基于 SVM 与 CNN 的人脸识别系统设计与实现

实验者：(请填写您的姓名)

2025 年 6 月 19 日

目录

摘要

本报告详细介绍了一个模块化人脸识别系统的设计与实现。该系统集成了两种主流的机器学习与深度学习方法：支持向量机（SVM）和卷积神经网络（CNN）。系统提供了一个功能完善的图形用户界面（GUI），允许用户加载标准数据集（如 Olivetti Faces, LFW）或自定义数据集，对选定模型进行训练、评估、保存和加载，并对外部图像进行实时人脸识别。在 SVM 模型中，我们实现了多种特征提取方法（如 HOG、LBP），并结合主成分分析（PCA）进行降维和网格搜索进行超参数优化。在 CNN 模型中，我们设计了一个包含残差连接和空间注意力机制的现代网络架构。实验结果表明，CNN 模型在识别准确率上显著优于传统的 SVM 模型，但 SVM 在训练速度和资源消耗上具有优势。本报告完整地阐述了系统的架构设计、关键技术实现、实验流程和结果分析，为构建和评估人脸识别系统提供了一个全面的实践案例。

1 引言

人脸识别作为生物特征识别领域的核心技术之一，在身份验证、安防监控、人机交互等方面有着广泛的应用前景。其主要任务是从图像或视频中检测、识别人脸，并确定其身份。传统的人脸识别方法通常依赖于手工设计的特征提取器（如 HOG、LBP）和经典的机器学习分类器（如 SVM）。近年来，随着深度学习的飞速发展，基于卷积神经网络（CNN）的方法因其强大的自动特征学习能力，在人脸识别任务上取得了突破性进展。

为了系统性地研究和比较这两种技术路线，本项目设计并实现了一个集成化的人脸识别系统。该系统的主要目标包括：

- 构建一个模块化的软件架构，将数据处理、模型训练、评估和界面展示分离。
- 实现一个基于传统机器学习的识别流程，以支持向量机（SVM）为核心，并集成多种特征工程技术。
- 实现一个基于深度学习的识别流程，设计并训练一个现代化的卷积神经网络（CNN）。
- 开发一个直观的图形用户界面（GUI），方便用户进行数据集管理、模型训练、性能评估和实时识别等操作。
- 对两种模型在相同数据集上的性能进行量化比较和分析，探讨各自的优缺点。

本报告将详细介绍系统的各个模块，展示关键代码实现，并通过实验验证系统的有效性和模型的性能。

2 系统设计与架构

本系统采用模块化设计思想，将整个系统划分为四个核心模块：GUI 模块、数据处理模块、模型模块和评估模块。这种设计提高了代码的可维护性和可扩展性。

2.1 总体架构

系统总体架构如图**??**所示。用户通过 GUI 与系统交互，GUI 负责调度其他模块完成相应任务，并将结果反馈给用户。

- **GUI 模块 (`main_window.py`)**: 作为系统的入口，提供所有功能的操作界面。
- **数据处理模块 (`data_loader.py`, `face_detector.py`)**: 负责数据集的加载、预处理、人脸检测与提取。

- **模型模块 (`svm_model.py`, `cnn_model.py`)**: 包含 SVM 和 CNN 两种识别模型的实现，负责模型的训练和预测。
- **评估模块 (`evaluator.py`)**: 负责计算模型的各项性能指标，并提供模型对比功能。

2.2 数据处理模块

数据处理是人脸识别流程的起点，其质量直接影响模型性能。本模块包含数据加载和人脸检测两部分。

2.2.1 数据加载 (`data_loader.py`)

`FaceDataLoader` 类负责加载和预处理数据集。它支持多种数据源：

- **内置数据集**: 通过 `scikit-learn` 加载 Olivetti Faces 和 LFW (Labeled Faces in the Wild) 数据集。
- **自定义数据集**: 从指定目录结构中加载图像。目录的每个子文件夹代表一个类别（人）。
- **样本数据集**: 用于在无法加载其他数据集时进行快速测试。

加载流程包括读取图像、转换为灰度图、统一尺寸（默认为 64x64）、归一化到 [0, 1] 范围，并最终划分为训练集和测试集。

2.2.2 人脸检测与提取 (`face_detector.py`)

`FaceDetector` 类用于从图像中定位并提取人脸区域。

- **检测方法**: 支持 Haar 级联分类器和基于深度学习的 DNN 检测器两种方法，并能自动选择最优方法。
- **人脸提取**: 检测到人脸后，提取最大的人脸区域。
- **预处理**: 对提取的人脸进行最终的预处理，如直方图均衡化（`equalizeHist`），以增强图像对比度，消除光照影响。这是保证训练和预测输入一致性的关键步骤。

