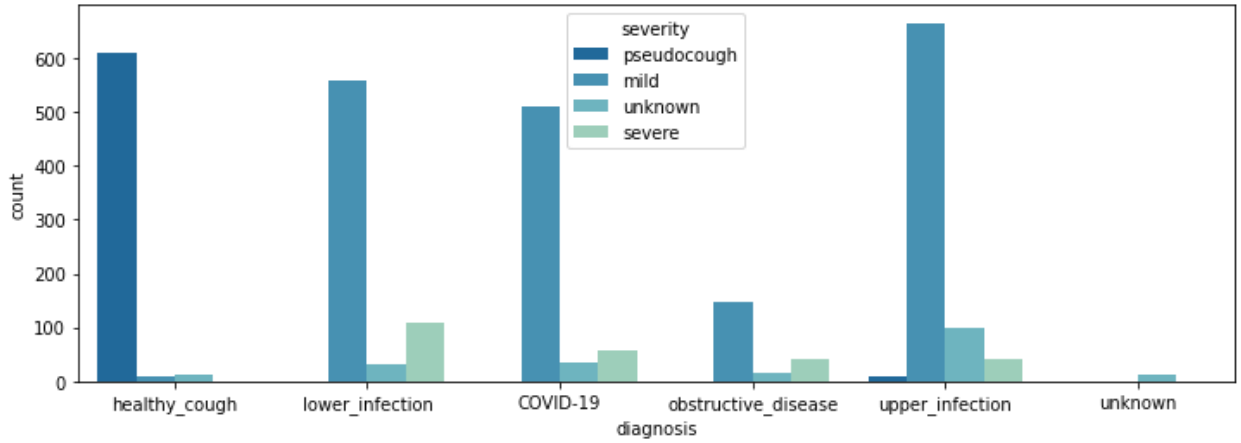


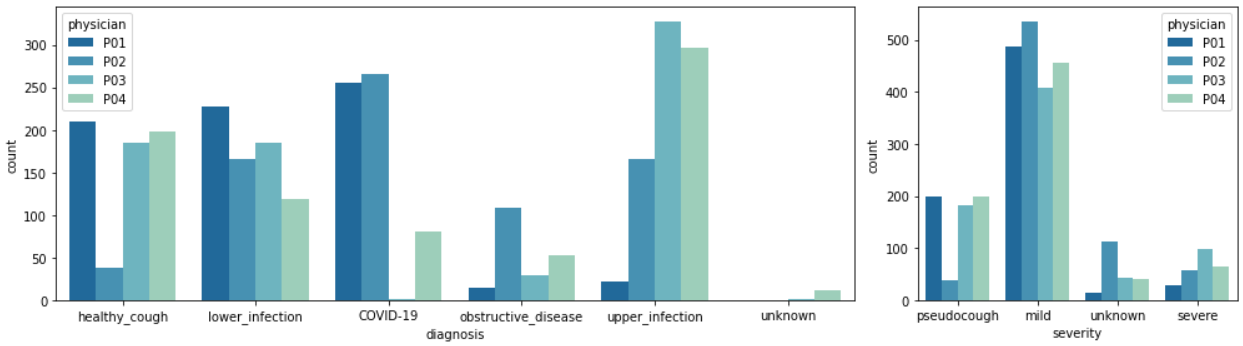
图 7.17：医生对患者病例严重程度的诊断。



7.3.5 专家状态标注与患者诊断

图 7.18 展示了医生对咳嗽的诊断及严重程度。可以注意到，图 7.13 所示提供给医生的状态与医生作出的诊断之间存在差异。

图 7.18：　左：相对于医生的咳嗽诊断。右：相对于医生的咳嗽严重程度　　.

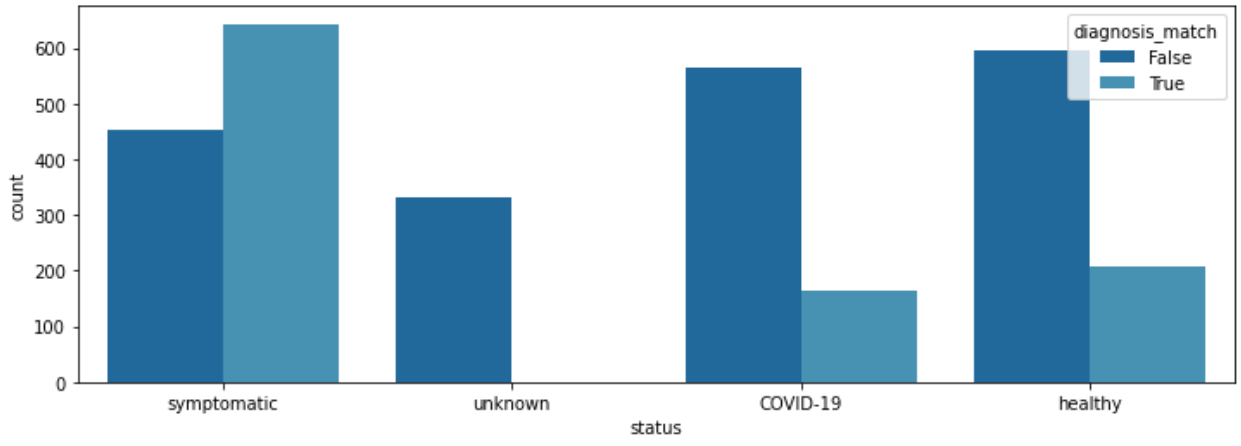


使用图 7.19 所示的代码片段检查诊断匹配情况，诊断与状态的匹配关系如图 7.20 所示。

图 7.19：用于表示诊断与状态匹配关系的代码片段

```
def get_诊断_匹配(通用唯一识别码):
    row = 已标注_df[已标注_df["uuid"] == 通用唯一识别码].iloc[0]

    如果 行.状态 == "健康":
        返回 行.诊断 == "健康_咳嗽"
    否则如果 行.状态 == "有症状":
        返回 行.诊断 在 [
            "下呼吸道_感染","上呼吸道_感染","阻塞性_疾病"
        ]
    否则 行.状态 == "新冠肺炎":
        返回 行.诊断 == "新冠肺炎"
    否则如果 行.状态 == "未知":
        返回 行.诊断 == "未知"


已标注_df["诊断_匹配"] = 已标注_df["通用唯一识别码"].应用(获取_诊断_匹配)

plt.图形(图形大小=(12, 4))
sns.计数图(数据=已标注_df, x="状态", 色调="诊断_匹配")
plt。show()
```

图 7.20：诊断与状态的匹配



研究发现，医生的诊断与患者报告的状态存在巨大差异。这些发现并不构成排除依据，但无疑会使人对患者所报告状态的有效性产生怀疑。

7.3.6 可听注释

图 7.21 显示了是否存在可听见的呼吸困难、喘鸣、窒息或充血。总体而言，只有极少数样本包含任何这类可听见的症状，因此会将其剔除，以免影响模型的训练过程。

图 7.21：数据集中可听到的呼吸困难、喘鸣、窒息或充血的情况

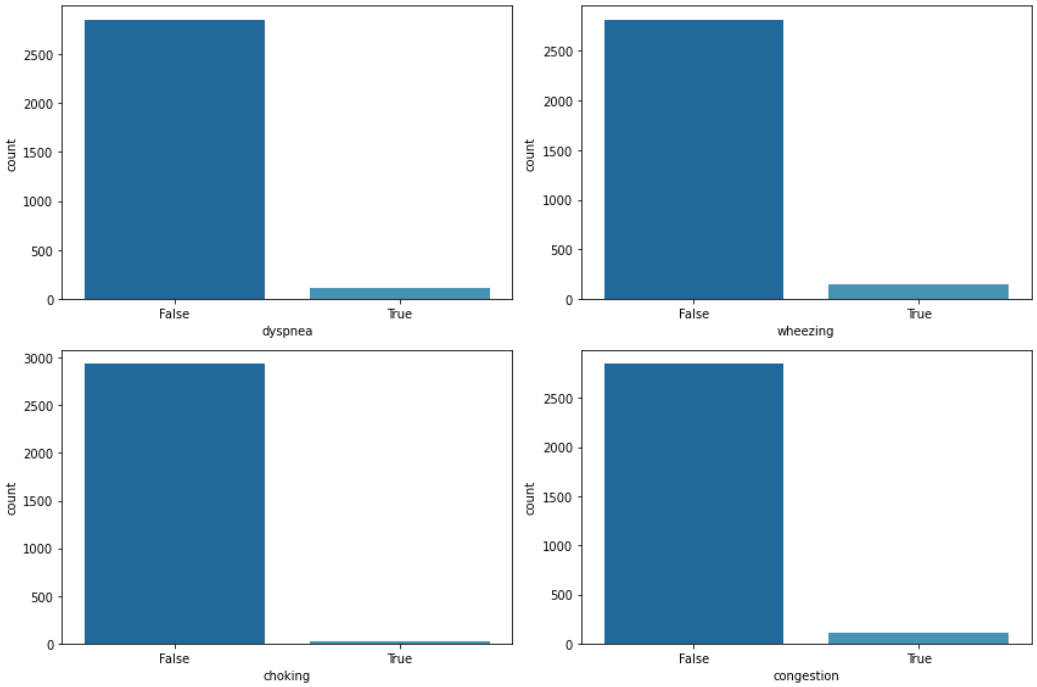
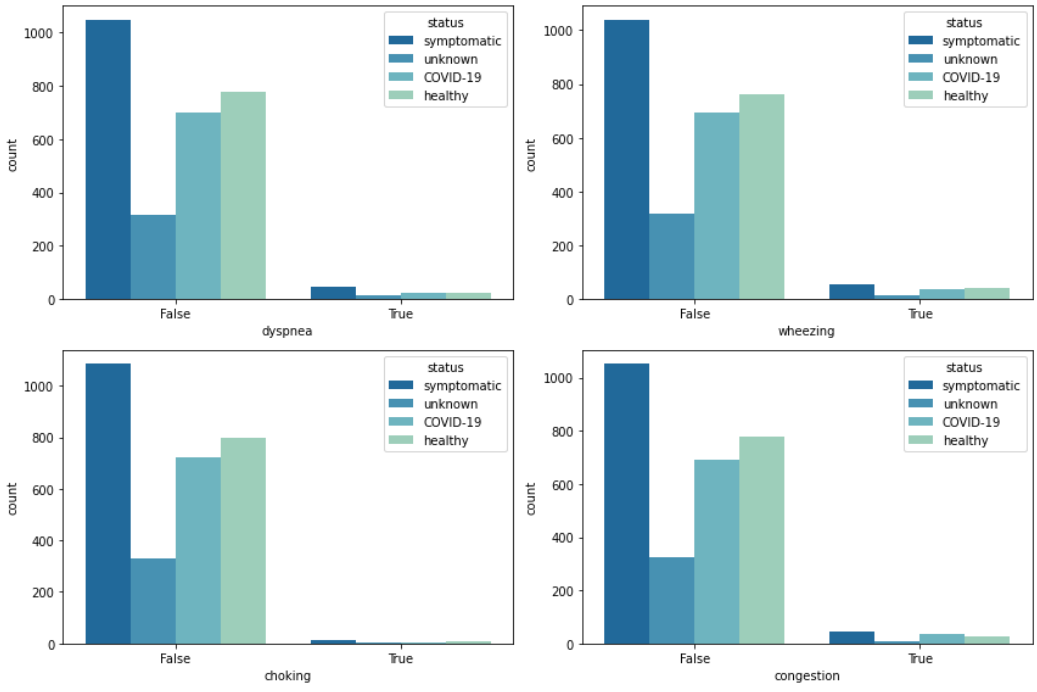


