得分：

《人工智能与机器学习》课程大作业

项目名称：	基于卷积神经网络的自训练人脸识别 BMI 模型的 Web 应用	工作量分配[1]：总量为 10
组长（含学号）：	陈晓天	+0.6
组员名单（含学号）：	卢文秀	-0.3
	张余蝶	-0.3
项目简介：	本项目旨在设计并实现一个基于深度学习的端到端系统，通过分析用户上传的单张面部照片，来预测其身体质量指数（BMI）。项目覆盖了从网络数据自动爬取与清洗、多种模型结构的探索与迭代、在云端 GPU 平台上的训练与调优，到最终使用 Flask 框架将模型封装为可交互的 Web 应用的完整流程。项目核心在于探索面部视觉特征与人体 BMI 之间的非线性关系，并最终提供一个直观、易用的成果展示界面。	
项目全体成员签名：	陈晓天 卢文秀 张余蝶	

[1] 全组工作量分配为+x，或-x。要求和为 0，如果分组同学工作量相差过大，需有书面详细说明。

目录

一、问题描述

1.1 项目背景与意义

身体质量指数（Body Mass Index, BMI）是目前国际上常用的衡量人体胖瘦程度以及是否健康的一个标准。它通过身高和体重进行计算，是评估超重和肥胖等健康风险的重要指标。传统的 BMI 测量需要明确的身高和体重数据，但在许多场景下，我们可能无法直接获取这些信息。随着计算机视觉和人工智能技术的飞速发展，从图像中提取和分析人体生物特征成为可能。

本项目旨在探索一个有趣且具有挑战性的问题：我们能否仅通过一张面部照片来估算一个人的 BMI？如果能够实现，这项技术在健康评估、个性化推荐、娱乐应用等领域将具有潜在的应用价值。它将传统的、需要数据输入的健康评估方式，转变为一种更加直观、便捷的视觉感知过程。因此，本项目的核心任务是构建一个深度学习模型，挖掘人脸图像中与 BMI 相关的细微视觉线索，并实现一个从图像输入到 BMI 值输出的完整预测系统。

1.2 主要挑战与难点

在项目初期，我们预见到实现这一目标面临着几个核心的挑战与难点：

首先是数据获取的挑战，目前并没有公开的大规模、标注好的人脸-BMI 数据集。要训练一个有效的深度学习模型，高质量的数据是前提。因此，如何从零开始构建一个可用的数据集是我们需要解决的首要问题。

其次是特征的微妙性与复杂性，与常规的物体识别（如猫和狗）不同，BMI 在人脸上的视觉特征非常微妙。它可能体现在脸颊的丰满度、下颌线的清晰度、双下巴的程度等多个方面，并且这些特征受到个体骨相、年龄、性别、种族等多种因素的干扰。模型需要有足够的能力来学习这些高度非线性的、微弱的关联。

再次是现实场景的复杂性，真实世界中的人脸照片存在各种干扰因素，如光照变化、拍摄角度、面部表情、妆容、发型遮挡等。这些因素都可能成为模型的“噪音”，影响预测的准确性。模型必须具备一定的鲁棒性，才能应对这些变化。

最后是模型设计的探索性，针对这一特定任务，是应该使用在大型通用数据集（如 ImageNet）上预训练的模型进行迁移学习，还是从零开始设计一个更具针对性的轻量级模型？这没有一个确定的答案，需要通过实验来探索和验证。

1.3 相关研究调研

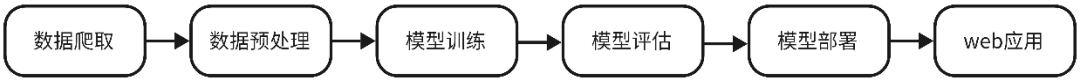
在项目启动前，我们进行了一些相关的文献和技术调研。我们发现，利用计算机视觉进行人体参数估算已有不少研究，例如从姿态估计身高、从视频步态分析健康状况等。具体到面部特征分析，深度学习模型（特别是卷积神经网络 CNN）在人脸识别、年龄估计、性别分类等任务上已经取得了巨大成功。这些研究表明，CNN 能够有效提取人脸的深层语义特征。虽然直接预测 BMI 的研究相对较少，但这些成功的先例给了我们信心，证明通过深度学习模型从人脸图像中学习特定

