基于残差网络的手语识别系统

V1.0

设计说明书

目录

一、引言

1. 背景

手语是一种重要的沟通方式，特别是对于听力障碍者或语言障碍者。与口头语言相比，手语具有动态性、多样性和非线性，传统上，手语识别主要依赖于计算机视觉和机器学习技术，以识别手势并将其转换为文字或语音。然而，由于手语的复杂性和多样性，以及手势识别中存在的挑战，例如光照变化、手势之间的相似性等，传统方法的性能可能受到限制。

近年来，深度学习技术的迅速发展为手语识别领域带来了新的机遇和挑战。深度学习模型能够从大量的数据中学习到表示特征，并且在许多计算机视觉任务中取得了显著的性能提升。其中，残差网络（Residual Networks）作为一种深度学习架构，被证明在图像分类、目标检测等任务中具有很好的效果，其通过引入残差连接和跨层连接，这样，网络不再需要直接拟合输入和输出之间的映射关系，而是通过学习残差来逐渐逼近理想映射，从而降低了学习的难度。残差连接的引入还有助于避免了梯度消失问题，使得深层网络的训练变得更加稳定，有效提高了网络的性能和收敛速度。这种优势使得残差网络成为处理复杂任务和大规模数据的理想选择，因此，在手语识别领域中应用残差网络可能为系统的性能和效率带来显著提升。

当前关于基于残差网络的手语识别系统的研究相对较少。现有的工作大多集中在传统深度学习网络或基于手工特征提取的方法上，对于残差网络在手语识别中的应用尚未充分挖掘。本文开发的基于残差网络的手语识别系统有望为聋哑人群体提供一个更加可靠、实用的交流工具，探索并利用残差网络的特性，构建一个高效、准确的手语识别系统。

2. 编写目的

（1）提高日常沟通效率：将残差网络引入手语识别中，使得系统能够更快速、更准确地将手势转换为文字或语音输出，帮助听力障碍者和语言障碍者在日常生活中更方便地与他人沟通交流。

（2）适应多样化手势场景：手语的手势具有多样性，且手语用户可能会在不同的环境和条件下进行交流。基于残差网络的手语识别系统基于深度学习模型，引入残差网络，使其具有较强的适应性，能够识别各种不同类型的手势，并在不同的光照、背景等条件下保持稳定的识别性能。

（3）实时识别：对于实时交流和即时反馈的场景，如视频通话或现场翻译，手语识别系统需要具备快速响应的能力。利用残差网络的并行计算优势和高效的模型结构，可实现对手语视频的实时识别，提高用户的交流效率和体验。

二、 程序概述

1. 软件简介

（1）Python

Python是一种高级编程语言，具有简洁、易读的语法特点，同时拥有庞大的第三方库支持和活跃的开源社区。其在科学计算、数据分析、人工智能等领域的广泛应用，其丰富的库和工具能够支持研究人员进行复杂的数据处理、模型建立与实验设计，其具有开放、易用、跨平台性等特性。

（2）OpenCV

OpenCV是一个开源的计算机视觉库，提供了丰富的图像处理和计算机视觉算法。在手语识别系统中，OpenCV可以用于图像的预处理，例如手势检测、手势跟踪、背景去除等。通过OpenCV，可以对输入的手语视频进行处理，使其适合用于训练和测试模型。

（3）TensorFlow

TensorFlow是一个开源的机器学习框架，由 Google 开发和维护。它提供了丰富的工具和库，用于构建和训练各种机器学习模型，特别是深度学习模型。TensorFlow最大的特点之一是其灵活性和可扩展性，使得用户可以在各种硬件平台上进行高效的模型训练和部署，包括 CPU、GPU 和 TPU。其采用的静态计算图和动态计算图的混合方式，使得用户可以根据需求选择适合的计算模式。

（4）Pycharm

PyCharm是由JetBrains公司开发的一款专业的Python集成开发环境（IDE），提供了丰富的功能和工具，包括智能代码编辑、代码自动完成、代码调试、版本控制、代码检查等，使开发者能够高效地编写、调试和管理Python代码。PyCharm支持多种Python开发框架，提供了丰富的插件生态系统，可根据需要扩展功能。且具有友好的用户界面。