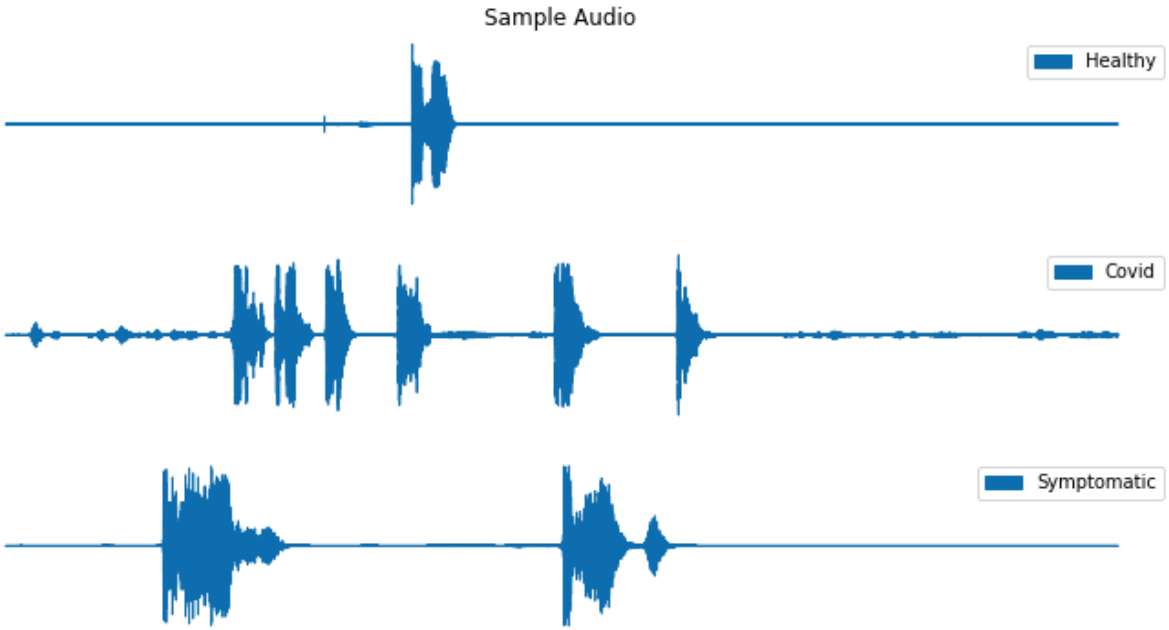
图 7.22：可听到的情况 数据集中呼吸困难、喘鸣、窒息或充血的情况与咳嗽状态的关系 *s*



8 音频预处理

在探索数据并进行适当清理后，下一步是对音频数据进行预处理。图 8.1 展示了从三种音频类别（健康、有症状和新冠肺炎）中随机选取的样本。

图 8.1： 从三种音频类别（健康、有症状和新冠肺炎）中随机选取的样本。



检查这些样本可以发现，音频样本的时长各不相同，而且包含大量静音或噪声。因此，预处理步骤不仅要对音频进行归一化并为模型做好准备，还必须统一音频时长并处理噪声和静音。

8.1 音频数据预处理

图 8.2 所示的函数会加载一个音频样本，并对音频进行归一化和降采样，以便更好地表示。该函数通过将音频转换为单声道（即单个音频通道）来处理音频片段。随后，将音频值归一化到 -1 至 1 的范围内。最后，对音频进行降采样以实现抗混叠。

图 8.2：用于加载和预处理咳嗽音频样本的函数

```
从 scipy 导入 signal
来自 scipy.signal 导入 butter, filtfilt


定义 预处理_咳嗽(x, fs, cutoff=6000, 归一化=真, filter_=真,降采样=真):
```

```
    # 对给定数据中的咳嗽样本进行归一化、低通滤波和降采样
文件夹
    fs_降采样 = 截止 * 2

    # 数据预处理
    如果 长度(x.shape) > 1:
        x = np.均值(x, 轴=1)  # 转换为单声道
    如果 归一化 e:
        x = x / (np.最大值(np.绝对值(x)) + 1e-17)  # 将幅度归一化到 -1 到 1 的范围
    如果 滤波器_:
        b, a = 巴特沃斯滤波器(
            4, 采样频率_降采样 / 采样频率, 滤波器类型="低通滤波器"
        ) # 四阶巴特沃斯低通滤波器
        x = filtfilt(b, a, x)
    如果 降采样:
        <escape>x = 信号。降采样(x, 整数(fs / fs_降采样))  # 用于抗混叠的降采样</escape>
混叠

    采样率_新 = 采样率_降采样

    返回 np.float32(x), 采样率_新
```

8.2 用于数据增强的咳嗽分割

重新查看图 8.1 所示的样本，可以发现单个音频样本包含多个咳嗽实例；因此，可以将单个音频样本分割成多个样本，并进行归一化，使其长度统一。图 8.3 所示的函数会将给定的预处理后的咳嗽分割成多个样本片段，每个片段都只包含一次咳嗽。具体做法是为单段长度设置一个阈值，阈值以下的单段被视为音频；随后，将阈值以上的连续信号拼接成一个音频片段，然后重复此过程。

图 8.3：用于生成咳嗽分割掩码以及分割后的咳嗽片段的函数

```
def segment_咳嗽( x, fs, 咳嗽_填充=0.2, min_咳嗽_len=0.2, th_l_multiplier=0.1, th_h_
```

```
    片段_th_l = th_l_乘数 * 均方根
    片段_th_h = th_h_乘数 * 均方根

    # 咳嗽片段
    咳嗽_片段 = []
    填充 = 取整(fs * 咳嗽_填充)
    最小_咳嗽_样本 = 取整(fs * 最小_咳嗽_长度)
    咳嗽_开始 = 0
    咳嗽_结束 = 0
    cough_in_进度      = False
    容差 = 取整(0.01 * fs)
    如下 _th_计数器 = 0

    对于 i, 样本 在 enumerate(x ** 2):
        if cough_in_progress:
            如果 样本 < 分_段_l:
                如下 _th_计数器 += 1
                如果 低于 _th_计数器 > 容差:
                    咳嗽_结束 = i + 填充 如果 (i + 填充 < len(x)) 否则 len(x)
- 1
                    cough_in_progress = False
                    如果 咳嗽_结束 + 1 - 咳嗽_开始 - 2 * 填充 >
最少_咳嗽_样本:
                        咳嗽_片段。追加(x[咳嗽_开始： 咳嗽_结束 + 1])
                        咳嗽_口罩[咳嗽_开始 : 咳嗽_结束 + 1] = 真
            否则 i == (长度(x) - 1):
                咳嗽_结束 = 我
                cough_in_progress = False
                如果 咳嗽_结束 + 1 - 咳嗽_开始 - 2 * 填充 > 最小值_咳嗽_样本:
                    咳嗽_片段。追加(x[咳嗽_开始 : 咳嗽_结束 + 1])
            else:
                下方 _th_计数器 = 0
        else:
            如果 样本 > 段_长度_小时:
                咳嗽_开始 = i - 填充 如果 (i - 填充 >= 0) 否则 0
                cough_in_progress = True

    返回 咳嗽_片段, 咳嗽_掩码
```

使用图 8.3 和图 8.2 中的两个函数，可以识别出图 8.4 所示的咳嗽片段和掩码，并使用掩码生成相应的咳嗽片段。生成的片段示例如图 8.5 所示。应用这两个函数的代码片段如图 8.6 所示。

图 8.4：采样音频中的咳嗽掩码

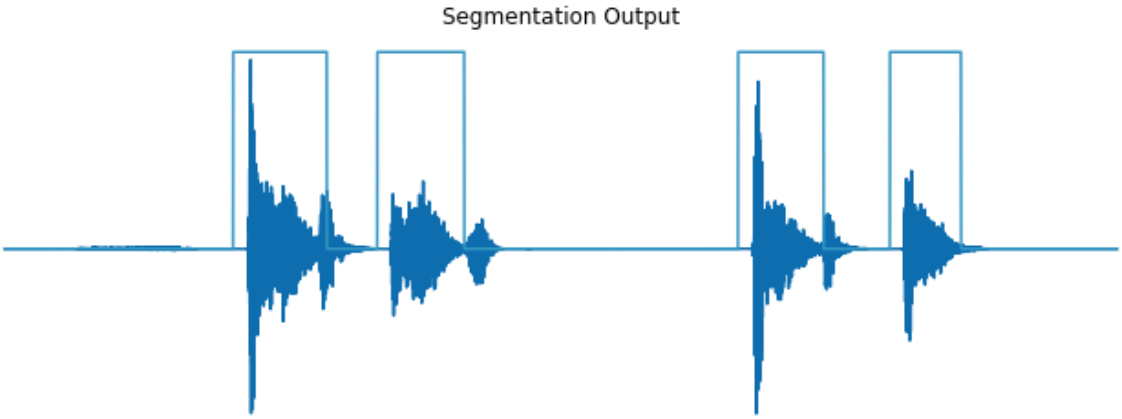
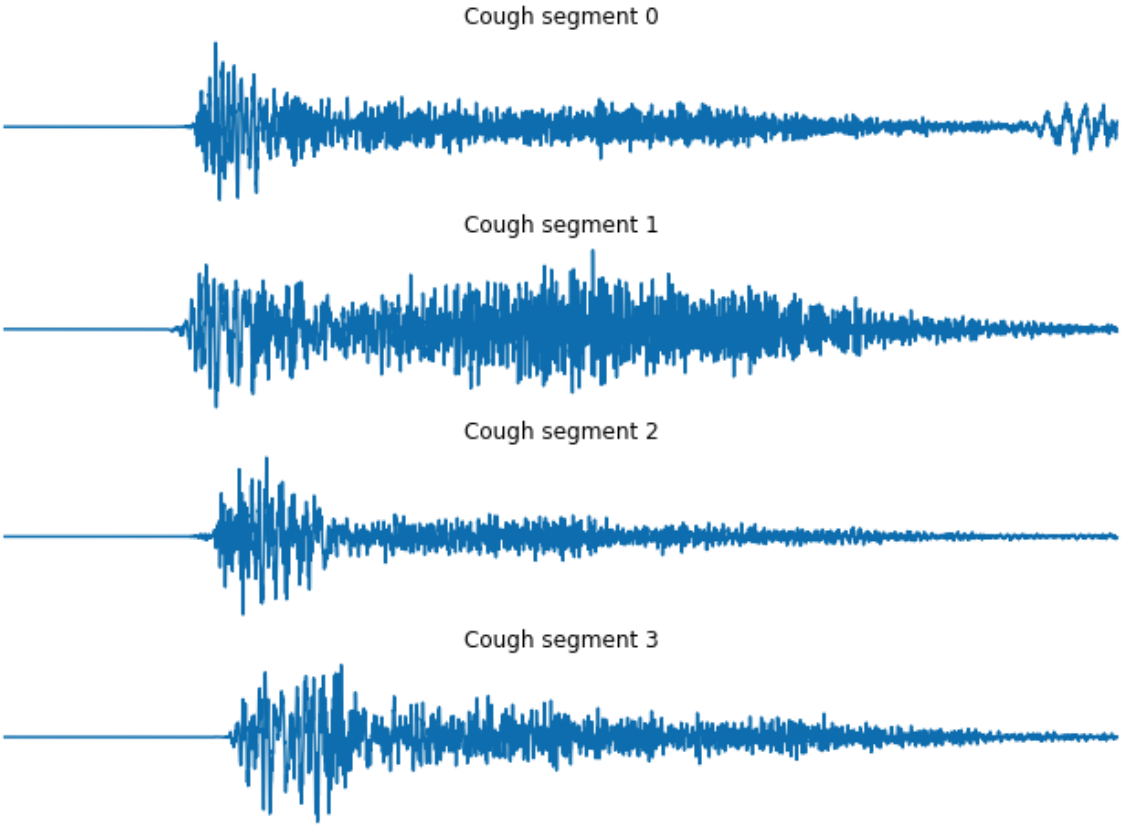


图 8.5：图 8.4 所示音频样本生成的咳嗽片段



*Figu*re 8.6：处理音频样本并生成其咳嗽片段的代码片段

```
# 预处理（归一化、低通滤波并对咳嗽样本进行降采样）已处理_音频，样本_率 = 预
```

```
# 使用信号功率上的滞回比较器将每段音频分割为单个咳嗽片段 cough_segments, cough_
```

8.3 存储预处理后的咳嗽片段

图8.7所示的代码片段对从所有类别中随机选取的若干样本进行预处理。选取时，会选出所有新冠肺炎音频样本，并从其他每个类别中随机选取数量相同的样本。这种选取方法平衡了各类别的数据分布。随后，图8.8展示了将处理后的样本保存为 NumPy文件的代码片段，其中会为每个类别创建一个目录。将数据保存为 NumPy文件可以显著加快加载过程。

图8.7：从每个类别中选取数量相等的样本，其中所有 COVID样本均被选中

```
类别_大小 = 长度(数据_完整[数据_完整.状态 == "新冠肺炎"])
class2df = {}
对于 状态 在 数据_原始.状态.删除缺失值()。唯一():
    class2df[状态] = 数据_完整[数据_完整.状态 == 状态].样本(类别_大小)
已选择_数据 = pd.concat(class2df.values())


音频_长度_以_秒为单位 = 1
用于_保存 = 默认字典(字典)
对于 i，行 在 tqdm（选定_数据。按行迭代():
    音频_路径 = os。路径。连接（
        音频 _ROOT，
        行.通用唯一识别码 + “.wav” ，
    )
    音频，采样率 = librosa。加载（音频_路径，单声道=真）

    # 预处理（归一化、低通滤波器和降采样咳嗽样本）
```

```
    已处理_音频，采样_率 = 预处理_咳嗽(音频，fs)

    # 使用基于信号功率的滞回比较器，将每段音频分割为单独的咳嗽
信号功率
    咳嗽_片段，咳嗽_掩码 = 片段_咳嗽(
        已处理_音频，
        采样_率，
        最小_咳嗽_长度=0.1，
        咳嗽_填充=0.1，
        th_l_乘数=0.1，
        th_h_乘数=2，
    )

    咳嗽_片段 = [
        librosa.util.fix_长度(
            咳嗽_片段，整数(采样_率 * 音频_长度_（单位为_秒）
        )
        用于 咳嗽_片段 在 咳嗽_片段
    ]

    咳嗽[行.状态]。扩展(咳嗽_片段)
    用于_保存[行.状态][行.通用唯一识别码] = 长度(咳嗽_片段)

咳嗽 = {状态：np.array(咳嗽_片段) for 状态，咳嗽_片段 in
咳嗽。项目()}
```