2.3 模型模块

模型模块是系统的核心，我们实现了 SVM 和 CNN 两种模型，它们都继承自抽象基类 'BaseFaceRecognitionModel'。

2.3.1 SVM 模型 (svm_model.py)

'SVMFaceRecognitionModel'实现了一个完整的人脸识别流水线：

1. **特征提取**: 将图像从高维像素空间转换为更具判别力的特征空间。支持：
   - **HOG (Histogram of Oriented Gradients)**: 捕捉人脸的轮廓和形状信息。
   - **LBP (Local Binary Patterns)**: 捕捉人脸的纹理信息。
   - **HOG+LBP**: 结合两者优势。
   - **原始像素**: 作为基线对比。

2. **降维**: 使用主成分分析（PCA）对提取的特征进行降维，减少计算量并去除冗余信息。
3. **分类**: 使用支持向量机（SVC）进行分类。
4. **超参数优化**: 通过网格搜索（‘GridSearchCV‘）自动寻找最优的 SVM 参数（如 C 和 gamma）。
5. **集成学习**: 可选地使用投票分类器（‘VotingClassifier‘）集成多个不同核函数的 SVM 模型，以提高稳定性和准确率。

2.3.2 CNN 模型 (cnn_model.py)

‘PyTorchCNNFaceRecognitionModel‘基于 PyTorch 框架实现。

- **网络架构**: 设计了 ‘ImprovedFaceRecognitionNet‘，一个现代化的 CNN 架构。
  - **残差块 (ResidualBlock)**: 借鉴 ResNet 思想，有效解决了深度网络中的梯度消失问题，使网络可以更深。
  - **空间注意力机制 (SpatialAttention)**: 使网络能够自适应地关注图像中的关键区域（如眼睛、鼻子、嘴巴），抑制无关背景。
- **数据增强**: 在训练过程中，对输入图像进行随机翻转、旋转、缩放和颜色抖动，增加数据多样性，提高模型的泛化能力。
- **训练策略**: 采用高级训练策略，包括带标签平滑的交叉熵损失函数、Adam 优化器和学习率动态调整（‘ReduceLROnPlateau‘），以实现更稳定和高效的训练。

2.4 评估模块 (evaluator.py)

‘ModelEvaluator‘类负责对训练好的模型进行客观、量化的性能评估。

- **性能指标**: 计算准确率（Accuracy）、精确率（Precision）、召回率（Recall）和 F1 分数（F1-Score）等常用指标。
- **分类报告**: 生成详细的分类报告，展示每个类别的性能。
- **模型对比**: 将多个模型的评估结果汇总到一张表格中，方便直观比较。

3 实现细节

本节展示部分关键功能的代码实现。

3.1 SVM 模型流水线

SVM 模型的训练过程被封装在一个 `scikit-learn` 的 `Pipeline` 中，这确保了数据处理步骤的一致性。以下是使用网格搜索优化单一 SVM 模型的代码片段。

```
    # svm_model.py
    def _train_single_model(self, X_features, y_encoded, n_components):
        # 创建处理管道
        self.pipeline = Pipeline([
            ('scaler', StandardScaler()),
            ('pca', PCA(n_components=n_components, whiten=True)),
```

```
            ('svm', SVC(probability=True, class_weight=self.class_weight))
        ])

        # 定义要搜索的参数网格
        param_grid = {
            'svm__C': [0.1, 1, 10, 100],
            'svm__gamma': ['scale', 'auto', 0.01, 0.1]
        }

        # 使用5折交叉验证进行网格搜索
        cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
        grid_search = GridSearchCV(
            self.pipeline, param_grid, cv=cv,
            scoring='accuracy', verbose=1
        )

        print("执行网格搜索以优化SVM参数...")
        grid_search.fit(X_features, y_encoded)

        # 使用找到的最佳模型
        self.pipeline = grid_search.best_estimator_
```

Listing 1: SVM 模型网格搜索与训练

3.2 CNN 网络架构

CNN 模型的核心是 'ImprovedFaceRecognitionNet'。以下代码展示了其网络结构，特别是残差块和注意力机制的集成。

```
# cnn_model.py
class ImprovedFaceRecognitionNet(nn.Module):
    def __init__(self, ...):
        super().__init__()
        # ...
        # 特征提取层
        self.layer1 = nn.Sequential(ResidualBlock(64, 64), ...)
        self.layer2 = nn.Sequential(
            ResidualBlock(64, 128, stride=2),
            SpatialAttention(kernel_size=7)  # 添加注意力
        )
        # ...
        self.global_avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        # ...

    def forward(self, x):
        x = self.conv_init(x)
        x = self.layer1(x)
        x = self.layer2(x) # 应用残差块和注意力
        # ...
        x = self.global_avg_pool(x)
        features = self.embedding_layer(x)
        output = self.classifier(features)
        return output
```