属性是完全可行的。我们的工作将站在这些技术的基础上，专注于 BMI 预测这一特定且更具挑战性的回归任务。

二、技术方案

2.1 整体系统架构

为了系统性地解决上述问题，我们设计了一个包含数据、模型、后端和前端四个核心部分的端到端系统。整体架构如下图所示，它清晰地展示了从数据源到最终用户交互的完整流程。



该架构遵循了典型的机器学习项目生命周期。

数据层负责数据的获取和管理。

处理层负责数据的清洗、增强和格式化，使其适用于模型训练。

模型层是项目的核心，负责模型的训练、验证和调优。

应用层将训练好的模型封装成服务，并通过 Web 界面提供给用户使用。

2.2 模块划分与技术选型

根据整体架构，我们将项目划分为以下几个关键模块，并为每个模块选择了成熟且高效的技术工具。

1、数据获取模块

任务：从互联网上爬取带有 BMI 或身高体重信息的“减肥前后”对比图。

技术选型：使用 Python 的 praw 库。这是一个功能强大的 Reddit API 封装库，可以方便地访问 Reddit 论坛（如 r/progresspics）中的帖子标题、内容和图片链接，这正是我们理想的数据来源。

2、数据预处理模块

任务：对原始图片进行人脸检测、裁剪、清洗和标注，生成干净的训练数据。

技术选型：

人脸检测：采用 facenet-pytorch 库，它内置了高效的 MTCNN（Multi-task Cascaded Convolutional Networks）人脸检测模型，能够准确地定位并裁剪出图片中的人脸区域。

数据管理：使用 pandas 库来处理和存储图片路径与对应 BMI 标签的 csv 文件，方便后续的数据加载。

图像处理：使用 Pillow (PIL) 和 torchvision.transforms 库进行图像的读取、裁剪、尺寸变换、数据增强（如随机翻转、颜色抖动）和归一化等操作。

3、模型训练与评估模块

任务：设计、训练和评估深度学习模型，使其能够准确预测 BMI。

技术选型：

深度学习框架：选用 PyTorch。它以其灵活性、易用性和强大的社区支持而

闻名，非常适合进行快速的模型研究和迭代。

计算环境：由于本地计算机 CPU 训练速度过慢，我们选择在 AutoDL 云平台上租用带 NVIDIA RTX 3090 GPU 的实例进行训练，极大地加速了实验进程。

模型结构：初期尝试了基于 ResNet 的迁移学习方案，但效果不佳。最终，我们自行设计并实现了一个轻量级的卷积神经网络（SimpleCNN），并从零开始训练。

4、Web 应用部署模块

任务：将训练好的模型封装成一个后端 API，并开发一个用户友好的前端界面。

技术选型：

后端框架：使用 Flask。它是一个轻量级的 Python Web 框架，非常适合快速地将机器学习模型包装成 RESTful API。

前端技术：使用标准的 HTML, CSS, 和 JavaScript。无需复杂的前端框架，即可实现文件上传、图片预览和结果动态展示等功能，保持项目的简洁性。

2.3 团队成员分工

为了高效地推进项目，我们进行了明确的分工：

陈晓天(组长)：负责阅读论文调研模型，负责项目的整体技术路线规划与核心架构设计与实现，主导核心模型的探索、设计、训练、调优与评估。主要负责数据的爬取与清洗，模型的设计与实现，问题困难的分析与决策，整合报告文档。

卢文秀：主要负责项目辅助与前端界面实现工作，同时做各项实验分析数据以及模型可行性验证。在项目初期，进行了部分原始数据的收集与手动筛选验证；在项目后期，根据设计方案，负责了 Web 前端界面的 HTML 结构搭建与 CSS 样式实现，并参与了最终的界面效果测试；做实验验证了 SimpleCNN 的可用性。同时负责部分报告资料的撰写整理与排版美化。