图8.8：将预处理后的数据存储到 NumPy 文件中，以便快速处理和加载

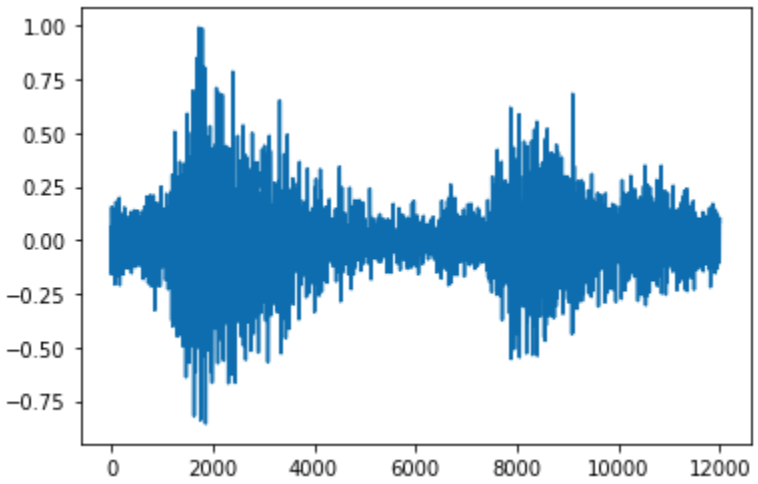
```
如果 不 os。path。exists(Coughvid_咳嗽_segments_path):
    操作系统。创建目录(Coughvid_咳嗽_片段_路径)

对于 状态，uuid2cough_计数 在 tqdm（对于_保存。项目（）：
    状态_路径 = 操作系统.路径.拼接(Coughvid_咳嗽_片段_路径，状态.转小写())
    如果 不存在 操作系统.路径.存在(状态_路径):
        操作系统.创建目录(状态_路径)

    已保存_计数 = 0
    对于 通用唯一识别码，咳嗽_计数 在 tqdm(uuid2cough_计数.项目()):
        咳嗽_片段 = 咳嗽[状态][已保存_数量：已保存_数量 + 咳嗽_数量]
        已保存_计数 += 咳嗽_计数

        对于 i，咳嗽_片段 在 枚举(咳嗽_片段):
            np.save(
                操作系统。路径。连接(状态_路径，f"{通用唯一识别码}-{i}.npy"),
                咳嗽_片段，
```

```
                允许_pickle=真.)
```

9 特征提取

数据预处理阶段之后是音频数据的特征提取。在开发用于音频任务的深度学习模型时，常见的做法是以可视化形式表示音频特征，并将生成的特征表示用于相应任务。事实证明，音频特征表示有多种不同的算法，而选择最优算法是完成音频任务的关键。以下各小节将讨论不同的特征提取算法，并展示在图9.1所示的随机选取音频样本上得到的结果。

图9.1：随机选取的音频样本



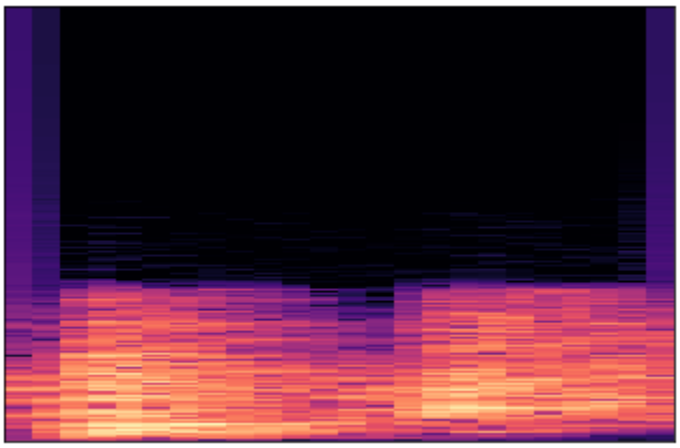
9.1 频谱图

频谱图是一种图形表示，用于展示波形中存在的各种频率下，信号强度（或“响度”）随时间的变化。我们不仅可以看到某个特定频率（例如 2 Hz 与 10 Hz）上的能量是更多还是更少，还可以观察能量水平如何随时间变化。

9.1.1 短时傅里叶变换

短时傅里叶变换（STFT）是处理音频信号极其有用的工具。它定义了一类特别有用的时频分布，用于描述任意信号的复振幅随时间和频率变化的特征。该分布通过对信号应用短时傅里叶变换（STFT）获得。简单来说，信号的STFT是通过对信号的短时片段局部应用快速傅里叶变换（FFT）计算得到的。通常，频谱图以热图的形式呈现，即通过颜色或亮度的变化表示强度。图9.2显示了图9.1的STFT

图9.2：图9.1的短时傅里叶变换



9.1.2 梅尔频谱图

人类以对数方式感知声音。与较高频率的差异相比，我们对较低频率的差异更加敏感。例如，我们可以轻松区分500 Hz和1000 Hz，却很难区分10,000 Hz和10,500 Hz，尽管两对频率之间的距离相同。因此，梅尔尺度应运而生。它是一种基于以下公理的对数尺度：尺度上相等的距离对应于相同的感知距离。图9.3显示了图9.1的梅尔频谱图。

图9.3：图9.1的梅尔频谱图

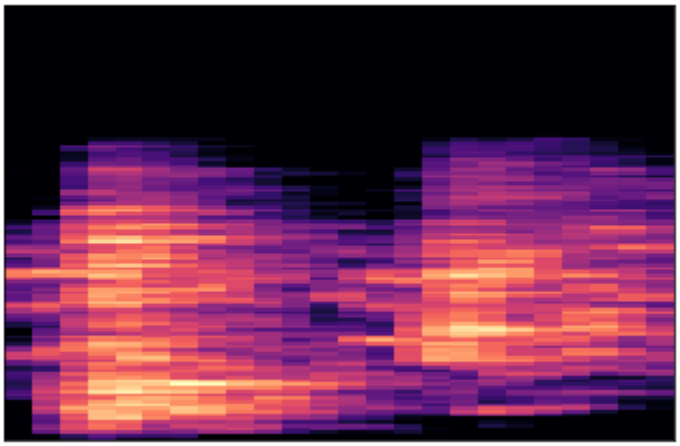
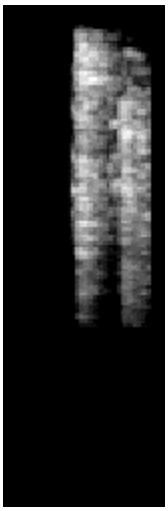


图9.4：这　是图9.1的梅尔频谱图，带有坐标轴和灰度　缩放

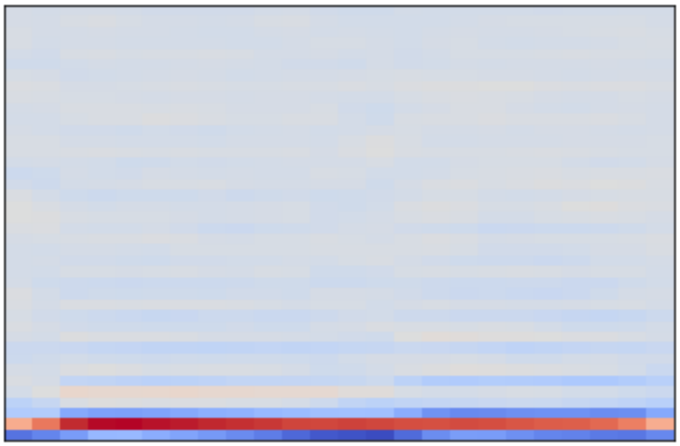


9.2 梅尔频率倒谱系数

梅尔频率倒谱系数（MFCC）是构成梅尔频率倒谱的系数。信号的倒谱包含有关其频谱带变化率的信息。

倒谱本质上是时域信号频谱的对数所形成的频谱。所得频谱既不处于频域，也不处于时域，因此被称为倒谱域。倒谱传达了构成语音共振峰（语音的一种性质特征组件）和音色的各种数值。图9.5展示了图9.1的梅尔频率倒谱系数。

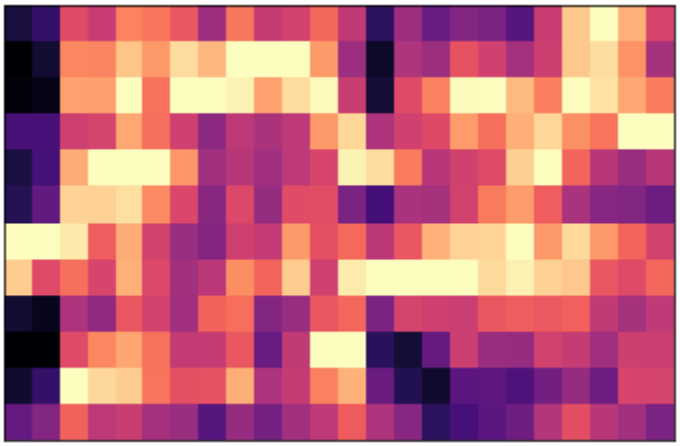
图9.5：图9.1 的梅尔频率倒谱系数。



9.3 色度特征

色度特征是一种非常有效的音乐音频表示方法，它采用一种包含 12 个元素的频谱能量表示，称为色度向量。色度向量中的每个元素都代表西方音乐中十二个平均律音级之一（以半音为间隔）。它可以根据输入声音信号的对数短时傅里叶变换计算得出，也称为色谱图或音级轮廓。图9.6 展示了图9.1 的色度特征。

图9.6：图9.1的色度特征



9.4 选定的特征表示算法

选定的特征表示算法是梅尔频谱图，它是音频数据的代表性算法。梅尔频谱图能够丰富地表示音频样本，同时计算开销不高（Doshi，2021年）。

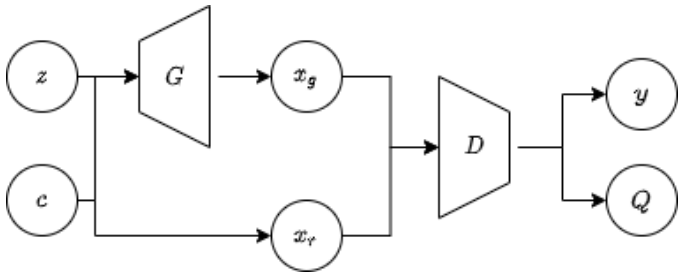
10 生成对抗网络实现

本报告第 2.1 节已详细讨论生成对抗网络。本节将讨论选定的生成对抗网络流程及其实现。针对咳嗽合成任务测试了多种生成对抗网络变体，包括深度卷积生成对抗网络（DCGAN）、WaveGAN（Donahue等人，2018年）和辅助分类器生成对抗网络（ACGAN）（Odena等人，2017年）。最终选定 ACGAN 作为最适合该任务的模型。

10.1 辅助分类器生成对抗网络（ACGAN）

如图10.1所示，辅助分类器生成对抗网络（ACGAN）是由Odena等人（2017年）提出的一种条件生成对抗网络变体，它改变了判别器，使其预测图像类别，而不是将其作为输入接收。判别器输出两个概率分布，一个表示给定图像是真实的还是生成的，另一个表示给定图像的类别标签。同样，生成器有两个输入：从潜在空间采样的噪声，以及所需生成样本的类别标签。这种简单的修改使训练过程更加稳定。

图10.1：ACGAN架构。c为类别标签，z为随机噪声，Q为判别器对类别标签的概率分布，y为判别器输入为真实图像的概率



选择 ACGAN 来完成这项任务是最佳方案，因为它支持使用多种类别标签来训练模型。如果使用普通的 DCGAN 或 WaveGAN，则只能使用 COVID-19 音频片段来训练模型；不过，由于生成器会接收所需数据的类别标签，因此可以使用更多类别标签来训练 ACGAN。

10.2 判别器

判别器以频谱图作为输入，该频谱图可以是生成的，也可以是真实的；判别器会判断输入所属的类别，并计算其为真实数据的概率。图 10.2 和图 10.4 展示了 ACGAN 的判别器架构。

