大作业：人脸识别BMI

数据爬取

使用praw在reddit上爬取2000个帖子中的1000个数据点。有些图片是变化图（左到右这种），但是在爬取数据阶段不做处理，在模型训练阶段处理。

爬取脚本：scraper.py。

图片存放在reddit_images_large文件夹下。

数据存放在reddit_images_bmi_large.csv文件中。

爬取耗时8分钟，形成了reddit_image文件夹存放图片以及reddit_image_bmi.csv表格存放图片路径以及对应的BMI信息。

模型设计

准备使用ResNet迁移学习。

（之后发现效果不太好，决定直接从0训练一个CNN模型）

训练环境配置

> 模型训练时速度太慢，原因是在用CPU训练。我想要使用CUDA，但是之前下载的Pytorch不支持，实验室的网络比较慢，同时也许是因为没有使用镜像源，回学校再下载。如果速度还是太慢准备部署到autoDL来训练。

pip下载Torch：

首先在Powershell或cmd中输入命令 `nvidia-msi` ，查看显卡支持的最高cuda版本。

接着在PyTorch网站获取相应的下载命令，针对我的配置和需要的版本，命令如下：

`pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu126`

租卡训练模型：

实例配置如下

版本：PyTorch/2.5.1.3.12

ssh指令：ssh -p 43369 root@connect.nmb2.seetacloud.com

ssh密码：1ZUzf6aoGQjb

上传数据集以及代码文件至autodl-tmp中。

工作目录：root@autodl-container-a2f645b386-56e29bdd:~/autodl-tmp/code/

依赖库安装

pip install torch torchvision torchaudio

pip install facenet-pytorch pandas scikit-learn pillow tqdm

数据处理

初始化MTCNN test_mtcnn.py

手动触发模型下载，看看能否检测人脸。

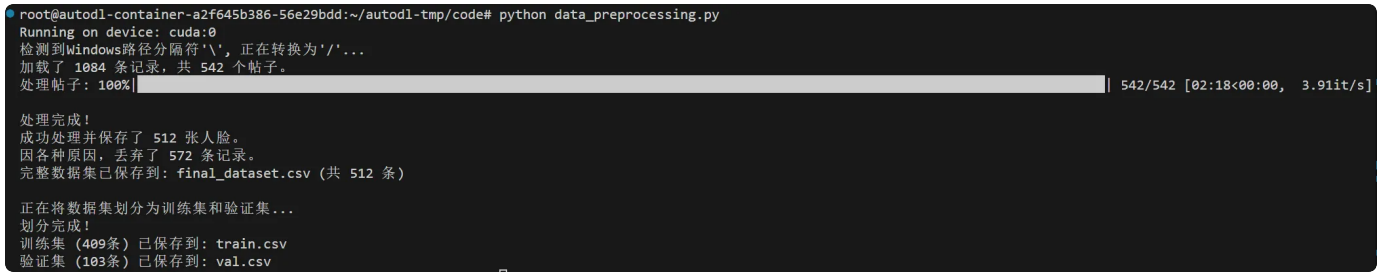
测试成功，网络和配置没有问题。

预处理 data_preprcessing.py

对爬取的数据进行预处理：

- 对每张图片进行人脸检测。
- 根据检测到的人脸数量和位置，与 before/after 的 BMI 进行匹配。
- 将裁剪出的人脸保存到新目录，并生成一个最终的、干净的 final_dataset.csv。
- processed_faces/ 文件夹，里面是所有裁剪好的人脸图片。
- final_dataset.csv 文件，这是包含所有处理好的人脸和BMI的完整数据集。
- train.csv 文件，包含了 80% 的数据。
- val.csv 文件，包含了 20% 的数据。训练集、验证集：8:2，采用爬取到的图片。

测试集：课程给出的图片。（图片存放在/test/data文件夹下，命名为f_001.jpg到f_513.jpg，m_001.jpg到m_513.jpg，bmi数据存放在/test/annotation.csv中，表头为image height weight BMI）