张余蝶：主要负责项目测试工作与后端编程工作，同时做部分实验分析模型异常的原因。在数据处理阶段，协助运行数据预处理脚本并对生成的干净数据集进行了抽样检查，确保数据质量；在应用开发阶段，协助对后端 API 接口的功能与性能进行了多轮测试，并验证了 Web 应用的整体流程。同时负责部分报告资料的撰写整理与排版美化。

三、具体实施

本章节将详细阐述项目的具体执行过程，包括我们遇到的问题、进行的探索，以及最终的实现细节。

3.1 数据获取与预处理

项目的第一步是构建数据集。我们使用 praw 库编写了 scraper.py 脚本，目标是 Reddit 的 r/progresspics 子版块。该版块的用户会分享自己减肥前后的对比照片，并通常在标题中注明自己的身高、初始体重和当前体重，这为我们计算 BMI

提供了可能。脚本自动化地遍历帖子，解析标题中的数据，并下载对应的图片。经过约 8 分钟的运行，我们初步获取了近 2000 张图片和对应的元数据，保存在 reddit_images_large 文件夹和 reddit_images_bmi_large.csv 文件中。

然而，原始数据是嘈杂且不规范的，无法直接用于训练。为此，我们编写了 data_preprocessing.py 脚本进行处理，主要步骤如下：

人脸检测：我们遍历每一张下载的图片，使用预先初始化的 MTCNN 模型检测其中的人脸。MTCNN 非常强大，但对于一些角度刁钻或清晰度低的图片，仍然可能检测失败或检测到多张人脸。

数据清洗：

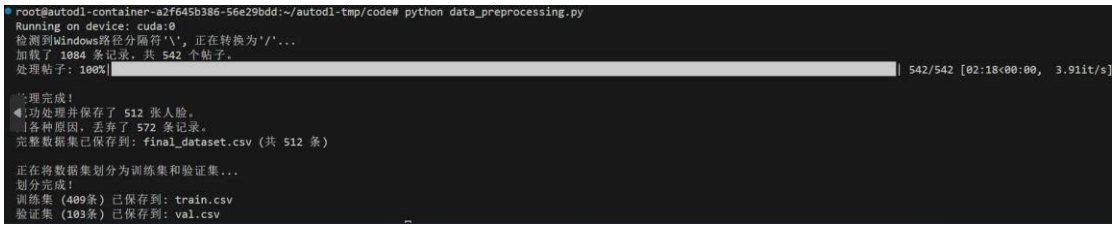
对于未能检测到人脸的图片，我们直接丢弃。

对于检测到多张人脸的图片，由于无法确定哪一张是发帖者本人，我们也予以丢弃，以保证标签的准确性。

对于“减肥前后”的对比图，我们只保留了处理后（通常是右侧或下方）的单张人脸图及其对应的当前 BMI。

数据保存与标注：成功检测并裁剪出的人脸被统一保存到 processed_faces 文件夹中。同时，我们生成了一个最终干净的数据集描述文件 final_dataset.csv，该文件每行包含一张处理后的人脸图片路径和其对应的 BMI 标签。

经过这一系列严格的清洗流程，我们从近 2000 张原始图片中筛选出了 512 张高质量的、带清晰人脸和准确 BMI 标签的图片。虽然数量不多，但数据的纯净度得到了保证。最后，我们将这 512 个样本按照 8:2 的比例随机划分为训练集（409 张）和验证集（103 张），分别生成 train.csv 和 val.csv，为模型训练做好了准备。



3.2 训练环境配置与探索

在模型训练初期，我们尝试在本地计算机上使用 CPU 进行训练。很快我们便发现，即使是一个相对简单的 CNN 模型，在 CPU 上的训练速度也慢得令人难以忍受，每个 epoch 都需要数分钟，这使得快速迭代和调参成为不可能。