图 10.2：判别器架构代码

```
def make_判别器_模型():
    模型 = 模型。Sequential()
    模型。添加(
```

```
        层。二维卷积(
            32,
            内核_大小=(3, 3),
            步幅=(1, 1),
            输入_形状=图像_形状.
            填充="same",
        )
    )
    模型。添加(层。批归一化(动量=0))
    模型。添加(层。带泄漏的ReLU(alpha=0.2))
    模型。添加(随机失活(0.5))

    模型。添加(
        层。二维卷积(
            64,
            卷积核_大小=(3, 3),
            步幅=(2, 2),
            输入_形状=图像_形状.
            填充="same",
        )
    )
    模型。添加(lavers.批归一化(动量=0))
    模型。添加(layers.带泄漏的ReLU(alpha=0.2))
    模型。添加(层。随机失活(0.5))

    模型。添加(层。二维卷积(128, 内核_大小=(3, 3), 步幅=(2, 2),
填充="same"))
    模型。添加(层。批归一化(动量=0))
    模型。添加(层。带泄漏的ReLU(alpha=0.2))
    模型。添加(层。随机失活(0.5))

    模型。添加(层。二维卷积(256, 卷积核_尺寸=(3, 3), 步长=(2, 2),
填充="same"))
    模型。添加(层。批归一化(动量=0))
    模型。添加(层。带泄漏的ReLU(alpha=0.2))
    模型。添加(层。随机失活(0.5))

    模型。添加(层。二维卷积(512, 卷积核_尺寸=(3, 3), 步长=(2, 2),
填充="同一"))
    模型。添加(层。批归一化(动量=0))
    模型。添加(层。带泄漏的ReLU(alpha=0.2))
    模型。添加(层。随机失活(0.5))

    模型。添加(层。展平())
```

```
    img = layers.Input(shape=IMG_SHAPE) 特征 = model(img) 有效性 =
```

10.2.1 结构

判别器接收一个形状统一的频谱图，然后通过一系列经过批归一化、带泄漏的ReLU和随机失活增强的卷积层对其进行处理。随后，输出会被展平并传递到两个不同的层，一个用于验证，另一个用于标记。有效性层用于判断给定输入是真实还是伪造，并使用 Sigmoid 激活函数；标签层则生成一个概率分布，表示给定图像属于特定类别的可能性，并使用 Softmax 激活函数。

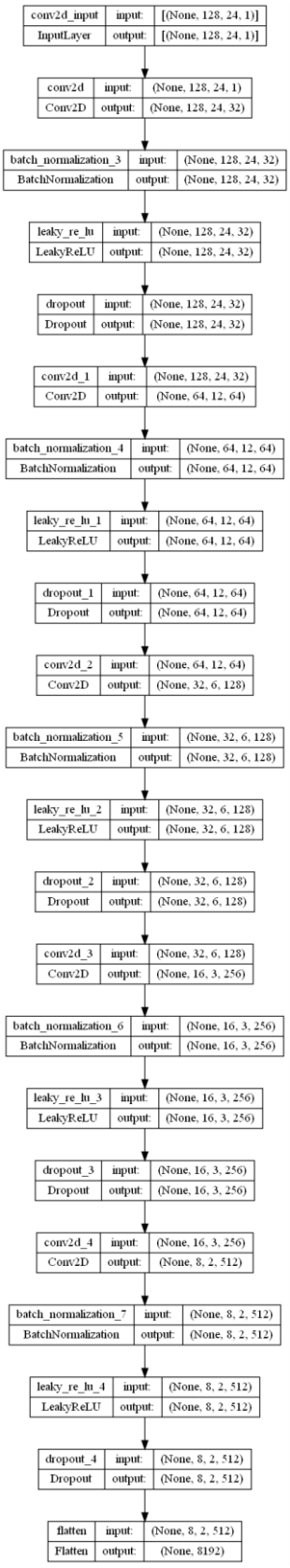
10.2.2 优化

为了训练模型，使其能够更好地对输入进行分类并验证输入，使用了两个损失函数。首先，使用 Adam优化器和较小的学习率来优化模型，事实证明这能确保训练稳定。至于损失函数，则选择了["二元_交叉熵"、"稀疏_分类_交叉熵"] 。二元交叉熵用于优化验证输[redacted]分类损失。

图10.3：判别器的优化器和损失函数选择

```
优化器 = tf.keras.optimizers.Adam(0.0002, 0.5) 损失 = ["binary_crossentropy", "
```

图10.4：判别器架构


conv2d_input | input: | [(None, 128, 24, 1)]
InputLayer | output: | [(None, 128, 24, 1)]
conv2d | input: | (None, 128, 24, 1)
Conv2D | output: | (None, 128, 24, 32)
batch_normalization_3 | input: | (None, 128, 24, 32)
BatchNormalization | output: | (None, 128, 24, 32)
leaky_re_lu | input: | (None, 128, 24, 32)
LeakyReLU | output: | (None, 128, 24, 32)
dropout | input: | (None, 128, 24, 32)
Dropout | output: | (None, 128, 24, 32)
conv2d_1 | input: | (None, 128, 24, 32)
Conv2D | output: | (None, 64, 12, 64)
batch_normalization_4 | input: | (None, 64, 12, 64)
BatchNormalization | output: | (None, 64, 12, 64)
leaky_re_lu_1 | input: | (None, 64, 12, 64)
LeakyReLU | output: | (None, 64, 12, 64)
dropout_1 | input: | (None, 64, 12, 64)
Dropout | output: | (None, 64, 12, 64)
conv2d_2 | input: | (None, 64, 12, 64)
Conv2D | output: | (None, 32, 6, 128)
batch_normalization_5 | input: | (None, 32, 6, 128)
BatchNormalization | output: | (None, 32, 6, 128)
leaky_re_lu_2 | input: | (None, 32, 6, 128)
LeakyReLU | output: | (None, 32, 6, 128)
dropout_2 | input: | (None, 32, 6, 128)
Dropout | output: | (None, 32, 6, 128)
conv2d_3 | input: | (None, 32, 6, 128)
Conv2D | output: | (None, 16, 3, 256)
batch_normalization_6 | input: | (None, 16, 3, 256)
BatchNormalization | output: | (None, 16, 3, 256)
leaky_re_lu_3 | input: | (None, 16, 3, 256)
LeakyReLU | output: | (None, 16, 3, 256)
dropout_3 | input: | (None, 16, 3, 256)
Dropout | output: | (None, 16, 3, 256)
conv2d_4 | input: | (None, 16, 3, 256)
Conv2D | output: | (None, 8, 2, 512)
batch_normalization_7 | input: | (None, 8, 2, 512)
BatchNormalization | output: | (None, 8, 2, 512)
leaky_re_lu_4 | input: | (None, 8, 2, 512)
LeakyReLU | output: | (None, 8, 2, 512)
dropout_4 | input: | (None, 8, 2, 512)
Dropout | output: | (None, 8, 2, 512)
flatten | input: | (None, 8, 2, 512)
Flatten | output: | (None, 8192)

10.3 生成器

如前所述，ACGAN 网络的生成器经过修改，可接受两个输入：噪声和待生成数据的类别标签。图 10.5 和图 10.9 展示了用于生成咳嗽音频片段的生成器架构。

图 10.5：生成器架构代码

```
定义 创建_生成器_模型():
    噪声= 层。输入(形状=(潜在_维度,))
    标签 = 层。输入(形状=(1,), 数据类型="int32")

    噪声_分支 = 层。全连接层(1024 * 16 * 3)(噪声)
    噪声_分支 = 激活函数。relu(噪声_分支)
    噪声_分支 = 层。重塑((16, 3, 1024))(噪声_分支)
    噪声_分支 = 模型。模型(输入=噪声，输出=噪声_分支)

    标签_分支 = 层。嵌入(输入_维度=50, 输出_维度=1)(标签)
    标签_分支= 层。全连接层(7 * 7, 输入_形状=(7, 7))(标签_分支)
    标签_分支= 激活函数。线性(标签_分支)
    标签_分支= 层。重塑(
        <style 标识符='1'>(<style 标识符='3'>7<style 标识符='5'>, <style 标识符='7'>7<style 标识符='9'>, <style 标识符='11'>1<style 标识符='13'>),
    )(标签_分支)
    标签_分支= 模型。模型(输入=标签，输出=标签_分支)

    # 权重初始化
    初始化 = tf.keras.initializers.随机正态分布(标准差=0.02)

    合并 = layers.拼接([噪声_branch.output, 标签_branch.output])

    合并 = layers.二维转置卷积(
        512, (5, 5), 步长=(2, 2), 填充="same", 卷积核_初始化器=初始化
    )(combined)
    组合 = 层。批归一化(动量=0)(组合)
    组合 = 激活函数。relu(组合)

    组合 = 层。二维转置卷积(
        256, (5, 5), 步幅=(2, 2), 填充="same", 卷积核_初始化器=初始化
    )(组合)
    组合 = 层。批归一化(动量=0)(组合)
    组合 = 激活函数。ReLU(组合)

    组合 = 层。Conv2DTranspose(
        128, (5, 5), 步幅=(1, 1), 填充="same", 内核_初始化器=初始化
    )(组合)
    组合 = 层。批归一化(动量=0)(组合)
```

```
    组合 = 激活函数。relu(组合)

    组合 = 层。Conv2DTranspose(
        1, (5, 5), 步幅=(2, 2), 填充="same", 内核_初始化器=初始化
    )(组合)
    组合= 激活。双曲正切(组合)

    模型= 模型。模型(
        输入=[标签_分支输入、噪声_分支输入], 输出=组合
    )

    tf.keras.utils.绘制_模型(模型, "generator.png", 显示_形状=真)

    返回 模型
```

10.3.1 类别标签

虽然 ACGAN 论文规定生成器模型应接收一个由潜在空间和类别标签拼接而成的向量输入，但现代方法会将两者分为两个分支：噪声分支和标签分支。标签分支被视为额外的特征图，这通过使用具有任意维度数量（本例中为 50）的学习到的嵌入来实现。然后，可以将嵌入层的输出传递给带有线性激活函数的全连接层，以生成一个 7×7 的特征图。

图 10.6：标签分支

```
    标签_分支 = 层.Embedding(输入_维=50, 输出_维=1)(标签) 标签_分支 = 层.
```

10.3.2 噪声

来自定义的 50,000 维潜在空间的噪声，经由带有 ReLU 激活函数的全连接层处理，生成 1024 个 16×3 的特征图，每个特征图都表示预期输出的低分辨率版本。随后，噪声和标签会先按通道拼接，再传递给模型。

图 10.7：噪声分支以及噪声分支与标签分支的拼接

```
    噪声 = 层。输入(形状=(潜在_维度,))
    标签 = 层。输入(形状=(1,), 数据类型="int32")
```

```
    噪声_分支= 层。全连接层(1024 * 16 * 3)(噪声)
    噪声_分支= 激活值。relu(噪声_分支)
    噪声_分支= 层。重塑((16, 3, 1024))(噪声_分支)
    噪声_分支= 模型。模型(输入=噪声，输出=噪声_分支)


      …

    合并 = 层。拼接([噪声_分支输出，标签_分支输出])
```

10.3.3 二维转置卷积

通过一系列二维转置卷积层，16×3 的特征图以二的迭代倍数进行上采样，直到达到所需的 128×24 输出形状。根据 ACGAN 论文，使用 ReLU 激活函数代替带泄漏的 ReLU。最后使用双曲正切激活函数，生成器的输出为单个特征图，其像素值介于 [-1, 1] 之间。

图 10.8：生成器的二维转置卷积

```
# 权重初始化
    初始化 = tf.keras.initializers.随机正态分布(标准差=0.02)

    合并 = 层。拼接([噪声_分支输出，标签_分支输出])

    组合 = 层。二维转置卷积(
        512, (5, 5), 步幅=(2, 2), 填充="same", 内核_初始化器=初始化
    )(组合)
    组合 = 层。批归一化(动量=0)(组合)
    组合 = 激活函数。relu(组合)

    组合 = 层。二维转置卷积(
        256, (5, 5), 步幅=(2, 2), 填充="same", 卷积核_初始化器=init
    )(combined)
    combined = 层。批归一化(动量=0)(combined)
    combined = 激活函数。relu(combined)

    combined = 层。二维转置卷积(
        128, (5, 5), 步幅=(1, 1), 填充="same", 卷积核_初始化器=init
    )(合并)
    合并 = 层。批归一化(动量=0)(合并)
    合并 = 激活函数。relu(合并)

    合并 = 层。二维转置卷积(
        1, (5, 5), 步幅=(2, 2), 填充="same", 卷积核_初始化器=初始化
    )(合并)
```

```
    组合 = 激活函数。双曲正切(组合)

    模型 = 模型。模型(
        输入=[标签_分支输入，噪声_分支输入]，输出=组合
    )

    tf.keras.utils.绘制_模型(模型，"generator.png"，显示_形状=真)

    返回 模型
```

