1000hz-BP.py 代码解读

✅ 功能说明 (Functionality Explanation):

该脚本的核心功能是**利用光电容积脉搏波 (PPG) 数据来预测个体的收缩压 (SBP) 和舒张压 (DBP)**。其主要执行流程可以概括为以下几个步骤：

1. **定义数据归一化方法**：确保预测时使用的数据预处理方式与模型训练时所用的方法（最小-最大归一化）保持一致。
2. **构建数据加载和网络模型结构**：定义了适用于 PyTorch 框架的自定义数据集类 (`PredictionDataset`) 和一个多层感知机（MLP）神经网络 (`NumericNetwork`)。
3. **加载归一化参数**：从训练数据中提取的特征和目标的最小-最大值，用于后续测试数据的归一化和预测结果的反归一化。
4. **处理待预测数据**：读取测试集 Excel 文件，提取相关特征，并使用加载的参数对这些特征进行归一化处理。
5. **集成模型预测**：脚本设计为加载通过10折交叉验证训练出的多个模型。对每一个模型，都执行一次完整的预测流程，然后将这10个模型的预测结果进行平均，以获得更稳健的最终预测值。
6. **结果处理与保存**：将平均后的 SBP 和 DBP 预测值进行反归一化（转换回原始的血压单位），并将最终结果保存到一个新的 Excel 文件中。

🔍 详细结构解读 (Detailed Structure Interpretation):

1. 数据归一化函数 `normalize_dataframe(df, min_max_values)`

此函数接收一个 Pandas DataFrame 和一个包含各列最小-最大值的字典作为输入。它遍历 DataFrame 中的指定列，并对每一列应用最小-最大归一化公式：$X_{norm} = \frac{(X - X_{min})}{(X_{max} - X_{min})}$。

2. 自定义数据集类 `PredictionDataset(torch.utils.data.Dataset)`

这是一个标准的 PyTorch `Dataset` 子类，用于封装预测数据集。

- `__init__(self, dataframe)`：构造函数接收一个已归一化的特征 DataFrame，并将其转换为 NumPy 数组。
- `__len__(self)`：返回数据集中样本的数量。
- `__getitem__(self, idx)`：根据索引 `idx` 返回对应样本的特征张量。

3. 模型定义 `NumericNetwork(torch.nn.Module)`

这是一个用于回归任务的多层感知机（MLP）网络结构。

- **输入层**：接收24个输入特征 (由 `input_size=len(feature_columns)` 动态确定)。
- **隐藏层**：包含4个全连接 (FC) 层。每个 FC 层之后通常会接一个批归一化 (`BatchNorm1d`) 层以稳定训练，一个 Dropout 层以防止过拟合，以及一个 LeakyReLU 激活函数引入非线性。
  - `fc1` : Linear(24, 128) -> BatchNorm1d(128) -> LeakyReLU -> Dropout(0.3)
  - `fc2` : Linear(128, 256) -> BatchNorm1d(256) -> LeakyReLU -> Dropout(0.3)
  - `fc3` : Linear(256, 512) -> BatchNorm1d(512) -> LeakyReLU -> Dropout(0.3)
  - `fc4` : Linear(512, 256) -> BatchNorm1d(256) -> LeakyReLU -> Dropout(0.3)
- **输出层**：从最后一个共享的隐藏层（ `fc4` 的输出）引出两个独立的全连接层，分别用于预测 SBP 和 DBP。
  - `fc_sbp` : Linear(256, 1) (预测 SBP)
  - `fc_dbp` : Linear(256, 1) (预测 DBP)
- **前向传播** `forward(self, x)`：定义了数据通过网络的流程，最终返回 SBP 和 DBP 的预测值。

4. 主执行流程 (Main Execution Pipeline):