Listing 2: 改进的 CNN 网络架构

3.3 GUI 模型训练逻辑

GUI 通过多线程来执行耗时的训练任务，避免界面冻结。训练完成后，通过 ‘root.after‘在主线程中更新 UI。

```python
# gui/main_window.py
def train_model(self):
    # ...
    model_name = self.model_var.get()
    model = self.models[model_name]

    # 定义训练任务
    def train():
        try:
            if model_name == "PyTorch CNN":
                train_info = model.train(self.X_train, self.y_train,
                                         self.X_test, self.y_test, ...)
            else:
                train_info = model.train(self.X_train, self.y_train)

            # 训练完成后，在主线程更新UI
            self.root.after(0, lambda: self.on_model_trained(model_name, train_info))
        except Exception as e:
            self.root.after(0, lambda: self.on_error(f"训练失败: {e}"))

    # 在新线程中启动训练
    threading.Thread(target=train, daemon=True).start()
```

Listing 3: GUI 中的异步模型训练

4 实验与结果分析

4.1 实验环境与数据集

- **硬件环境**: Intel Core i7 CPU, 16GB RAM, NVIDIA GeForce RTX 3060 GPU
- **软件环境**: Windows 11, Python 3.9, PyTorch 1.12, scikit-learn 1.1, OpenCV 4.6
- **数据集**: 本实验主要使用 Olivetti Faces 数据集。该数据集包含 40 个不同的人，每人 10 张 64x64 的灰度图像，共 400 张。我们按照 80% 训练集和 20% 测试集的比例进行划分。

4.2 实验结果

我们分别对 SVM 模型（使用不同特征）和 CNN 模型进行了训练和评估。

4.2.1 SVM 模型性能

我们测试了 SVM 在使用不同特征提取方法时的性能，结果如表**??**所示。

表 1: SVM 模型在 Olivetti 数据集上的性能

特征方法	准确率	精确率 (宏)	召回率 (宏)	F1 分数 (宏)
原始像素	0.7250	0.7315	0.7250	0.7198
HOG	0.9500	0.9583	0.9500	0.9497
LBP	0.8875	0.8952	0.8875	0.8864
HOG+LBP	**0.9750**	**0.9792**	**0.9750**	**0.9748**

从表中可以看出，使用手工设计的特征（HOG, LBP）显著优于直接使用原始像素。HOG 特征在捕捉人脸结构方面表现出色，而 HOG 与 LBP 的结合则达到了最佳性能，准确率达到 97.5%。这证明了特征工程在传统机器学习方法中的重要性。

4.2.2 CNN 模型性能

CNN 模型经过 30 个 epoch 的训练后，其在测试集上的性能非常出色。训练过程中的损失和准确率变化如图**??**所示。

图 1: CNN 模型训练历史曲线（左：损失，右：准确率）

从图中可以看出，训练损失和验证损失都稳步下降，而准确率则稳步上升，最终收敛在一个较高的水平，没有出现明显的过拟合现象。

4.3 模型对比分析

我们将表现最好的 SVM 模型（HOG+LBP 特征）与 CNN 模型进行综合比较，结果如表**??**所示。

表 2: SVM 与 CNN 模型性能对比

指标	SVM (HOG+LBP)	PyTorch CNN
准确率	0.9750	**0.9875**
F1 分数 (宏)	0.9748	**0.9875**
训练时间	**∼30 秒**	∼3 分钟
模型复杂度	低	高
特征工程	手动设计	**自动学习**

结果分析:

- **准确率**: CNN 模型凭借其端到端的特征学习能力，在准确率上略微超过了精心设计的 SVM 模型，达到了 98.75%。
- **训练时间**: SVM 模型的训练速度远快于 CNN 模型。这主要是因为 SVM 的计算量集中在特征提取和相对简单的优化问题上，而 CNN 需要通过反向传播迭代更新数百万个参数。
- **开发成本**: SVM 模型需要大量关于特征工程的先验知识来设计有效的特征提取器。而 CNN 模型将这一过程自动化，开发者可以更专注于网络架构的设计。

5 结论

本项目成功设计并实现了一个功能全面的人脸识别系统，集成了 SVM 和 CNN 两种主流方法。通过实验对比，我们得出以下结论：

1. 对于传统机器学习方法，有效的特征工程是成功的关键。基于 HOG+LBP 特征的 SVM 模型在小规模、规整的数据集（如 Olivetti）上能够取得非常高的识别精度。

2. 基于深度学习的 CNN 模型展现了更强的性能和潜力。其端到端的学习方式无需手动设计特征，在准确率上达到了更高的水平，并且具有更好的泛化潜力，尤其是在处理更复杂、更多样化的数据集（如 LFW）时优势会更加明显。

3. 两种方法各有优劣。SVM 在训练速度和资源需求上占优，适合快速原型开发或资源受限的场景。CNN 则在性能上领先，是当前大规模、高精度人脸识别应用的主流选择。

未来工作: 未来的改进方向可以包括：使用更大规模的数据集（如 CASIA-WebFace）进行训练以提高模型泛化能力；引入更先进的 CNN 架构（如 MobileFaceNet）和损失函数（如 ArcFace Loss）来进一步提升识别精度；以及将训练好的模型部署到移动端或嵌入式设备上，实现真正的落地应用。

参考文献

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886-893, 2005.

[2] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971-987, 2002.

[3] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine Learning*, vol. 20, no. 3, pp. 273-297, 1995.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770-778, 2016.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.