10.3.4 优化与损失

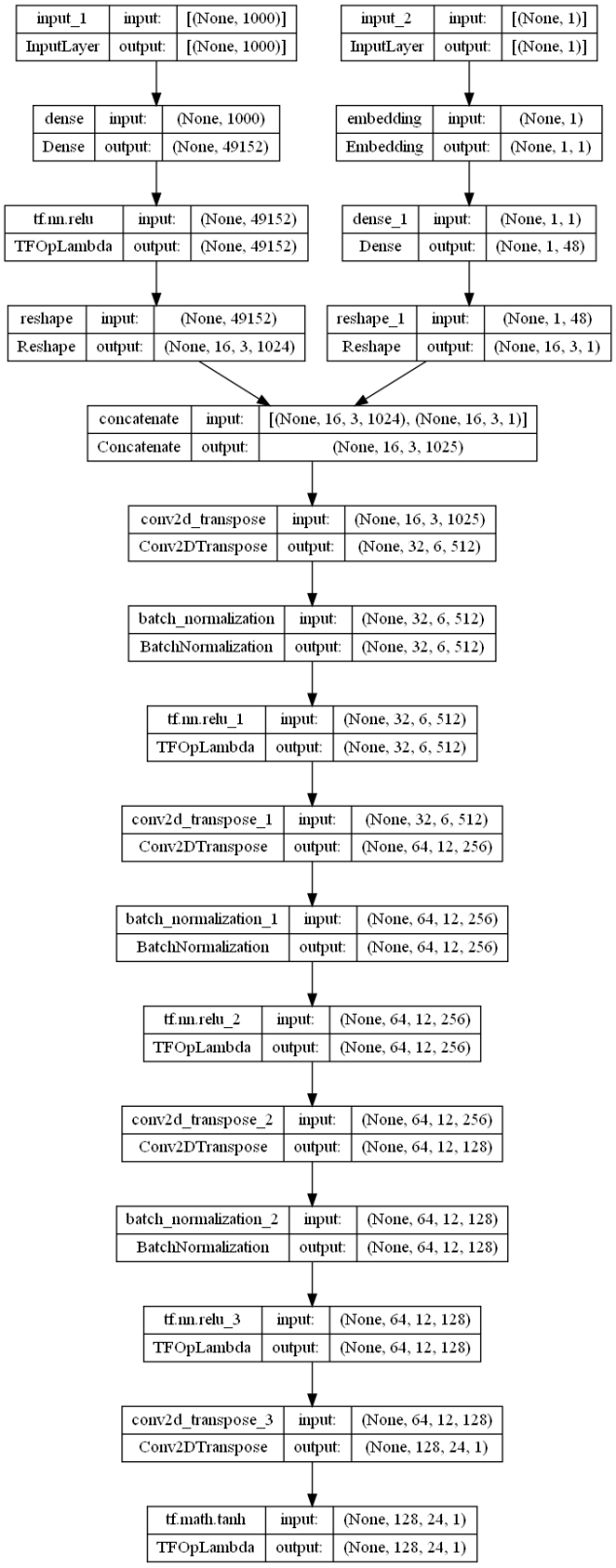
生成器网络使用判别器损失进行优化。具体做法是创建一个由生成器和判别器组成的复合模型，并确保在复合模型反向传播时不更新判别器的权重。这样就能以间接的方式优化生成器。由于复合模型使用判别器损失，因此它的编译方式与判别器相同。

图 10.9：构建复合模型

```
噪声= 层。输入(形状=(潜在_维度,))
标签= 层。输入(形状=(1,))
图像= 生成器([标签、噪声])

判别器.trainable = 假
遍历 层 属于 判别器.layers:
    如果 不 是实例(layer, 批归一化层):
        layer.trainable = 假


有效, 目标_标签 = 判别器(img)

组合 = 模型.模型([标签, 噪声], [有效, 目标_标签])

合并。编译(损失=损失，优化器=优化器)
```

10.3.5 生成器架构

图10.10：生成器架构



10.4 训练与评估

构建 ACGAN 后，下一步是在处理后的数据集上训练模型。图 10.11 展示了一个定制的训练循环，用于训练 ACGAN 模型并同时评估其性能。

图 10.11：定义 ACGAN 的训练循环

```
def 训练(epochs):
    global x_训练

    nb_测试 = 测试_尺寸
    nb_训练 = 训练_大小

    训练_历史 = defaultdict(list)
    测试_历史 = defaultdict(list)

    用于 轮次 在 范围(轮次):

        打印("轮次 {} / {}".格式(轮次 + 1, 轮次))
        批次数_批次 = 整数(批次_训练 / 批次_大小)
        进度_条 = 进度条(目标=批次_批次)

        轮次_生成器_损失 = []
        轮次_判别器_真实_损失 = []
        轮次_判别器_伪造_损失 = []

        对于 索引 在 范围(批次_批次):
            进度_条。更新(索引)

            噪声 = np.random.正态分布(0, 0.02, (批次_大小, 潜在_维度))

            图像_批次 = []
            对于 _ 在 范围(批次_大小):
                try:
                    图像_批次。追加(下一个(x_训练))
                除外 StopIteration:
                    x_训练 = 加载_并_预处理_音频(x_训练_文件_路径)
                    图像_批次。追加(下一个(x_训练))
            图像_批次 = np.数组(图像_批次)

            标签_批次 = y_训练[索引 * 批次_大小:(索引 + 1) * 批次_大小]

            采样得到的_标签 = np.random.randint(0, 1, BATCH_SIZE)

            生成_图像= 生成器。预测(
```

```
                [采样的_标签。重塑((-1, 1)), 噪声], 详细=0
            )

            # X = NumPy 拼接((图像_批次, 生成的_图像))
            # y = NumPy 数组函数([1] * 批次_大小 + [0] * 批次_大小)

            # aux_y = NumPy 拼接((标签_批次, 采样的_标签))

            真实的_实际的_标签 = NumPy。数组函数([1] * 批次_大小)
            伪造_实际_标签 = np.array([0] * 批次_大小)

            轮次_判别器_真实_损失。追加(
                判别器。训练_于_批次(
                    图像_批次, [真实_实际_标签, 标签_批次]
                )
            )
            轮次_判别器_伪造_损失。追加(
                判别器。训练_于_批次(
                    生成的_图像, [伪造的_实际_标签, 采样的_标签]
                )
            )

            噪声 = np.random.normal(
                0, 0.02, (NUM_CLASSES * BATCH_SIZE, LATENT_DIM)
            )
            采样的_标签 = np.random.randint(
                0, NUM_类别 - 1    , NUM_类别 * 批次     _SIZE
            )

            技巧标签 = 名词全为 1 的数组(数值_类别 * 批次_大小)

            轮次_生成器_损失。追加(
                组合。训练_于_批次(
                    [采样_标签。重塑((-1, 1)), 噪声], [技巧标签,
采样_标签1
                )
            )

        print("\n正在测试第 {}:".format(epoch + 1))
        噪声 = np.random.normal(0, 0.02, (nb_测试, LATENT_DIM))

        采样的_标签 = np.random.randint(0, NUM_类别 - 1, nb_测试)
        生成的_图像 = 生成器。predict(
            [采样的_标签。重塑((-1, 1)), 噪声], verbose=假
        )
```

```
        # X = NumPy 拼接((x_测试, 生成的_图像))
        # y = NumPy 数组函数([1] * nb_测试 + [0] * nb_测试)
        # 辅助_y = NumPy 拼接((y_测试, 采样的_标签), 轴=0)

        真实_实际_标签 = NumPy数组函数([1] * 个_测试)
        假_实际_标签 = np.数组([0] * nb_测试)

        判别器_测试_真实_损失 = 判别器。评估(
            x_测试，[真实_实际_标签，y_测试]，详细程度=假
        )
        判别器_测试_假_损失 = 判别器。评估(
            生成的_图像，[假_实际_标签，采样的_标签]，详细程度=假
        )

        判别器_训练_真实_损失 = np.平均值(np.数组(epoch_判别器_真实_损失),
轴=0)
        判别器_训练_假样本_损失 = np.mean(np.array(epoch_disc_fake_loss),
轴=0)

        噪声 = np.random.normal(0, 0.02, (2 * nb_测试, LATENT_DIM))
        采样_标签 = np.random.randint(0, NUM_类别 - 1, NUM_类别 *
nb_测试)

        技巧标签 = np.ones(NUM_类别 * nb_测试)

        生成器_测试_损失 = 合并。评估(
            [采样得到的_标签。重塑((-1, 1)), 噪声],
            [技巧标签，采样得到的_标签],
            详细的=假      ,
        )

        生成器_训练_损失 = np.mean(np.array(epoch_gen_损失), axis=0)

        训练_历史["生成器"]。追加(生成器_训练_损失)
        训练_历史["判别器_真实"]。追加(判别器_训练_真实_损失)
        训练_历史["判别器_假"]。追加(判别器_训练_假_损失)

        测试_历史["生成器"]。追加(生成器_测试_损失)
        测试_历史["判别器_真实"]。追加(判别器_测试_真实_损失)
        测试_历史["判别器_假"]。追加(判别器_测试_假_损失)

        # 显示生成器损失
        标头 = "{0:<27秒} | {1:10秒} | {2:14秒} | {3:10秒} "。format(
            "组件", "总损失", "真实/伪造损失, 类别损失"
```

```
        )
        打印(标题)
        打印("-" * 长度(标题))
        行_格式 = "{0:<27s} | {1:10.2f} | {2:14.2f} | {3:10.2f} "

        打印(行_格式.格式化("生成器（训练）", *训练_历史["生成器"][-
1]))
        打印(ROW_FMT.format("generator (测试)", *测试_history["generator"][-1]))
        打印("-" * len(header))

        # 显示判别器损失
        标题 = "{0:<27s} | {1:10s} | {2:14s} | {3:10s} | {4:18s} |
{5:14s}".format(
            "组件",
            “总损失”      ,
            “真实/伪造损失”,
            "class loss",
            "真实/伪造准确率",
            "class 准确率    ",
        )

        打印（标题）
        打印("-" * 长度(标题))
        行_格式 = "{0:<27s} | {1:10.2f} | {2:14.2f} | {3:10.2f} | {4:18.2f} |
{5:14.2f}"

        打印(
            行_格式.格式化(
                "真实判别器（训练）",
*训练_历史记录["真实判别器_"][-1]
            )
        )
        打印(
            行_格式.格式化(
                "判别器生成样本（训练）",
*训练_历史["判别器_生成样本"][-1]
            )
        )
        打印(
            行_格式.格式化(
                "判别器真实样本（测试集）",
*测试_历史记录["判别器_真实"][-1]
            )
        )
        打印(
```

```
            行_格式.格式(
                "判别器生成样本（测试集）",
*测试_历史["判别器_虚假"][-1]
            )
        )
        打印("-" * 长度(标题))

        # 每 10 个 epoch 保存一次模型
        如果（轮次+ 1）% 10 == 0:
            检查点。保存(文件_前缀=检查点_前缀)
            pickle（序列化）转储（
                {"训练": 训练_历史记录，"测试": 测试_历史记录},
                打开(
                    os.path.拼接(训练_检查点_目录，"acgan-
history.pkl"),
                    "wb",
                ),
            )

        r，c = 2，2
        噪声 = np.random.normal(0，0.02，(r * c，LATENT_DIM))

        采样的_标签 = np.array([num for _ in range(r) for num in range(c)])
        采样的_标签 = 采样的_标签。重塑((-1，1))
        生成_图像 = 生成器。predict([采样的_标签，噪声])
        生成_图像 = 0.5 * 生成_图像 + 0.5

        图形，坐标轴 = plt.子图(r，c)
        计数 = 0
        对于 i 在 范围(r):
            对于 j 在 范围(c):
                # 保存音频
                梅尔频谱_逆变换 = 特征_到_音频(生成_图像[计数，:，:，:])
                sf.写入(
                    os.path.拼接(生成的_音频_路径，
f"轮次_{轮次}_{计数}.wav"),
                    mel_inv，
                    fs，
                )
                # 将图像添加到图表
                axs[i，j].显示图像(生成_图像[计数，:，:，0]，色彩映射="灰度")
                axs[i，j].axis("off")
                计数 += 1
        图。保存图像(os.path.拼接(generated_图像_路径，"%d.png" % 轮次))
        plt。关闭()
```

```
pickle.dump({"训练": train_训练历史, "测试": test_训练历史}, open(os.path.join(
```

10.4.1 训练

对于每个批次，都会生成图像批次以及这些频谱图各自对应的标签，随后要求生成器模型针对不同的类别标签生成频谱图。判别器使用真实图像及其标签进行训练，该训练过程中的损失被标记为真实损失，表示真实数据上的类别和验证损失。然后，使用生成的样本训练模型，并将测得的损失标记为虚假损失。每个周期结束后都会记录这两种损失并显示出来。