脚本的 `if __name__ == "__main__":` 部分按以下顺序执行：

- **环境设置**：自动检测并设置可用的计算设备 (CUDA GPU 或 CPU)。
- **参数定义**：硬编码了特征列名 (`feature_columns`)、目标列名 (`target_columns`)、训练/测试文件路径以及模型文件路径模式。
- **归一化参数计算**：
  - 读取训练数据 (`train_file_path`)。
  - 计算训练数据中**所有特征列和目标列**的最小值和最大值，并存储在 `min_max_values` 字典中。这些值将用于后续测试数据的特征归一化和预测结果的反归一化。
- **测试数据预处理**：
  - 读取测试数据 (`test_file_path`)。
  - 仅提取 `feature_columns` 定义的特征。
  - 使用 `normalize_dataframe` 函数和从训练数据得到的 `min_max_values` 对测试特征进行归一化。
- **数据加载器构建**：
  - 使用归一化后的测试特征创建 `PredictionDataset` 实例。
  - 将数据集包装在 `DataLoader` 中，以便进行批量预测，设置 `batch_size=64` 且不打乱数据 (`shuffle=False`)。
- **模型加载与集成预测**：
  - 使用 `glob.glob` 找到所有符合 `model_paths` 模式（例如 `'./models_10fold/*.pth'` ）的预训练模型文件。
  - 遍历每个找到的模型文件：
    - 实例化 `NumericNetwork` 。
    - 加载模型权重 (`load_state_dict`)。
    - 将模型移至选定的设备 (`.to(device)`)。
    - 设置模型为评估模式 (`model.eval()`)。
    - 在 `torch.no_grad()` 上下文中进行预测，以禁用梯度计算。
    - 遍历 `pred_loader` 中的数据批次，获取模型对 SBP 和 DBP 的原始（归一化）预测。
    - 收集所有批次的预测结果，并存储每个模型的完整预测序列。
  - 将所有模型（10折）的 SBP 和 DBP 预测结果分别进行平均，得到最终的集成预测值。
- **结果反归一化**：
  - 从 `min_max_values` 字典中获取 SBP 和 DBP 的原始最小-最大值。
  - 使用反归一化公式 $X = X_{norm} \times (X_{max} - X_{min}) + X_{min}$ 将平均后的预测值转换回其原始尺度。
- **结果保存**：
  - 创建一个包含反归一化后 SBP 和 DBP 预测值的 Pandas DataFrame。
  - 将该 DataFrame 保存到名为 `prediction_results.xlsx` 的 Excel 文件中，不包含索引。

🛠️ 改进建议 (Improvement Suggestions):

模型 (Model) 部分:

1. **模块化预测头 (Modular Prediction Heads)**：虽然共享中间层然后分离预测头是常见且合理的设计，但可以将 SBP 和 DBP 的最终预测层（例如 `fc_sbp` 和 `fc_dbp` ）封装到独立的 `nn.Module` 子类中，或至少在 `NumericNetwork` 内部更清晰地组织，以提高代码的可读性和模块性。
2. **使用 `torch.nn.Sequential`**：对于网络中连续的层序列（如 FC -> BatchNorm -> Activation -> Dropout），可以使用 `torch.nn.Sequential` 来容器化这些层，使 `NumericNetwork` 的 `__init__` 和 `forward` 方法更简洁。

```
# 例如，一个隐藏层块可以这样定义：
# self.hidden_block1 = nn.Sequential(
#     nn.Linear(input_size, hidden_size),
#     nn.BatchNorm1d(hidden_size),
*#     nn.LeakyReLU(),
```

```
*#     nn.Dropout(dropout_rate)
# )
```

3. **探索残差连接 (Residual Connections)**：对于包含多个堆叠层的网络，可以尝试加入残差连接（类似 ResNet 中的设计）。这有助于缓解梯度消失问题，并可能提升深层网络的训练效果和稳定性。