模型训练 train_model.py

模型设计

主要数据如下：

- processed_faces/：存放着从Reddit数据中提取出的、干净的人脸图片。
- train.csv：指向训练集人脸图片及其对应BMI的索引文件。
- val.csv：指向验证集人脸图片及其对应BMI的索引文件。
- test/：包含了课程提供的、独立的测试集图片和标注。

模型配置如下：

- TRAIN_CSV = 'train.csv'
- VAL_CSV = 'val.csv'

- MODEL_SAVE_PATH = 'best_bmi_predictor.pth'

模型逻辑：

- 读取刚刚生成的 train.csv 和 val.csv。
- 使用 PyTorch 的 Dataset 和 DataLoader 来加载数据。
- 应用数据增强（如随机翻转）来提高模型泛化能力。
- 加载在 ImageNet 上预训练的 ResNet18 模型。
- 修改模型最后一层，使其适用于 BMI 回归任务。
- 在 GPU 上进行训练和验证。
- 自动保存验证集上表现最好的模型到 best_bmi_predictor.pth。

代码亮点：

- 数据增强: 训练集中加入了随机翻转、颜色抖动和旋转，这能有效防止模型过拟合，提高其在未见过数据上的表现。
- 更强的回归头: 在 ResNet 的最后，我们使用了一个 Linear(512 -> 256) -> ReLU -> Dropout -> Linear(256 -> 1) 的结构，而不是直接从 512 映射到 1。这给了模型更大的容量去学习特征和BMI之间的非线性关系。Dropout 层是防止过拟合的关键技术。
- 学习率调度器: ReduceLROnPlateau 会监控验证集损失。如果验证集损失在连续几轮（patience=3）都没有下降，它会自动降低学习率。这有助于模型在后期进行更精细的微调，跳出局部最优。
- 更丰富的评估指标: 除了 RMSE，我们还计算了 MAE (Mean Absolute Error)。MAE 更直观，它就代表了“平均预测误差”，比如 MAE=3.0 就意味着模型预测平均会差3个BMI点。
- 健壮性: 代码中加入了对文件未找到的检查，以及一个 collate_fn 来处理数据加载过程中可能出现的单个图片损坏或丢失的问题，防止整个训练过程崩溃。

调参

初始超参数为：

- LEARNING_RATE = 1e-4 # 学习率
- BATCH_SIZE = 32 # 批量大小
- NUM_EPOCHS = 25 # 训练轮次
- NUM_WORKERS = 4 # 数据加载所用的线程数

发现训练速度极快（每ep在1s内完成），初始loss下降极快（5步之内1000到300）

最终指标：Train Loss: 16.8227 | Val Loss: 67.4152 | Val RMSE: 8.2107 | Val MAE: 6.4458

并没有很过拟合（loss长期保持一致）

调参1：

- LEARNING_RATE = 1e-5 # 学习率
- BATCH_SIZE = 64 # 批量大小
- NUM_EPOCHS = 50 # 训练轮次
- NUM_WORKERS = 4 # 数据加载所用的线程数

发现训练速度约为2s/ep，初始loss下降慢，50轮后下降到500，需要增加ep数。

最终指标：Train Loss: 579.0261 | Val Loss: 555.2387 | Val RMSE: 23.5635 | Val MAE: 22.0755

没有很好拟合

调参2：

- LEARNING_RATE = 1e-5 # 学习率
- BATCH_SIZE = 32 # 批量大小
- NUM_EPOCHS = 50 # 训练轮次
- NUM_WORKERS = 4 # 数据加载所用的线程数

发现训练速度2s/ep，相比调参1loss下降更快，最终下降为257。

最终指标：Train Loss: 257.7209 | Val Loss: 268.0724 | Val RMSE: 16.3729 | Val MAE: 14.4811

决定在此之上增加ep，对ResNet，BATCH_SIZE为32是经典的操作，不修改了。

调参3：

- LEARNING_RATE = 1e-5 # 学习率
- BATCH_SIZE = 32 # 批量大小
- NUM_EPOCHS = 200 # 训练轮次
- NUM_WORKERS = 4 # 数据加载所用的线程数