（5）Keras

Keras是一个高级神经网络API，可以在 TensorFlow、Microsoft Cognitive Toolkit、Theano 等深度学习框架的基础上运行。其设计简洁，易于使用，可快速建立和实验各种神经网络模型，用于图像分类、文本生成或其他任务。Keras 提供

丰富的预定义层和模型，同时也支持自定义层和模型，其设计理念注重用户友好性、模块化和可扩展性。

2. 功能

有以下主要功能：

1) 手势识别：利用残差网络提取手语图像中的特征，通过堆叠残差块构建深层的特征提取网络。残差块帮助网络学习较为抽象和丰富的特征表示，从而更好地区分不同手语手势之间的差异，然后将提取的特征传递给全连接层进行分类，最终实现对手语的识别。

2) 手势分类：手语分类利用残差网络作为分类器。输入手语图像数据，经过残差网络的卷积和池化操作后，将提取的特征通过全连接层进行分类，实现对手语图像的分类。

3. 开发流程

基于残差网络的手语识别系统开发流程如下：

1）手语手势图像数据集采集

数据集选用美国手语（ASL）数据集ASL alphabet，包含超过78000张手语手势图像，涵盖了26个英文字母，每个字母包含3000张手势图像。

2）数据增强

明亮度调整（变亮、变暗），尺寸放大及缩小，旋转（90°、135°、180°），增加随机噪声（均匀噪声、高斯噪声），水平及竖直翻转等方式扩充数据集。

3）模型选择：

以ResNet18作为预训练模型，损失函数选择交叉熵损失函数，优化器选择为自适应矩估计优化器。

6）模型训练：

将改进后的模型进行训练，设置训练轮次，本系统实验设置轮次为30轮，每3轮学习率下降0.1~0.8。

7）界面设计：

根据功能模块和用户场景，设计合适的信息架构。确定界面中各个元素的组织结构、布局和关系。

4. 实现方法

收集手语手势图像并构建数据集，训练集、验证集和测试集的比例划分为7：

1.5：1.5；然后对图像大小进行归一化、灰度化；设置30轮训练轮次以及学习率，使用TensorFlow加载预训练的ResNet18模型，根据手势类别数量替换模型的全连接层。然后选择交叉熵损失函数作为分类任务的损失函数，使用自适应矩估计优化器Adagrad优化模型参数；加载手语数据集并设置加载器，用TensorFlow对训练集上进行模型编译和训练；每个epoch结束后用验证集评估模型的性能并调整学习率和超参数；最后用测试集对训练好的模型进行评估，计算AP值、F1指数等指标，验证模型在手语识别任务上的性能表现。

（1）收集数据集

数据集选用美国手语（ASL）数据集ASL alphabet，包含超过78000张手语手势图像，涵盖了26个英文字母，每个字母包含3000张手势图像，A：将大拇指伸出，其余四指合拢成拳。B：将大拇指伸出，其余四指伸直。C：将大拇指和食指张开成C形。D：将大拇指伸出，其他四指弯曲放在掌心上。E：伸出所有的手指。F：将大拇指弯曲，放在食指的侧面。G：将大拇指伸出，其他四指弯曲成手掌形。H：将大拇指伸出，食指和中指并拢。I：只伸出食指。J：将大拇指伸出，其余四指弯曲形成J形。K：将大拇指和中指并拢，其他三指弯曲。L：将大拇指和食指成L形。M：将大拇指和食指张开成M形。N：将大拇指和中指并拢，其他三指伸出。O：将所有的手指合拢成O形。P：将大拇指和食指并拢，中指和无名指并拢，小指伸出。Q：将大拇指伸出，中指和无名指并拢，其他两指弯曲。R：将大拇指和食指成R形。S：将食指和中指并拢。T：将大拇指伸出，其他四指并拢。U：将食指和中指伸出，其余三指弯曲。V：将食指和中指分开成V形。W：将食指、中指和无名指分开，形成W形。X：将大拇指和食指交叉。Y：将大拇指和食指分开成Y形。Z：将大拇指和无名指成Z形。数据集部分图像如图 1 所示。