图 10.12：训练判别器并记录真实损失和虚假损失

```
        对于 索引 在 范围(nb_批次):
            进度_条。更新（索引）

            噪声 = np.random.normal(0, 0.02, (BATCH_SIZE, LATENT_DIM))


            图像_批次 = []
            对 _ 在 范围(BATCH_SIZE):
                try:
                    图像_批次。追加(下一个(x_训练))
                除了 StopIteration:
                    x_训练 = 加载_和_预处理_音频(x_训练_文件_路径)
                    图像_批次。追加(下一个(x_训练))
            图像_批次 = np.数组(图像_批次)

            标签_批次 = y_训练[索引 * 批次_大小:(索引 ) * 批次_大小]

            采样的_标签 = np.random.randint(0, 1, 批次_大小)

            生成的_图像 = 生成器。预测(
                [采样的_标签。重塑((-1, 1)), 噪声], 详细=0
            )

            # X = NumPy 拼接((image_batch, generated_images))
            # y = NumPy 数组函数([1] * BATCH_SIZE + [0] * BATCH_SIZE)

            # aux_y = NumPy 拼接((标签_batch, sampled_标签))
```

```
            真实_实际_标签 = np.array([1] * 批次_大小)
            虚假_实际_标签 = np.array([0] * 批次_大小)

            轮次_判别器_真实_损失。追加(
                判别器。训练_于_批次(
                    图像_批次，[真实_实际_标签，标签_批次]
                )
            )
            轮次_判别器_伪造_损失。追加(
                判别器。训练_在_批次(
                    生成的_图像，[伪造的_实际_标签，采样得到的_标签]
                )
            )
```

随后，在通过反向传播更新判别器的权重后，使用生成噪声和随机选取的标签训练复合模型。由于复合模型由生成器和判别器组成，噪声和类别标签会被转换为生成图像，再由判别器进行标记。由于两个模型彼此相连，判别器的损失会反向传播至生成器，而不会改变判别器的权重。由于判别器在生成器之前完成更新，该损失会提升生成器的性能。

图 10.13：训练生成器

```
            噪声 = np.random.normal(
                0，0.02，(NUM_类别 * BATCH_SIZE，LATENT_DIM)
            )
            已采样_标签 = np.random.randint(
                0，NUM_类别 - 1    ，NUM_类别 * BATCH    _SIZE
            )

            技巧标签 = np.ones(NUM_类别 * BATCH_SIZE)

            轮次_生成器_损失。追加(
                结合。训练_于_批次(
                    [采样的_标签。重塑((-1，1))，噪声]，[技巧标签，
已采样_标签]
                )
            )
```

10.4.2 验证

在每个训练轮次结束时，都会使用生成器和判别器此前均未见过的测试数据对二者进行测试。除不更新权重外，测试过程与训练过程完全相同。

图 10.14：验证生成器和判别器的性能

```
        打印("\n第 {} 轮测试：".格式化(第 + 1 轮))
        噪声 = np.random.normal(0, 0.02, (nb_测试, 潜在_维度))

        采样的_标签 = np.random.randint(0, NUM_CLASSES - 1, nb_测试)
        生成的_图像 = 生成器。预测(
            [采样的_标签。重塑((-1, 1)), 噪声], 详细=假
        )

        # X = np.concatenate((x_测试, 生成的_图像))
        # y = np.array([1] * nb_测试 + [0] * nb_测试)
        # 辅助_y = np.concatenate((y_测试, 采样的_标签), axis=0)

        真实_实际_标签 = np.array([1] * nb_测试)
        伪造_实际_标签 = np.array([0] * nb_测试)

        判别器_测试_真实_损失 = 判别器。评估(
            x_测试，[真实_实际_标签，y_测试]，verbose=假
        )
        判别器_测试_伪造_损失 = 判别器。评估(
            生成的_图像，[伪造_实际_标签，采样的_标签]，verbose=假
        )

        判别器_训练_真实_损失 = np.mean(np.array(epoch_disc_real_loss),
轴=0)
        判别器_训练_伪造_损失 = np.mean(np.array(epoch_disc_fake_loss),
轴=0)

        噪声 = np.random.normal(0, 0.02, (2 * nb_测试, LATENT_DIM))
        采样_标签 = np.random.randint(0, NUM_CLASSES - 1, NUM_CLASSES *
nb_测试)

        技巧标签 = np.ones(NUM_类别 * nb_测试)

        生成器_测试_损失 = 组合。评估(
            [采样的_标签。重塑((-1, 1)), 噪声],
            [技巧标签，采样的_标签],
            详细输出=假    ，
        )
```

```
        生成器_训练_损失 = np.mean(np.array(epoch_生成器_损失), axis=0)

        训练_历史["生成器"].追加(生成器_训练_损失)
        训练_历史["判别器_真实"].追加(判别器_训练_真实_损失)
        训练_历史["判别器_伪造"].追加(判别器_训练_伪造_损失)

        测试_历史["生成器"]。追加(生成器_测试_损失)
        测试_历史["判别器_真实"]。追加(判别器_测试_真实_损失)
        测试_历史记录["判别器_假"]。追加(判别器_测试_假_损失)
```

10.4.3 评估

最后，对 ACGAN 的性能进行逐步评估并展示，以确保训练过程顺利进行。此外，保存生成的样本并将其转换为音频，以便实时评估生成音频的性质。

图 10.15：展示 ACGAN 的总体损失并保存生成的样本

```
        # 显示生成器损失
        标题 = "{0:<27s} | {1:10s} | {2:14s} | {3:10s} "。格式(
            "组件", "总损失", "真实/伪造损失", "类别损失"
        )
        打印(header)
        打印("-" * 长度(header))
        行_格式 = "{0:<27s} | {1:10.2f} | {2:14.2f} | {3:10.2f} "

        打印(行_格式.格式("生成器（训练）", *训练_历史记录["生成器"][-
1]))
        打印(行_格式.格式化("生成器（测试）", *测试_历史["生成器"][-1]))
        打印("-" * 长度(header))

        # 显示判别器损失
        页眉 = "{0:<27s} | {1:10s} | {2:14s} | {3:10s} | {4:18s} |
{5:14s}"。格式(
            "组件",
            “总损失”      ,
            “真实/伪造损失”,
            "class loss",
            “真实/伪造准确率”,
            "class 准确率    ",
        )

        打印(标题)
        打印("-" * 长度(标题))
```

```
        行_格式 = "{0:<27s} | {1:10.2f} | {2:14.2f} | {3:10.2f} | {4:18.2f} |
{5:14.2f}"

        打印(
            行_FMT。格式(
                "判别器真实样本（训练）",
*训练_历史记录["判别器_真实"][-1]
            )
        )
        打印(
            行_格式.格式化(
                "判别器生成样本（训练）",
*训练_历史记录["虚假判别器_虚假"][-1]
            )
        )
        打印(
            行_格式..格式(
                "判别器真实样本（测试集）",
*测试_历史["真实判别器_真实"][-1]
            )
        )
        打印(
            行_格式..格式(
                "判别器生成样本（测试集）",
*测试_历史记录["判别器_伪造"][-1]
            )
        )
        打印("-" * 长度(标头))

        # 每 10 个训练周期保存一次模型
        如果 (训练周期 + 1) % 10 == 0:
            检查点。保存(文件_前缀=检查点_前缀)
            pickle。转储(
                {"train": 训练_历史记录, "test": 测试_历史记录},
                打开(
                    os.path.join(训练_检查点_目录, "acgan-
history.pkl"),
                    "wb",
                ),
            )

        r, c = 2 , 2
        噪声 = np.random.normal(0, 0.02, (r * c, LATENT_DIM))

        采样得到的_标签 = np.数组([数量 对于 _ 在 范围(r) 对于 数量 在 范围(c)])
```

```
    sampled_标签 =  sampled_标签.reshape((-1, 1)) gen_imgs =  生成器.predict([sampled_标
```

11 系统实现

该项目的最终目标是打造一款新冠肺炎检测应用，利用咳嗽中的模式检测新冠肺炎。因此，我们开发了一个平台，其中嵌入了经过训练的卷积神经网络，可通过咳嗽检测新冠肺炎，如图 11.1 所示。

图 11.1：通过咳嗽进行新冠肺炎检测的平台主页



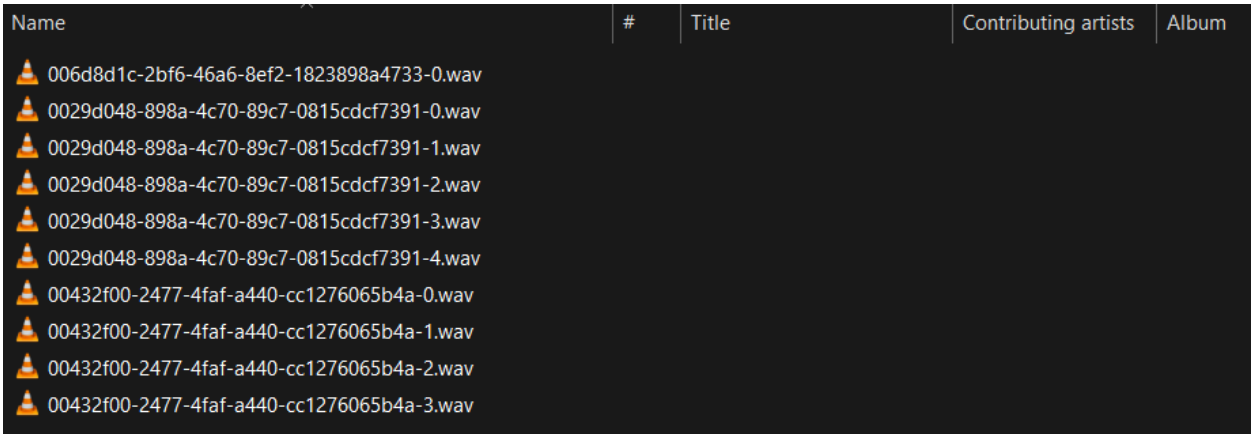
11.1 上传音频片段

如图 11.2 所示，平台中设有一个文件上传小组件，允许用户上传多个音频文件进行分类。点击“上传文件”后，系统会提示用户选择任意数量的音频文件，如图 11.3 所示。