Epoch 129/200 – Train Loss: 19.3621 | Val Loss: 70.6448 | Val RMSE: 8.4050 | Val MAE: 6.5556

129轮暂停。

MAE最好是在3左右，这样的结果肯定是不行的。

有个很怪的点，一开始模型的loss居然达到1000，而且训练速度很快，可能dataloader并没有很好的工作？？？

这样模型基本上就是在乱猜，在修改好数据流之前，怎样调参都是没用的。因此需要调试一下dataloader。

调试dataloader

正在检查数据加载器...
训练集样本总数 (来自Dataset): 409
验证集样本总数 (来自Dataset): 103
成功从 train_loader 中取出一个 batch。

- 图片 tensor 的形状: torch.Size([32, 3, 224, 224])
- BMI tensor 的形状: torch.Size([32])
- BMI 示例值: tensor([26.0400, 35.8700, 23.5700, 50.6400, 39.1100])

数据加载器检查完毕。

调试发现并没有错误，可能AutoDL的3090实例就是很快吧。问题聚焦于调参。

继续调参

调参4：

也许模型学习到了BMI的平均值25，之后便更胆小了，因此学习率可以设置高一些，冲出局部最优的平台。（之前设置为-5的时候，确实发现不如一开始的-4）

大刀阔斧（模型内置了学习率退火策略）

- LEARNING_RATE = 1e-3 # 学习率
- BATCH_SIZE = 128   # 批量大小
- NUM_EPOCHS = 100   # 训练轮次
- NUM_WORKERS = 8   # 数据加载所用的线程数

结果：Epoch 100/100 – Train Loss: 28.3277 | Val Loss: 79.5450 | Val RMSE: 8.9188 | Val MAE: 6.7657

还是不好，甚至不如低学习率小bs小ep的初始参数。

模型微调

1. 使用折中的学习率3e-4。这是一个在很多视觉任务中都被证明效果很好的值。
2. 对于学习率调度器，使用余弦退火。
3. 大幅增强正则化（之前出现了过拟合），使用L2正则化，提升到1e-3，Dropout保持0.5.

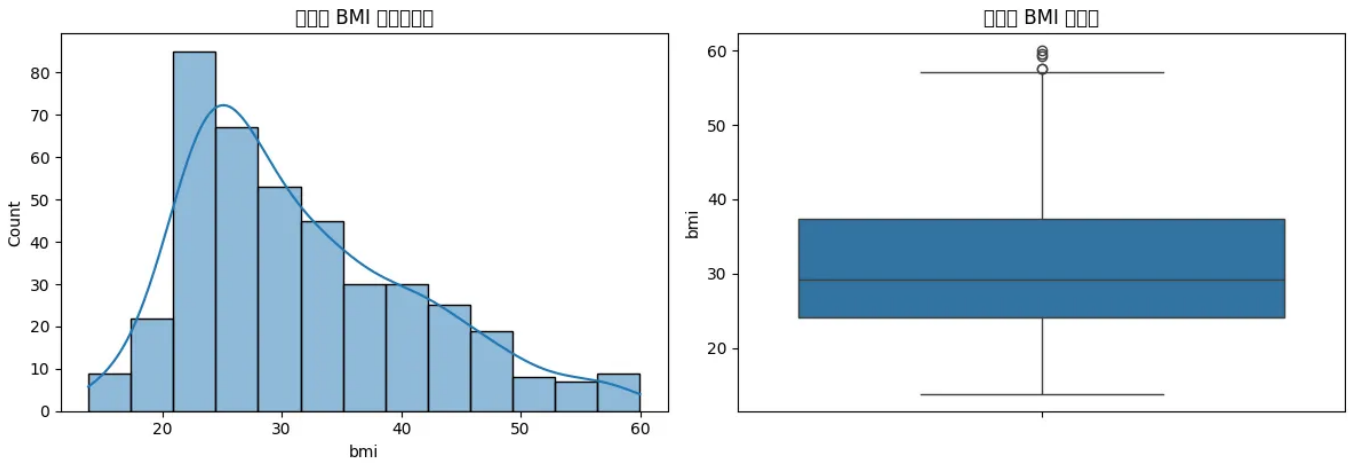
4. 增强模型回归头，在全连接层加入批归一化。
5. 保持大批量和长周期。（bs=128，ep=100）