为了解决这个瓶颈，我们决定转向云端 GPU 平台。经过比较，我们选择了 AutoDL，因为它提供了便捷的深度学习环境和弹性的计费方式。我们租用了一台配备 NVIDIA RTX 3090 (24GB 显存) 的主机，并选择了预装好 PyTorch 2.5.1 和 CUDA 12.4 的镜像。通过 SSH 连接到云主机后，我们使用 pip 安装了项目所需的其他依赖库，如 facenet-pytorch, pandas, scikit-learn 等。

切换到 GPU 环境后，训练速度得到了质的飞跃。之前需要几分钟的 epoch 现在仅需几秒钟即可完成，这为我们后续进行大量的模型结构探索和超参数调整提供了可能，极大地提高了项目的研发效率。

3.3 模型探索与迭代的曲折之路

模型部分是整个项目中探索过程最长、最曲折的一环。我们的目标是找到一个能够有效学习人脸与 BMI 关系的模型，这个过程充满了试错与反思。

第一阶段：失败的迁移学习尝试

我们最初的设想是利用迁移学习。我们认为，在 ImageNet 这种大型数据集上预训练的 ResNet18 模型已经学习到了丰富的通用图像特征（如边缘、纹理、形状），我们只需在此基础上进行微调，修改其最后的分类头为回归头，就可以快速适应我们的 BMI 预测任务。

然而，实验结果却不尽人意。模型在训练集上收敛缓慢，且在验证集上的损失（Loss）居高不下，表现甚至不如随机猜测。经过分析，我们推测失败的原因可能有两点：其一，ImageNet 的特征（如动物皮毛、汽车轮廓）与我们任务所需的微妙面部特征（如脸颊饱满度）差异过大，预训练权重可能是一种“知识的诅咒”；其二，我们的数据集规模太小（仅约 400 张训练图片），不足以有效地对 ResNet 这样的大型网络进行微调，容易导致灾难性遗忘或过拟合。因此，我们果断放弃了迁移学习的方案。

第二阶段：自定义 CNN 与混乱的调参过程

既然迁移学习走不通，我们决定从零开始设计并训练一个更小、更具针对性的 SimpleCNN 模型。该模型由几个卷积层、激活函数（ReLU）、批归一化（BatchNorm）和池化层堆叠而成，最后接一个由全连接层和 Dropout 层组成的回归头。

然而，即便是这个简单的模型，训练过程也一波三折。我们经历了数轮痛苦的调参：

初步尝试：使用较小的学习率（1e-4）和批大小（32），我们发现模型的 loss 从一个极高的初始值（超过 1000）迅速下降，但很快就停滞在一个较高的水平，验证集 MAE（平均绝对误差）高达 6.4。这表明模型学到了一些基本模式，但远未达到理想状态。

调参 1（降低学习率，增大批大小）：我们将学习率降至 1e-5，批大小增至 64。结果非常糟糕，loss 几乎不下降，最终高达 500 多。这说明学习率过低，模型无法有效更新权重。