图 11.2：设置为仅接受音频文件的上传文件小组件



图 11.3：选择用于分类的音频文件



11.2 查看已上传音频

上传所需的音频片段后，用户可以查看文件名，并在预测前实时播放音频文件，如图 11.4 所示。

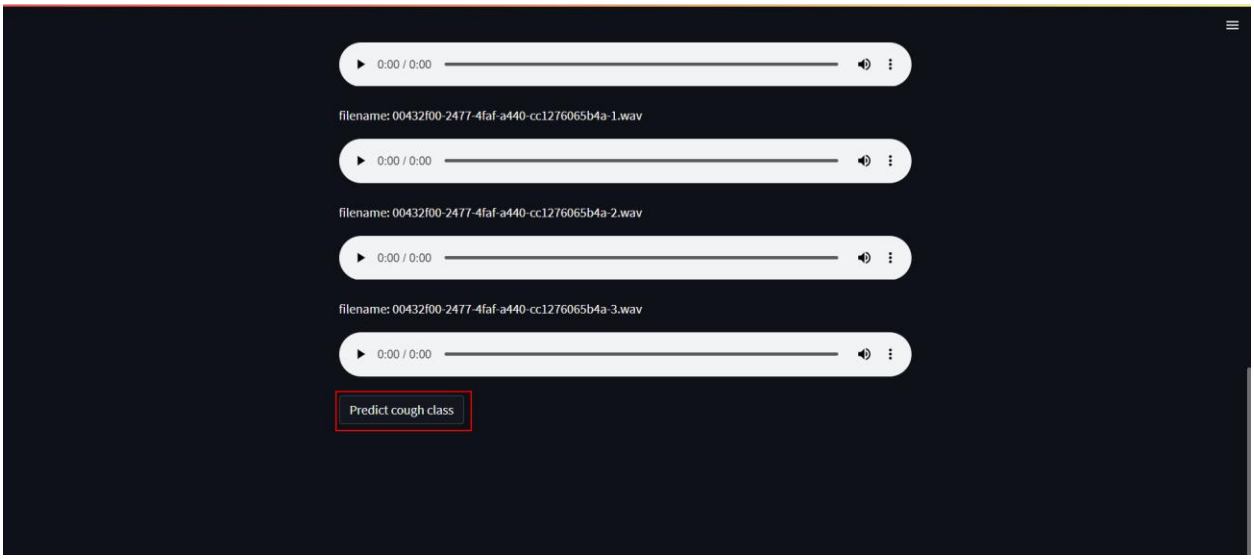
图 11.4：查看并播放上传的音频



11.3 音频样本分类

滚动到音频样本预览的末尾，用户可以找到一个名为“预测咳嗽类别”的按钮。点击后，后端会运行训练好的模型，对上传的音频样本进行分类并显示结果。

图 11.5：预测咳嗽类别按钮

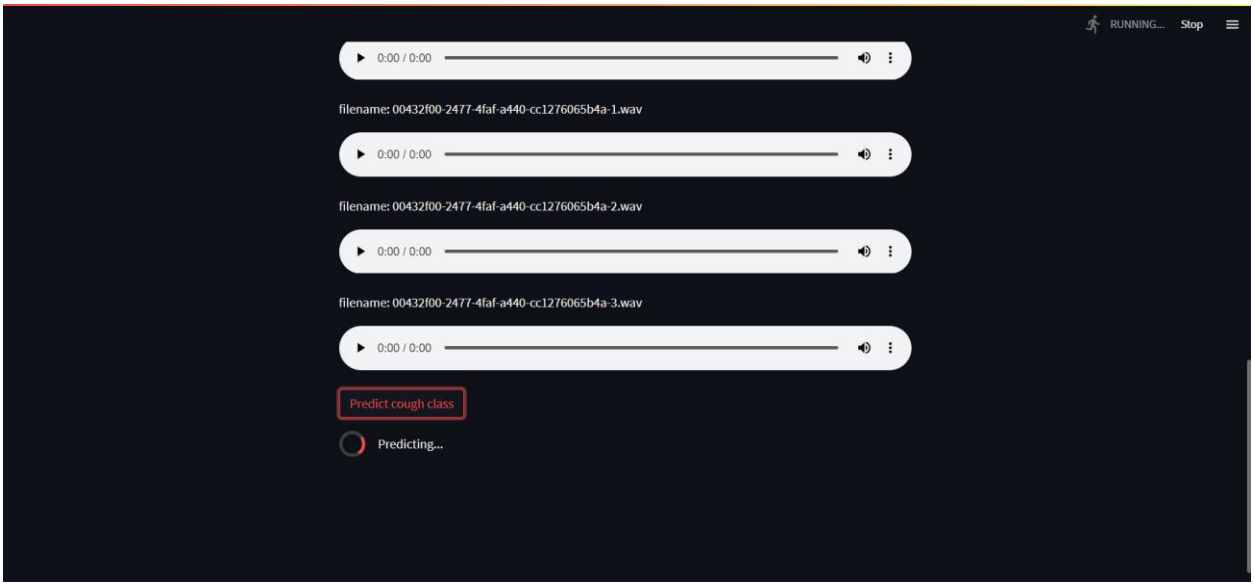


点击“预测咳嗽”按钮后，会出现以下两种情况之一。如果未上传音频样本，则会显示错误消息，如图 11.6 所示；否则，将运行模型，并在模型预测类别期间显示加载旋转图标，如图 11.7 所示。

图 11.6：错误消息，显示用户未上传音频片段



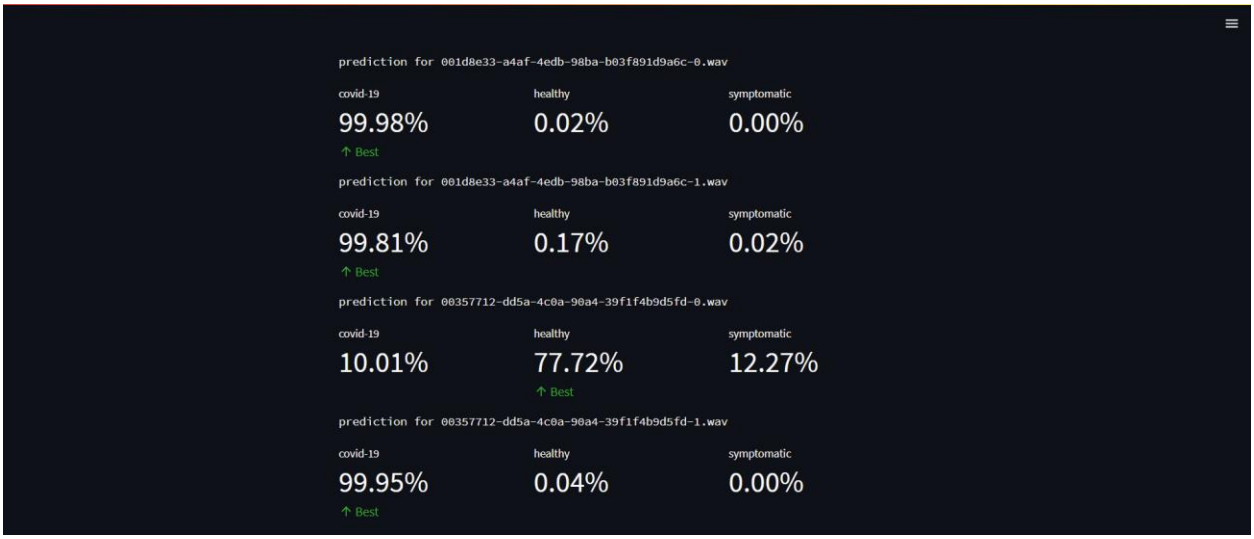
图 11.7：加载旋转图标，表示模型正在预测输出



11.4 查看模型预测结果

模型完成预测后，将显示每个已上传音频样本的预测结果。如图 11.8 所示，界面会显示音频属于三个类别之一的概率，并突出显示最有可能的选项。

图 11.8：显示模型预测结果



12 结论

过去两年，2019 冠状病毒病（COVID-19）的传播给医疗保健领域造成了灾难。面对医疗保健领域中另一个代表性不足的领域，许多研究致力于采用生成对抗网络（Goodfellow 等人，2014）来缓解 COVID-19 领域的数据稀缺问题。本文回顾了现有的所有利用生成对抗网络辅助 COVID-19 检测的研究。所讨论的文献将生成对抗网络用于多种不同任务，涵盖了生成对抗网络在图像数据上的诸多应用，例如图像合成、语义分割、分类和显著图生成。图像合成被用于数据集扩充和数据匿名化（Jiang 等，2020；Waheed 等，2020；Zunair 和 Hamza，2021a；Liu 等人，2020）。常见的方法是图像到图像翻译，即训练生成对抗网络将给定图像从 COVID-19 阳性域转换到阴性域。语义分割则用于肺部和病灶区域分割，以提升分类器的性能（Motamed 等人，2021；Mahapatra 和 Singh，2021；Menon 等人，2021；Liu 等人，2020）。大多数论文采用 U-Net 作为基础来开发分割模型，只有 Menon 等人（2021）使用了循环一致性。Motamed 等人（2021）的研究表明，语义分割能够持续提升图像合成任务的效果。同样，Liu 等人（2020）表明，使用合成数据扩充分割模型的数据集，并使合成数据呈现目标区域的空间概率图分布，能够改善分割图。因此，未来在这两类任务中的应用都应考虑结合另一类任务，以获得更好的结果。

分类生成对抗网络被用于直接对给定图像的标签进行分类（Waheed 等，2020；Motamed 等人，2021年）。尽管这一应用受到的关注较少，但它展示了生成对抗网络在半监督设置中的应用，这对于稀缺数据集而言是一种稳健的方法。显著图生成提供了对模型内部工作机制的洞察，并可用作合理性检查。（Dravid 等人，2021年）证明，生成对抗网络可以被利用来生成丰富的激活图，突出显示给定类别所特有的特征。这种方法在新领域和已知领域中都具有很大的潜力。激活图可用于确保模型关注的是正确的特征；在新领域中，这种表示可以提供对该领域的洞察，正如（Zunair 和 Hamza，2021b）所展示的那样。所提出的用于显著图生成的生成对抗网络还可以进一步改进，以生成双向的显著图，即从源到目标以及从目标到源。随后可以对生成的两张图进行比较，以获得更好的特征表示。

本文回顾了用于预测新冠肺炎传播的时间序列预测研究，其中生成对抗网络被用作基于智能体和 SEIRS 类型模型的计算高效替代方案，对环境的时空空间进行建模，并预测疾病传播情况（Silva等人，2021a；Silva等人，2021b）。虽然这一方法已在计算物理领域得到探索，但应用于病毒建模尚属新颖，可为预测病毒影响和帮助遏制传播提供更好的方法。

所有回顾的论文都使用放射影像数据，即胸部X光片（CXR）和计算机断层扫描（CT），作为模型的目标领域。图2展示了按所选数据划分的论文分布情况，结果表明，相关出版物转而更多使用CXR数据而非CT数据，这可能是因为通常更容易获取CXR图像；不过，同时采用这两种放射影像数据已成为常态。总体而言，放射影像数据得到了广泛采用，因为相较于逆转录聚合酶链式反应（RT-PCR），它能够可靠地呈现新冠肺炎症状的特征（Ai等人，2020；Fang等人，2020；Bernheim等人，2020；Chung等人，2020；Zhao等人，2020b；Kanne等人，2020；Guan等人，2020；Ai等人，2020；Huang等人，2020）。然而，RT-PCR和放射影像并非检测新冠肺炎的唯一可靠方法。许多出版物，例如（Darici 等人，2022；Sharma 等人，2021；Muguli等人，2021；Orlandic 等人，2021；Södergren等人，2021），都展示了通过患者咳嗽检测新冠肺炎的可喜成果。（Darici 等人，2022）针对利用咳嗽音频检测新冠肺炎这一任务测试了多个模型。研究发现，有限的数据集可能导致偏差，并得出结论：要提升模型性能，还需要更多数据。（Lella和PJA，2021）为这项研究收集了约五千个音频样本，其中仅有300个被标记为新冠肺炎阳性患者。尽管数据明显不平衡，仍有一些成果喜人的出版物问世，也举办了相关挑战赛。DiCOVA挑战赛的首届（Muguli等人，2021）和第二届（Sharma 等人，2021）旨在鼓励研究人员分析感染和未感染新冠肺炎者的录音。然而，与此前的数据集类似，各赛道所用的数据集也存在较大偏差：第一赛道的数据集仅有965个样本，其中172个为新冠肺炎阳性；第二赛道有471个样本，其中仅71个为新冠肺炎阳性（Sharma 等人，2021）。事实上，这种偏差也反映在参赛队伍的表现上；正如他们在论文中所述（Sharma 等人，2022），排名前四的队伍均未达到世界卫生组织提出的 ≥ 70% 标准（特异度为95%）。虽然简单组合这些队伍所开发模型的得分便可达到既定目标，但作者指出“未来仍有进一步发展的空间”（Sharma 等人，2021）。尽管这种新方法在新冠肺炎检测和应对数据不平衡方面具有潜力，但据我们所知，目前尚无已发表研究将生成对抗网络应用于这一特定子领域。

聚焦于所讨论的文献，不同作者并未采用统一的评估矩阵。相反，每篇论文都会在目标使用场景中应用所提出的生成对抗网络，以展示其有效性。尽管同一讨论部分中的各篇论文研究对象相同，但所提出的使用场景各不相同，进而导致评估方法也有所不同。例如，(Motamed 等人, 2021; Mahapatra 和 Singh, 2021; Menon 等人, 2021; Liu 等人, 2020) 的目标相同：在 COVID-19 领域开发用于肺部分割的分割模型；然而，各篇论文对其模型的评估方式各不相同。(Motamed 等人, 2021) 使用 Sørensen–Dice系数评估模型在 50 张未纳入训练的手动分割胸部X光片上的准确率；(Mahapatra 和 Singh, 2021) 将该模型与其他分割模型进行比较评估；(Menon 等人, 2021) 通过使用该模型扩增所选分类器的数据集来评估模型；而 (Liu 等人, 2020) 则提出了一种用于评估模型病灶纳入率的评估矩阵。

这种评估差异增加了将所提出的模型与其他已发表研究进行比较的难度，进而也使得确定最适合部署和未来改进的方法变得更加困难。同样，所讨论的图像合成论文均旨在通过用合成数据扩增给定数据集来提升分类器的性能，但评估方法各不相同。(Jiang 等，2020) 使用 FID、峰值信噪比、结构相似性指数和均方根误差，将图像质量与其他模型进行比较；通过逐步向 U-Net 添加合成数据并测量性能，评估生成数据的影响；并通过逐步向 U-Net 添加合成数据并测量性能，评估生成数据的影响；(Waheed 等，2020) 使用合成数据和不使用合成数据分别训练 VGG-16 分类器，并比较其性能；而 (Zunair 和 Hamza，2021a) 使用合成数据和不使用合成数据分别训练 VGG-16、ResNet-50、DenseNet-102 和 (DenseNet121 + BGT)，并比较其性能。尽管每篇论文都通过在扩增后的数据集上训练分类器来评估其模型，但分类器架构和模型配置的选择各不相同，因此仍存在缺乏统一评估矩阵所导致的问题。为此，需要一种基于算法的评估矩阵，以客观地评估针对每项特定任务设计的生成对抗网络。