图1　ASL alphabet

（2）数据增强与预处理

对ASL alphabet进行数据清洗、归一化、标准化，然后对图像进行旋转、缩放、平移、翻转等操作，然后进行随机旋转、裁切、添加高斯噪声，增加数据多样性和鲁棒性，确保数据的质量和可用性，减少模型训练的时间和资源消耗，部分图像数据增强情况如图 2 所示。

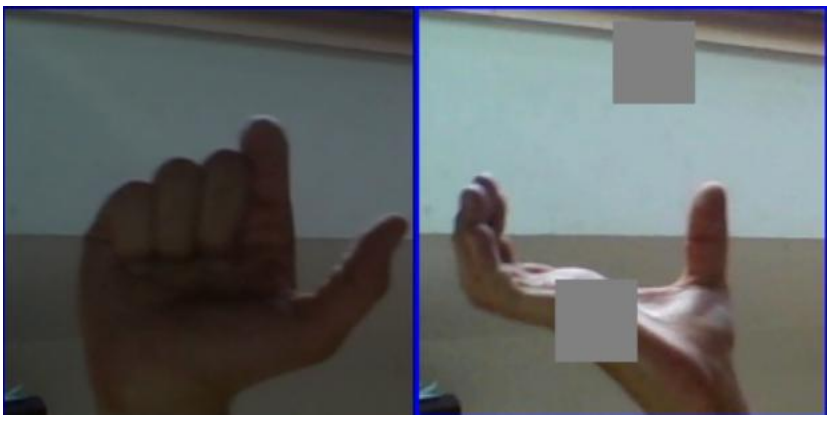

图2　图像数据增强

（3）模型选择

1）ResNet18的基本原理

ResNet18是一种深度卷积神经网络（CNN）模型，其在图像分类、目标检测等领域表现出色。模型核心思想是通过“残差连接”（shortcuts）来解决深度神经网络中梯度消失或爆炸的问题，使得网络能够更深地学习而不影响其性能。

2）残差块的设计

在ResNet18中，每个残差块（Residual Block）由两个或更多的卷积层组成，通过“捷径”（Shortcut Connection）将输入信号直接传递到输出端，与经过非线性变换后的信号相加。这种设计允许网络学习输入与输出之间的残差，而不是直接学习复杂的映射关系。

3）网络结构

ResNet18的网络结构如图 3 所示，通常包括以下几个部分：

卷积层：网络的第一层是一个7x7的卷积层，步长为2，用于提取输入图像的低级特征。

残差块：接下来的部分由多个残差块组成，每个残差块包含两个3x3的卷积层，每个卷积层后面都跟着一个批量归一化（Batch Normalization）层和一个ReLU激活函数。