数据处理 (Data Processing) 部分：

1. **归一化参数的精确性**：
   - 在计算 `min_max_values` 时，虽然脚本最终正确地仅对特征列进行归一化，并将目标列的min-max值用于反归一化，但更清晰的做法可能是分别管理特征的归一化参数和目标的归一化参数。
   - 目前，`min_max_values` 同时存储了特征和目标的min/max值。当调用 `normalize_dataframe(test_df_features.copy(), min_max_values)` 时，由于 `test_df_features` 只包含特征列，因此函数只会使用 `min_max_values` 中对应特征列的条目，这是正确的。
   - 然而，为了更佳的可读性和维护性，可以考虑创建两个独立的字典：一个用于特征（从训练集的特征列计算），一个用于目标（从训练集的目标列计算）。

```
# 建议的方式：
# feature_min_max_values = {col: (train_df[col].min(), train_df[col].max()) for col in
feature_columns}
# target_min_max_values = {col: (train_df[col].min(), train_df[col].max()) for col in
target_columns}
# # 然后在归一化特征时使用 feature_min_max_values
# test_df_normalized = normalize_dataframe(test_df_features.copy(), feature_min_max_values)
# # 在反归一化目标时使用 target_min_max_values
# sbp_min, sbp_max = target_min_max_values['SBP']
# dbp_min, dbp_max = target_min_max_values['DBP']
```

   这避免了单个 `min_max_values` 字典潜在的混淆。
2. **Excel 读取引擎**：在调用 `pd.read_excel()` 时，显式指定 `engine="openpyxl"` (如果使用的是 `.xlsx` 文件) 或其他适用引擎，可以避免因环境差异导致的兼容性问题或警告。

程序结构优化 (Program Structure Optimization)：

1. **函数/类封装 (Encapsulation)**：将代码的主要逻辑块封装成独立的函数或类，例如：
   - `load_and_preprocess_data(file_path, feature_columns, min_max_scaler_params, is_train=False)`
   - `load_prediction_model(model_path, input_size, device)`
   - `perform_prediction(model, data_loader, device)`
   - `ensemble_predictions(all_preds_list)`
   - `denormalize_predictions(preds, target_min_max_values)`
   - `save_results_to_excel(results_df, output_path)` 这样做可以显著提高代码的可读性、可维护性和可测试性。
2. **日志记录 (Logging)**：使用 Python 的 `logging` 模块替代 `print()` 语句。日志记录提供了更灵活的输出控制（例如，不同级别的日志信息、输出到文件、时间戳等），这对于调试和生产环境中的监控非常有用。
3. **配置管理 (Configuration Management)**：将文件路径、模型参数（如 `batch_size`）、列名等硬编码的常量移到一个配置文件（如 JSON, YAML）或通过命令行参数 (`argparse` 模块)传入。这使得修改配置更加方便，无需直接编辑代码。
4. **类型提示 (Type Hinting)**：在函数签名和关键变量声明中加入类型提示，可以提高代码的可读性，并帮助静态分析工具（如 MyPy）捕获潜在的类型错误。

预测逻辑改进 (Prediction Logic Improvements)：

1. **GPU 支持已实现**：脚本已包含 `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")` 以及将模型和数据迁移到相应设备的逻辑 (`model.to(device)`, `batch.to(device)`)，这是很好的实践。

2. **异常处理 (Exception Handling)**：在文件读取、模型加载等可能发生 I/O 错误或运行时错误的地方，添加 `try-except` 块来捕获和处理异常，使程序更加健壮。例如，如果模型文件不存在或损坏，程序应能优雅地处理而不是直接崩溃。
3. **模型路径 `glob` 改进**：
   - 当前的 `model_paths = './models_10fold/*.pth'` 与 `glob.glob(model_paths)` 配合使用是正确的，用于查找目录下的所有 `.pth` 文件。
   - 确保 `models_10fold` 目录结构与此模式匹配。

其他 (Miscellaneous)：

1. **代码注释和文档字符串 (Comments and Docstrings)**：为函数、类和复杂的代码块添加清晰的注释和文档字符串，解释其功能、参数和返回值。
2. **设置随机种子 (Random Seeds)**：虽然在预测脚本中随机性因素较少（主要在于 `DataLoader` 的 `shuffle`，此处为 `False`），但在涉及任何随机过程（如某些数据增强，虽然此处未用）或为了完全复现性时，在脚本开头设置 NumPy 和 PyTorch 的随机种子是一个好习惯。

这些建议旨在提高代码的鲁棒性、可维护性、可读性和灵活性。根据具体需求和项目上下文，可以选择性地采纳。