本研究探讨了生成对抗网络如何利用咳嗽辅助新冠肺炎检测。现有文献显示，使用生成对抗网络合成胸部 X 光影像和 CT 扫描取得了令人鼓舞的结果。此外，现有用于检测新冠肺炎的分类器也表现出良好效果，但有研究报告指出，数据不平衡会导致分类偏差。为缓解这种偏差，本研究开发了一种生成对抗网络，用于扩增和合成新冠肺炎咳嗽数据，以扩充现有数据集，并帮助分类器更好地检测新冠肺炎。

参考文献

Adewumi, T., Alkhaled, L., Alkhaled, H., Liwicki, F. 和 Liwicki, M., 2022年。ML_LTU 在 SemEval-2022 任务 4 中的研究：T5 助力识别谄媚与居高临下的语言。arXiv 预印本 arXiv:2204.07432。Alkhaled, L., Adewumi, T. 和 Sabry, S.S., 2023年。Bipol：一种具有可解释性的 NLP 多轴偏差评估新指标。arXiv 预印本 arXiv:2304.04029。

Al-Masri, A.N.A. 和 Mokayed, H., 2021年。基于深度学习特征提取和粒子群支持向量机状态识别的齿轮智能故障诊断。《智能系统与物联网学报》，4(1)，第26–40页。

Arjovsky, M.、Chintala, S.、Bottou, L.，2017年。《Wasserstein 生成对抗网络》。arXiv 预印本，arXiv:1701.07875。Ben-Zvi, N.（2022年1月3日）。《面向2022年的深度学习硬件指南——Towards Data Science》。Medium。检索日期：2022年1月5日。
h
t
t
p
s://
to
wardsdatascience.com/another-deep-learning-hardware-guide-73a4c35d3e86Brownlee J.（2019年）。《生成对抗网络（GAN）的入门介绍》。[在线]Machine Learning Mastery。网址：https://machinelearningmastery.com/what-are- generative-adversarial-networks-gans/。Costa, P.、Galdran, A.、Meyer, M. I.、Niemeijer, M.、Abramoff, M.、Mendonca, A. M. 和 Campilho, A.（2018年）。《端到端对抗性视网膜图像合成》。IEEE《医学影像学汇刊》，37(3)，781–791。https://doi.org/10.1109/tmi.2017.2759102Darici, E.、Rasmussen, N.、Xiao, J.、Chaudhari, G.、Rajput, A.、Govindan, P.……和 Pilanci, M.（2022年）。《利用大型聚合数据集和深度学习进行基于咳嗽的新冠肺炎分类》。arXiv 预印本，arXiv:2201.01669。Diwan, S. S.、Ravichandran, S.、Govindarajan, R. 和 Narasimha, R.（2020年）。《通过对咳嗽/打喷嚏气流进行直接数值模拟，理解新冠肺炎类感染的传播动力学》。印度国家工程院学报，5(2)，255–261。Donahue, C.、McAuley, J. 和 Puckette, M.（2018年）。《对抗性音频合成》。arXiv 预印本，arXiv:1802.04208。

Doshi, K.（2021年12月31日）。《音频深度学习入门（第 2 部分）：梅尔频谱图为何表现更佳》。Medium。https://towardsdatascience.com/audio-deep-
l
e
a
rning-
made-simple-part-2-why-mel-spectrograms-perform-better-aad889a93505 Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. 和 Bengio, Y.（2014）。《生成对抗网络》。获取地址：https://arxiv.org/pdf/1406.2661.pdf。Great Learning Team。（2021年9月21日）。为什么 Python 如此受人工智能和机器学习领域青睐？GreatLearning 博客：助你规划职业生涯的免费资源！检索日期：2022年2月28日，来源：https://www.mygreatlearning.com/blog/why-is-python-so-popular-for-ai-and-machine-learning/ Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A., 2017。《改进 Wasserstein 生成对抗网络的训练》。arXiv 预印本 arXiv:1704.00028。Horev R.（2019）。《基于风格的生成对抗网络——生成和调整逼真的人工人脸》|Lyrn.AI。[在线] 获取地址：https://www.lyrn.ai/2018/12/26/a-style-based-generator-architecture-for-generative-adversarial-networks/。Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A., 2016。《基于条件对抗网络的图像到图像翻译》。arXiv 预印本 arXiv:1611.07004。Khan, M.A.U., Nazir, D., Pagani, A., Mokayed, H., Liwicki, M., Stricker, D. 和 Afzal, M.Z., 2022。《深度补全方法综合综述》。Sensors, 22(18), p.6969。Kanchi, S., Pagani, A., Mokayed, H., Liwicki, M., Stricker, D. 和 Afzal, M.Z., 2022。《EmmDocClassifier：面向稀缺数据的高效多模态文档图像分类器》。Applied Sciences, 12(3), p.1457。Lella, K. K., & PJA, A.（2021）。《利用一维卷积神经网络并基于咳嗽、呼吸和语音等参数，通过人类呼吸音增强数据进行新冠肺炎自动诊断》。AIMS Public Health, 8(2), 240–264。https://doi.org/10.3934/publichealth.2021019 Maier, A.（2020）。《我们能否解决医疗应用中的数据短缺问题？》[在线
]
Medium。访问地址：https://towardsdatascience.com/will-we-ever-solve-the-shortage-医疗应用中的数据-70da163e2c2d [访问日期：12月6日。2021]

Mehdi Mirza 和 Simon Osindero。条件生成对抗网络。arXiv 预印本 arXiv:1411.1784，2014年。Khalid, M., Yusof, R. 和 Mokayed, H.，2011年。利用模糊逻辑推理融合多个分类器，实现在线签名验证。《国际创新计算期刊》，7(5)，第2709-2726页。Mokayed, H. 和 Mohamed, A.，2014年。基于序数结构模糊逻辑的通用结构化文档分类器稳健阈值化技术。《国际创新计算、信息与控制期刊》，10(4)，第1543-1554页。Mokayed, H., Shivakumara, P., Woon, H.H., Kankanhalli, M., Lu, T. 和 Pal, U.，2021年。一种用于无人机图像中的车牌号码检测的新型DCT-PCM方法。《模式识别快报》，148，第45-53页。Mokayed, H., Quan, T.Z., Alkhaled, L. 和 Sivakumar, V.，2022年10月。基于深度学习计算机视觉技术的实时人体检测与计数系统。载于《人工智能与应用》。Mokayed, H., Palaiahnakote, S., Alkhaled, L. 和 AL-Masri, A.N.，2022年11月。无人机图像中的车牌号码检测。载于《人工智能与应用》。Mokayed, H., Clark, T., Alkhaled, L., Marashli, M.A. 和 Chai, H.Y.，2022年12月。受限计算系统上可实现实时多目标跟踪与对象识别任务。载于《2022年IEEE工业工程国际会议》

d

工程管理（IEEM）（第1523-1528页）。IEEE。Mokayed, H., Palaiahnakote, S., Alkhaled, L. 和 AL-Masri, A.N.，2022年11月。无人机图像中的车牌号码检测。载于《人工智能与应用》。Muguli, A., Pinto, L., Sharma, N., Krishnan, P., Ghosh, P. K., Kumar, R., ... & Nanda, V.（2021年

).

DiCOVA挑战赛：使用声学进行新冠肺炎诊断的数据集、任务和基线系统。arXiv预印本 arXiv:2103.09148。Nation, D.（2019年12月11日）。机器学习的最佳编程语言是什么？Medium。检索于2022年2月28日，来自

h

ttps://to

wardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7Ng, H.S., Tay, Y.H., Liang, K.M., Mokayed, H. 和 Hon, H.W., 2015。基于 SIFT 特征的马来西亚特殊车牌检测与识别。arXiv预印本 arXiv:1504.06921。

Nikolaidou, K.、Seuret, M.、Mokayed, H. 和 Liwicki, M.，2022年。《历史文档图像数据集综述》。《国际文档分析与识别期刊》（IJDAR），25(4)，第305–338页。Odena, A.、Olah, C. 和 Shlens, J.（2017年7月）。《基于辅助分类器生成对抗网络的条件图像合成》。载于《国际机器学习会议》（第2642–2651页）。PMLR。

.

Orlandic, L.、Teijeiro, T. 和 Atienza, D.（2021年）。COUGHVID众包数据集：用于研究大规模咳嗽分析算法的语料库。《Scientific Data》，8(1)，1–10。Orlandic, L.、Teijeiro, T. 和 Atienza, D.（2021年）。COUGHVID众包数据集：用于研究大规模咳嗽分析算法的语料库。《Scientific Data》，8(1)。https://doi.org/10.1038/s41597-021-00937-4Pierre, S.（2021年7月13日）。人工智能如何彻底改变医疗保健。《Built In》。
h
t
t
ps://builtin.co
m/healthcare-technology/how-ai-revolutionize-healthcarePinhasi, A.（2021年12月13日）。迈向机器学习开发方法论——第一部分。Medium。https://assaf-pinhasi.medium.com/towards-a-development-methodology-for-machine-learning-part-i-f1050a0bc607Rago, F.（2018年）。组织内人工智能软件开发方法论。组织内人工智能软件开发方法论。Rohner, T. 和 Netguru（2021年5月27日）。Python 与 C++ 在机器学习中的比较。Netguru。检索于2022年2月28日，网址：https://www.netguru.com/blog/cpp-vs-pythonr-project.（无日期）。R：什么是 R？R项目。检索于2022年2月28日，网址：https://www.r- project.org/about.htmlS, S.（2021年5月25日）。人工智能在医疗保健领域面临的挑战与未来发展。《Analytics India Magazine》。检索于2021年12月26日，网址：https://analyticsindiamag.com/challenges-and-future-of-ai-in-healthcare/Senyard, A.、Kazmierczak, E. 和 Sterling, L.（2004年）。神经网络软件工程方法。神经网络软件工程方法。

Sharma, N. K., Chetupalli, S. R., Bhattacharya, D., Dutta, D., Mote, P., & Ganapathy, S.（2021年）。第二届DiCOVA挑战赛：使用声学进行新冠肺炎诊断的数据集与性能分析。

Sharma, N. K.、Muguli, A.、Krishnan, P.、Kumar, R.、Chetupalli, S. R. 和 Ganapathy, S.（2022年）。利用生成对抗网络进行新冠肺炎的基于声音的检测测试：首届基于声学的新冠肺炎诊断（DiCOVA）挑战赛总结。《计算机语音与语言》，73，101320。https://doi.org/10.1016/j.csl.2021.101320

Sharma, N., Krishnan, P., Kumar, R., Ramoji, S., Chetupalli, S. R., Ghosh, P. K., & Ganapathy, S. (2020). Coswara——用于新冠肺炎诊断的呼吸、咳嗽和语音数据库。arXiv 预印本 arXiv:2005.10548。

Stack Overflow。（2021年）。Stack Overflow 2021年开发者调查。检索于2022年2月28日，来源：https://insights.stackoverflow.com/survey/2021

van de Wolfshaar, J.（2018年6月25日）。使用生成对抗网络进行半监督学习——Jos van de Wolfshaar。《Medium》。https://medium.com/@jos.vandewolfshaar/semi-supervised-learning-with-gans-23255865d0a4

Voon, W., Hum, Y.C., Tee, Y.K., Yap, W.S., Salim, M.I.M., Tan, T.S., Mokayed, H. 和 Lai, K.W.，2022年。采用迁移学习的七种卷积神经网络（CNN）在乳腺组织病理图像中浸润性导管癌（IDC）分级任务上的性能分析。《科学报告》，12(1)，第19200页。

Y. J. Tan、K. S. Sim 和 F. F. Ting，“使用卷积神经网络进行乳腺X线成像系统的乳腺癌检测”，2017年国际机器人、自动化与科学会议（ICORAS），2017年，第1-5页，doi: 10.1109/ICORAS.2017.8308076。

Yadav, A., Shah, S., Xu, Z., Jacobs, D., Goldstein, T., 2018年。使用预测方法稳定对抗网络。Yi, X., Walia, E., & Babyn, P. (2019年)。生成对抗网络在医学影像中的应用：综述。《医学图像分析》，58，101552。https://doi.org/10.1016/j.media.2019.101552

Yoon, J.、Jarrett, D. 和 Van Der Schaar, M.（2019年）。时间序列生成对抗网络。Zhu, J. Y.、Park, T.、Isola, P. 和 Efros, A. A.（2017年）。使用循环一致对抗网络进行非配对图像到图像转换。载于 IEEE 国际计算机视觉会议论文集（第 2223-2232 页）。Zindi。（2020年3月4日）。Zindi.Africa。检索于2022年1月5日，网址：https://zindi.africa/learn/what-to-consider-when-choosing-a-laptop-for-machine-learning