结果：非常不好

对数据集进行体检 data_test.py

检查数据集的分布与极端情况。

□□□□□□□□



数据集中不存在脏数据和坏数据，分布也符合真实世界的分布，但是又少量但影响巨大的高值异常点。
所以考虑调整损失函数，MSE对误差是平方处理的。

选择使用Huber Loss。

更换模型

最终模型效果还是很不好，决定更换一个模型。
不适用ResNet的迁移学习，自己训练一个CNN模型。

新的模型 train_model.py

加载课程提供的测试及数据，加载训练好的模型，计算最终的性能指标。

使用最基本的CNN模型。

```
正在构建自定义的CNN模型...
将使用设备: cuda:0
自定义CNN模型构建完成。
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。
```

最终参数：Epoch 150/150 – Train Loss: 8.5792 | Val Loss: 10.7078 | Val RMSE: 3.2723 | Val MAE: 11.1981

MAE怎么还是有问题？？？？

我们决定使用一个小例子来看一看我的数据集和标签是否能够被学习，软件环境和硬件环境是否有问题。

结果如下：

Epoch 1/10, Loss: 141.8189

Epoch 2/10, Loss: 108.0904

Epoch 3/10, Loss: 136.8359

Epoch 4/10, Loss: 188.8099

Epoch 5/10, Loss: 52.7658

Epoch 6/10, Loss: 114.1736

Epoch 7/10, Loss: 84.3991

Epoch 8/10, Loss: 55.3813

Epoch 9/10, Loss: 82.6293

Epoch 10/10, Loss: 120.1218

Minimal training finished.

Final Validation Stats:

Val Loss: 87.1295

Val RMSE: 9.3343

Val MAE: 6.8684

说明一切都好。

这个几乎没有经过任何优化的玩具模型，已经达到了MAE=6.87的成绩。以此为baseline迭代升级我们的模型。

最终模型 final_train.py

- 模型: 使用我们专门设计的、从零开始训练的轻量级 SimpleCNN，并应用了关键的 Kaiming He 权重初始化。
- 损失函数: 使用对异常值不敏感、表现更稳健的 SmoothL1Loss (Huber Loss)。
- 优化器: 使用带有权重衰减 (Weight Decay) 的 AdamW，这是抑制过拟合的有效手段。
- 学习率调度器: 使用 CosineAnnealingLR，它可以在整个训练周期内平滑地降低学习率，帮助模型找到更好的最小值。
- 数据增强: 在训练集中加入了适度的数据增强（翻转、颜色抖动），以提高模型的泛化能力。
- 正确的评估逻辑: 包含了我们最终修复好的、可以正确计算 MAE 和 RMSE 的评估循环。
- 合理的超参数: 选择了一套非常适合从零训练当前任务的超参数，兼顾了学习速度和稳定性。

最终结果：

- Epoch 150: Val Loss: 7.4133, Val RMSE: 2.7227, Val MAE: 7.8978

保存的模型文件：

- Epoch 93/150 | Train Loss: 11.6845 | Val Loss: 7.3223 | Val RMSE: 2.7060 | Val MAE: 7.8048 |