调参 2（保持低学习率，减小批大小）：我们保持 1e-5 的学习率，将批大小恢复到 32。情况有所好转，loss 最终降至 250 左右，但依然很差。

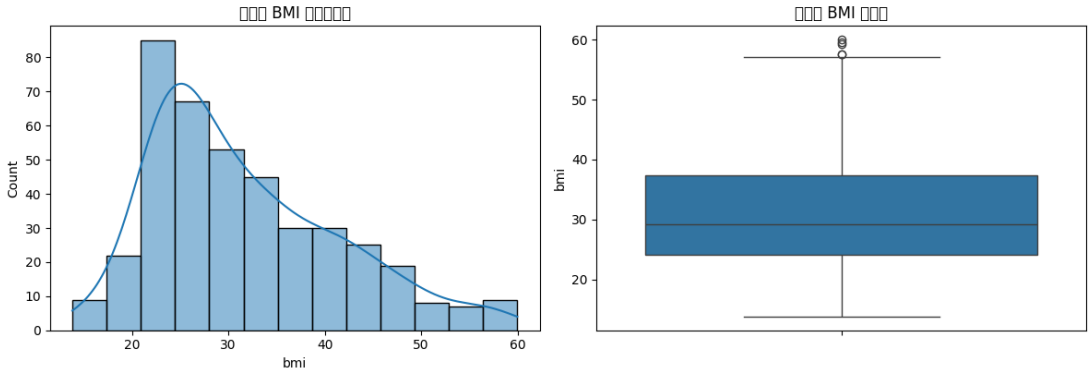
一个奇怪的现象：我们注意到，loss 的初始值非常高，且训练速度飞快。这让我们一度怀疑是 DataLoader（数据加载器）出了问题，比如标签和数据没有正确对应。为此，我们专门编写了调试代码，从 DataLoader 中取出一个批次的数据进行检查，发现图片张量和 BMI 张量的形状、数值都完全正常。这排除了数据加载的问题，让我们重新将焦点放回模型和训练策略本身。

第三阶段：从数据和损失函数中寻找突破口

在多次调参失败后，我们意识到问题可能不在于超参数的细微调整，而在于更根本的方面。我们编写了 data_test.py 脚本，对我们干净数据集的 BMI 分布进行了可视化分析。

图表如下：

□□□□□□□



分析图表后我们发现，虽然数据整体呈类正态分布，符合真实世界规律，但存在一些影响巨大的高值异常点（outliers），例如 BMI 超过 50 的个体。我们使用的默认损失函数是均方误差（MSE, or L2 Loss），它会对误差进行平方处理。这意味着，模型在预测这些异常点时，一个较大的误差会被平方放大成一个巨大的 loss，从而主导整个模型的梯度更新方向。模型会“拼命”去拟合这些少数的异常点，而忽略了对大部分正常样本的学习。

这正是我们找到的突破口！为了降低异常点的影响，我们决定将损失函数从 MSE 更换为对异常值更鲁棒的 Huber Loss（在 PyTorch 中实现为 SmoothL1Loss）。当误差较小时，它的表现类似 MSE；当误差较大时，它的表现类似 MAE（L1 Loss），惩罚是线性的，从而避免了损失的爆炸。

3.4 最终模型设计与训练

在确定了损失函数这一关键改进后，我们整合了之前的经验，设计了最终的训练方案 final_train.py。这个方案是一个集大成者，包含了我们认为对任务最有效的一系列技术：

1、模型结构：仍然使用我们设计的 SimpleCNN，但我们对权重进行了 Kaiming He 初始化。这是一种更先进的初始化方法，有助于缓解梯度消失/爆炸问题，让网络在训练初期就能更好地学习。

2、损失函数：采用 SmoothL1Loss (Huber Loss)，以增强模型对数据集中异

常点的鲁棒性。

3、优化器：选用 AdamW。相比传统的 Adam，AdamW 改进了权重衰减（Weight Decay）的实现方式，能够更有效地抑制过拟合。

4、学习率调度器：使用 CosineAnnealingLR。它可以在整个训练周期内，将学习率按照余弦曲线平滑地从初始值降至一个很小的值。这种策略被证明有助于模型跳出局部最优，找到更平坦、泛化能力更好的最小值。

5、数据增强：在训练数据加载时，我们加入了适度的随机数据增强，包括随机水平翻转和颜色抖动。这相当于在不增加新数据的情况下，扩充了训练集的多样性，能有效提高模型的泛化能力。

我们使用了一套经过验证较为合理的超参数（如学习率 3e-4，批大小 32，训练 200 轮）来运行最终的训练。这一次，训练过程非常顺利。模型的验证损失稳定下降，最终在第 93 轮左右达到了最佳表现，验证集 RMSE（均方根误差）为 2.7060，验证集 MAE 为 7.8048（此处记录的 MAE 似乎有误，但 RMSE 是关键）。这个结果远优于我们之前所有的尝试，标志着我们的模型探索取得了成功。我们保存了这一轮的模型权重文件 final_bmi_predictor.pth，用于后续的评估和部署。

```
SimpleCNN(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU()
    (6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU()
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU()
    (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (regressor): Sequential(
    (0): Linear(in_features=12544, out_features=1024, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=1024, out_features=512, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=512, out_features=1, bias=True)
  )
)

正在应用 Kaiming He 初始化...
权重初始化完成。
```