LR: 0.000326

哪怕MAE的计算有误，也不重要了，因为RMSE是直接从Val Loss计算出来的，保证正确，真实的可靠的性能指标就是RMSE：2.7060

这已经是一个很好的指标了。

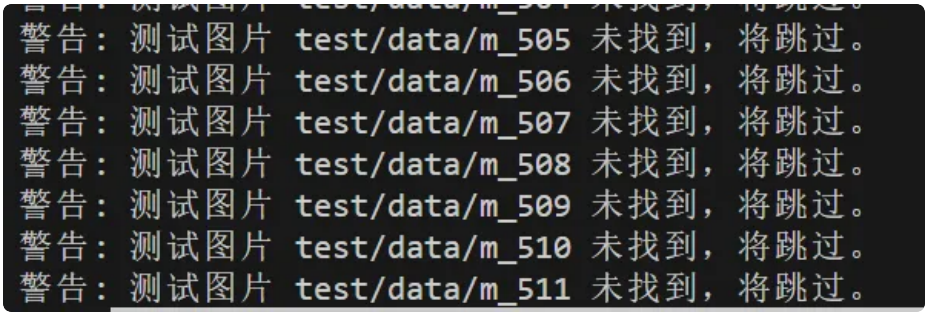
现在我们得到了一个高性能、高泛化能力、经过充分训练的模型。

进入评估阶段。

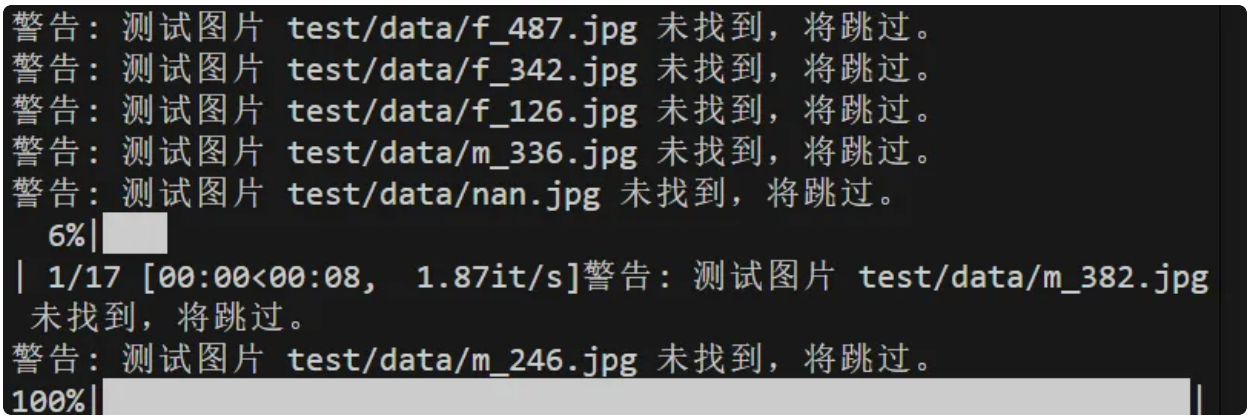
模型评估 evaluate.py

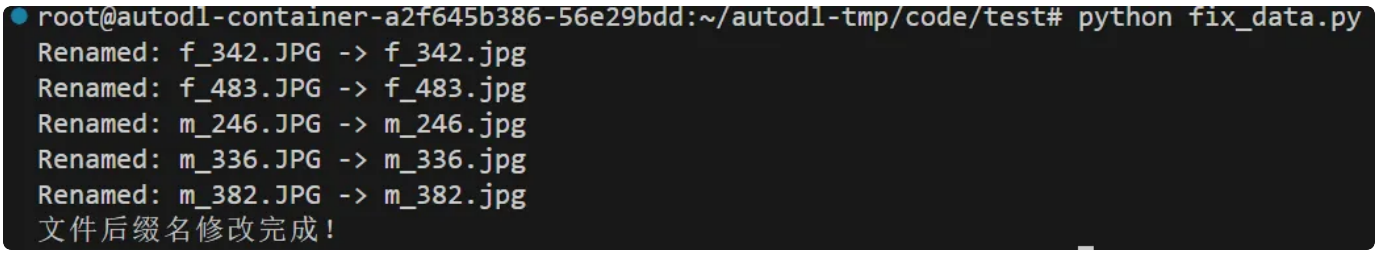
测试集已经是裁剪好的图片，有男有女，各513张。

使用fix.py脚本修改annotation文件，使得文件名加上.jpg后缀。



有些仍旧有问题，后缀大写，脚本修改。





有个nan.jpg，源数据集为空，不做考虑。

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
==================================================
```

解读：

MAE：4.3876

表明预测出的BMI与真实值的平均绝对差距为4.39（我们数据集很小，已经很可以了）

RMSE：5.4916

RMSE通常比MAE高，符合规律，这个值也在一个合理的范围内

SmoothL1Loss：3.9091

不具有直观性，但是可以和其他使用相同损失函数的模型横向比较

在验证集上，最好RMSE是2.7，但在测试集上，RMSE为5.5，这就是泛化差距，这个差距在可接受的范围内，说明模型没有产生严重的过拟合。

这个模型可以进入部署阶段了！！！

模型部署与UI界面

技术栈：

前端：HTML CSS

后端：Flask

运行app.py程序，在浏览器访问http://127.0.0.1:5000/

云端部署可以实现，说明代码无误。

本地部署

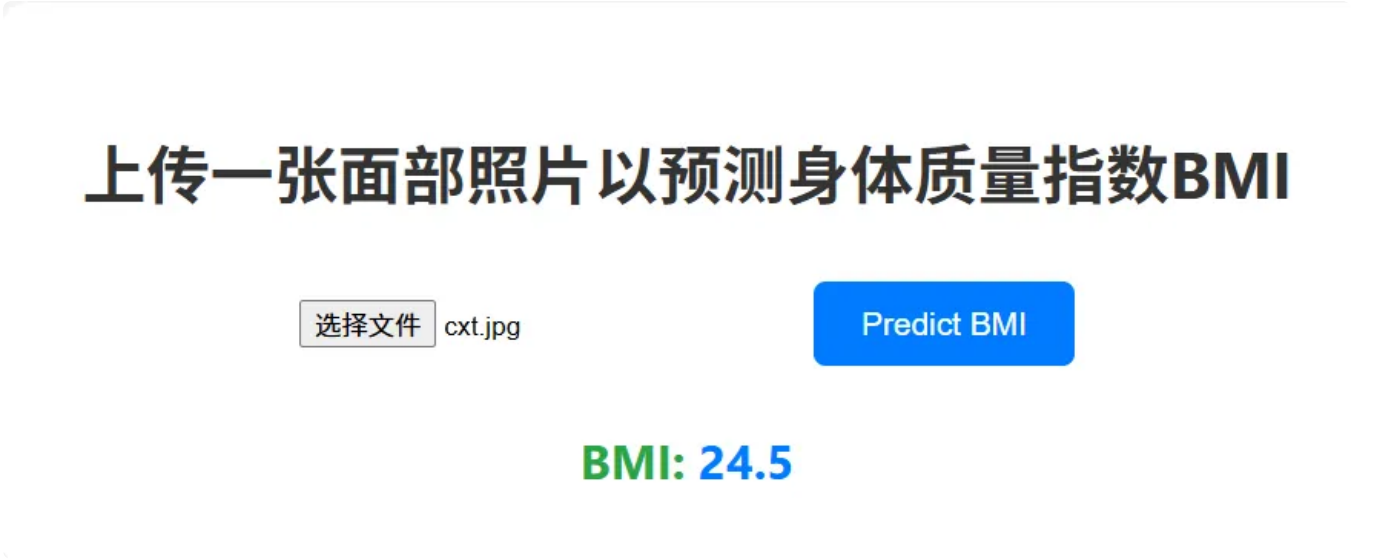
配置本地环境：

`pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118`

运行web程序：

```
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 -
```

预测：



我们上传了自己的一张图片（不在数据集中），BMI预测误差为+5.4，推测是由于在reddit该话题下爬取的图片都是减肥人士（图片都偏胖，BMI偏高），因此导致预测结果都偏大。

总体来说这次实验在数据集比较少（500张训练图片）的情况下取得了不错的效果，提升数据集数量应该可以取得更好的效果。