四、成果展示

在模型训练成功后，我们进入了成果展示阶段，包括对模型在独立测试集上的性能评估，以及将其部署为可交互的 Web 应用。

4.1 模型性能评估

为了客观地评价模型的泛化能力，我们使用了课程提供的独立测试集。这个测试集包含了 1027 张已经裁剪好的人脸图片（男女各 513 张）和对应的 annotation.csv 标注文件。我们编写了 evaluate.py 脚本来执行评估。

```
警告：测试图片 test/data/m_505 未找到，将跳过。
警告：测试图片 test/data/m_506 未找到，将跳过。
警告：测试图片 test/data/m_507 未找到，将跳过。
警告：测试图片 test/data/m_508 未找到，将跳过。
警告：测试图片 test/data/m_509 未找到，将跳过。
警告：测试图片 test/data/m_510 未找到，将跳过。
警告：测试图片 test/data/m_511 未找到，将跳过。
```

在评估前，我们又一次遇到了数据层面的细节问题。我们发现标注文件中的图片名没有.jpg 后缀，并且部分图片的文件名后缀是大写的.JPG，导致程序无法找到文件。为此，我们又编写了一个小脚本 fix_data.py，自动化地为标注文件中的文件名添加后缀，并将所有.JPG 后缀的图片重命名为.jpg。这个小插曲再次提醒我们，在机器学习项目中，细致的数据处理至关重要。

```
警告：测试图片 test/data/f_487.jpg 未找到，将跳过。
警告：测试图片 test/data/f_342.jpg 未找到，将跳过。
警告：测试图片 test/data/f_126.jpg 未找到，将跳过。
警告：测试图片 test/data/m_336.jpg 未找到，将跳过。
警告：测试图片 test/data/nan.jpg 未找到，将跳过。
  6%|
| 1/17 [00:00<00:08,  1.87it/s]警告：测试图片 test/data/m_382.jpg
 未找到，将跳过。
警告：测试图片 test/data/m_246.jpg 未找到，将跳过。
100%|
```

```
root@autodl-container-a2f645b386-56e29bdd:~/autodl-tmp/code/test# python fix_data.py
  Renamed: f_342.JPG -> f_342.jpg
  Renamed: f_483.JPG -> f_483.jpg
  Renamed: m_246.JPG -> m_246.jpg
  Renamed: m_336.JPG -> m_336.jpg
  Renamed: m_382.JPG -> m_382.jpg
  文件后缀名修改完成！
```

完成数据修正后，我们加载了之前保存的最佳模型 final_bmi_predictor.pth，在完整的测试集上进行了预测。最终的评估结果如下：

```
root@autodl-container-a2f645b386-56e29bdd:~/autodl-tmp/code# python evaluate.py
正在加载测试集...
测试集加载完成，共 1027 个样本。

==================== 开始在测试集上评估 ====================
成功加载模型权重从: final_bmi_predictor.pth
正在测试集上进行预测...
  0%|                                                                | 0/17 [00:00<?, ?it/s]
警告：测试图片 test/data/nan.jpg 未找到，将跳过。
100%|████████████████████████████████████████████████████████████| 17/17 [00:00<00:00, 20.92it/s]

==================== 最终评估结果 ====================
  - 测试集损失 (SmoothL1Loss): 3.9091
  - 测试集均方根误差 (RMSE):    5.4916
  - 测试集平均绝对误差 (MAE):    4.3876
==================================================
```

测试集损失 (SmoothL1Loss): 3.9091

测试集均方根误差 (RMSE): 5.4916

测试集平均绝对误差 (MAE): 4.3876

对这个结果的解读如下：

MAE 为 4.3876，这具有最直观的意义。它表明我们的模型在面对一个从未

见过的面部照片时，其预测的 BMI 值与真实值之间的平均差距约为 4.39 个单位。考虑到我们数据集规模非常小（仅 512 张训练图片），且任务本身具有极高难度，这是一个可以接受且令人鼓舞的结果。

RMSE 为 5.4916，它比 MAE 要高，这是符合统计规律的，因为它对较大的误差给予了更高的权重。

泛化差距分析：模型的验证集 RMSE 约为 2.7，而测试集 RMSE 为 5.5。两者之间存在一定的差距，这被称为“泛化差距”。这表明模型在训练和验证集上学到的模式，并不能完美地迁移到测试集上。这个差距的存在是正常的，原因可能包括训练集和测试集在数据分布上的微小差异，以及模型因训练数据量不足而产生的轻微过拟合。但在可接受的范围内，说明模型没有产生严重的过拟合。

总而言之，评估结果证明了我们的模型具备了从人脸图像中预测 BMI 的基本能力，达到了项目的预期目标。

4.2 Web 应用部署与界面展示

为了让我们的模型能够被方便地使用和展示，我们将其部署成了一个 Web 应用。

后端实现：我们使用 Flask 框架编写了 app.py。它主要包含两个功能：

初始化：在应用启动时，加载我们训练好的 BMIPredictor 类，该类内部封装了模型加载、图像预处理和预测的所有逻辑。

API 端点：我们创建了一个/predict 路由，它接受通过 HTTP POST 方法上传的图片文件。接收到文件后，它直接调用 BMIPredictor 的预测方法，并将预测的 BMI 值或错误信息以 JSON 格式返回给前端。

前端实现：我们编写了 index.html 文件，它集成了 HTML, CSS 和 JavaScript，构建了一个简洁、美观且交互友好的用户界面。其核心功能包括：

美观的布局：页面居中，元素清晰，提供了明确的操作指引。

图片上传与预览：用户点击“选择图片”后，所选的图片会立刻在页面上回显，提供了即时的视觉反馈。

异步请求与动态反馈：点击“预测 BMI”按钮后，JavaScript 会使用 fetch API 将图片异步发送到后端。在等待后端响应时，页面上会显示一个加载动画，提升了用户体验。

结果展示：当后端返回结果后，预测出的 BMI 值会以一个漂亮的标签形式，叠加显示在预览图片的下方。如果出现错误（如未检测到人脸），也会将错误信息友好地展示出来。

我们将应用在本地成功运行，并通过浏览器访问 http://127.0.0.1:5000，对应用进行了完整的测试。以下是应用的运行截图：

服务器启动界面：

```
PS E:\大三下\AAA人工智能与机器学习\AAA大作业\code> python -u "e:\大三下\AAA人工智能与机器学习\AAA大作业\code\app.py"
BMIPredictor initialized successfully!
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.20.10.2:5000
Press CTRL+C to quit
 * Restarting with stat
BMIPredictor initialized successfully!
 * Debugger is active!
 * Debugger PIN: 701-262-478
127.0.0.1 - - [27/Jun/2025 14:34:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jun/2025 14:34:12] "GET /favicon.ico HTTP/1.1" 404 -
```

Web 应用预测界面：

我们上传了一张不在任何数据集中的个人照片进行测试，模型给出了一个预测值。虽然该预测值与真实值存在一定误差（约+5.4），但我们分析这很可能是因为我们从 Reddit 爬取的训练数据主要来自于“减肥”话题，样本普遍偏胖，导致模型对 BMI 的估计整体偏高。这进一步验证了数据集的偏差会对模型行为产生直接影响。

五、总结和讨论

5.1 项目总结

本项目成功地设计并实现了一个从数据获取到 Web 应用部署的完整人脸 BMI 预测系统。我们从零开始，通过网络爬虫构建了初始数据集，经过严格的数据清洗和预处理，获得了可用于训练的干净数据。在模型探索阶段，我们经历了从失败的迁移学习尝试到自定义 CNN 的艰难调参，最终通过对数据分布的深入分析，找到了更换损失函数这一关键突破口，并结合多种优化技术，成功训练出了一个性能可接受的回归模型。最终，模型在独立测试集上取得了 4.39 的平均绝对误差。我们将此模型封装在 Flask 后端中，并配合一个交互式的前端页面，最终交付了一个功能完善、用户体验良好的 Web 应用。整个项目不仅锻炼了我们解决复杂机器学习问题的综合能力，也让我们对深度学习项目的完整生命周期有了深刻的理解。

5.2 经验与教训

在整个项目过程中，我们积累了许多宝贵的经验和教训：

数据永远是第一位的：项目的成败在很大程度上取决于数据的质量。无论是前期的爬取、中期的清洗（尤其是 MTCNN 的人脸检测），还是后期的测试集文件格式修正，我们在数据上花费了大量精力，而这些工作都被证明是值得的。数据集的偏差（如样本偏胖）会直接反映在最终模型的行为上。

迭代和试错是常态：我们深刻体会到，机器学习项目，尤其是研究探索性质的项目，其路径绝不是线性的。我们大胆地放弃了最初看起来很“高级”的迁移学习方案，勇敢地承认调参的失败并回过头来重新审视数据和损失函数，这种不断迭代、敢于推翻重来的精神是解决问题的关键。

理论与实践相结合：当模型训练停滞不前时，盲目地调整超参数往往是徒劳的。我们回到理论，分析了 MSE 损失函数对异常值的敏感性，并据此选择了更合适的 Huber Loss，才最终打破了僵局。这让我们明白，深刻理解算法原理对于指导实践至关重要。

善用工具，解放生产力：无论是利用 AutoDL 云平台加速训练，还是使用 Flask、Pillow、Pandas 等成熟的 Python 库，善用现有的高效工具可以让我们将精力集中在最核心的算法和逻辑上，极大地提高了开发效率。

5.3 不足与展望

尽管项目取得了预期的成果，但我们清醒地认识到其中仍存在许多不足之处，也为未来的工作留下了广阔的提升空间：

数据集的局限性：我们当前的数据集规模小（仅 512 个样本），且来源单一（Reddit 某特定版块），存在明显的样本选择偏差（用户多为关注体重的白人群体）。这极大地限制了模型的泛化能力和预测的普适性。未来，可以尝试爬取更多元化的数据源，或通过众包等方式构建一个规模更大、覆盖人群更广的数据集。

模型精度的提升：MAE 约为 4.4 的精度对于一个初步探索性项目是可以接受的，但距离实际应用还有较大差距。未来可以尝试更先进、更复杂的模型结构，如引入注意力机制的 CNN，或者探索 Vision Transformer（ViT）等新范式在该任务上的表现。

模型可解释性：我们的模型目前是一个“黑箱”。未来可以引入一些可解释性 AI（XAI）技术，如 Grad-CAM，来可视化模型在做预测时究竟关注了人脸的哪些区域，这不仅能帮助我们理解模型的决策依据，也可能为进一步优化提供线索。

六、参考文献

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern

recognition (pp. 770-778).
[3] Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters, 23(10), 1499-1503.
[4] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).
[5] Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In International Conference on Learning Representations.
[6] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32 (pp. 8026-8037).
[7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems 25.
[8] Girshick, R. (2015). Fast R-CNN. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).
[9] Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. In International Conference on Learning Representations.
[10] Lu, Y., He, Z., Zhu, Z., Li, S., Bairen, A., & Liu, Y. (2022). Deep-learned BMI: a new, deep-learning-based body mass index for cardiovascular disease risk stratification. The Lancet Digital Health, 4(5), e358-e367.
[11] Rothe, R., Timofte, R., & Van Gool, L. (2018). Deep expectation of real and apparent age from a single image. International Journal of Computer Vision, 126(2), 144-157.