全连接层：在经过多个残差块处理后，特征图的大小逐渐减小，通道数逐渐增加。最后，特征图会被展平并输入到一个全连接层中，以输出最终的分类结果 。

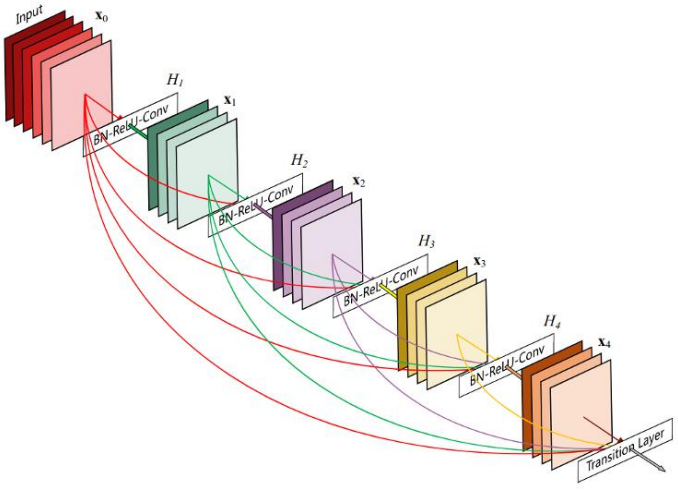

图3 ResNet网络结构

（4）核心原理

当输入和输出的特征图大小相同时，使用恒等映射直接进行特征映射，如图 4 所示，输入信号直接传递到输出端，与经过非线性变换后的信号相加。这种方式不需要额外的参数，因为它直接将输入信号复制到输出。当输入和输出的特征图大小不同时，使用残差映射进行残差连接，如图 5 所示，输入信号经过一个非线性变换，如ReLU激活函数后，与原始输入信号相减。这种方式需要额外的参数，因为它涉及到对输入信号的变换捷径分支的卷积核尺寸为1×1×1，步长为 2×2×2，用于将特征图尺寸减小为原来的 1/2，同时将通道数

变为原来的 2 倍，当网络已经达到最优状态时，残差映射可以被推导为0，只剩下恒等映射，这样理论上网络一直处于最优状态，网络的性能也就不会随着深度增加而降低。

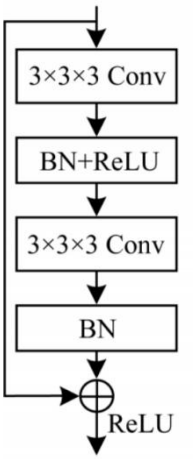

图4 恒等映射

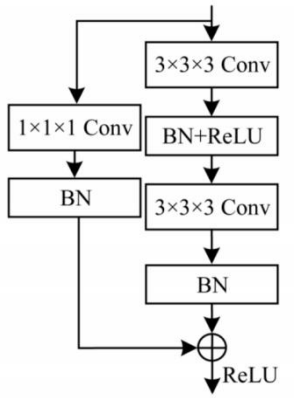

图5 残差映射

5. 系统框图

程序中主要系统架构设计关系如图 6 所示。

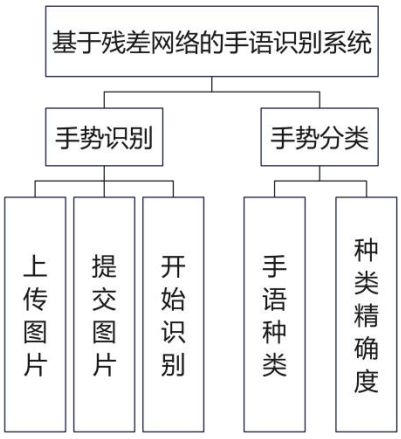

图6 程序模块关系图

6. 系统开发流程图

系统开发流程如图 7 所示，依次为原始数据集的构建、对图像数据明暗亮度增强及尺寸归一化预处理、划分训练集测试集和验证集、模型训练、模型识别与分类测试、构建手语识别系统。

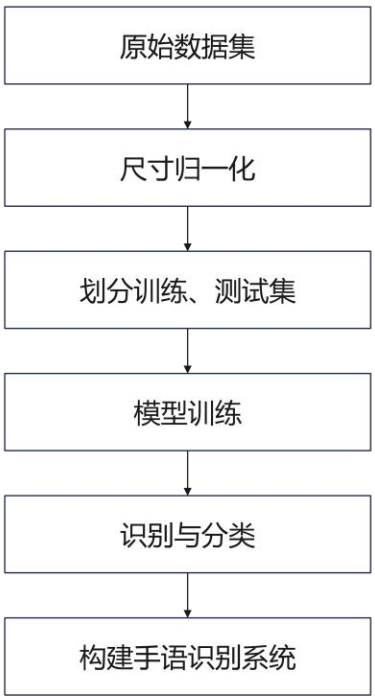

图7 系统开发流程

三、开发环境

1) 硬件环境：CPU：Intel Xeon E5-2667v4 GPU:NVIDA RTX4090 RAM：32GB

2) 系统环境：Windows10 x64

3) 编程语言：Python

4) 主要工具：Python、OpenCV、TensorFlow、Pycharm、Keras

四、系统UI

1. 主界面

基于残差网络的手语识别系统主界面如图 8 所示，点击“上传图片”按钮将手语手势图片进行上传操作，上传图片完成后，点击“开始识别”按钮对已上传的手势图片进行识别与分类操作，上传图片操作如图 9 所示。

图8 主界面

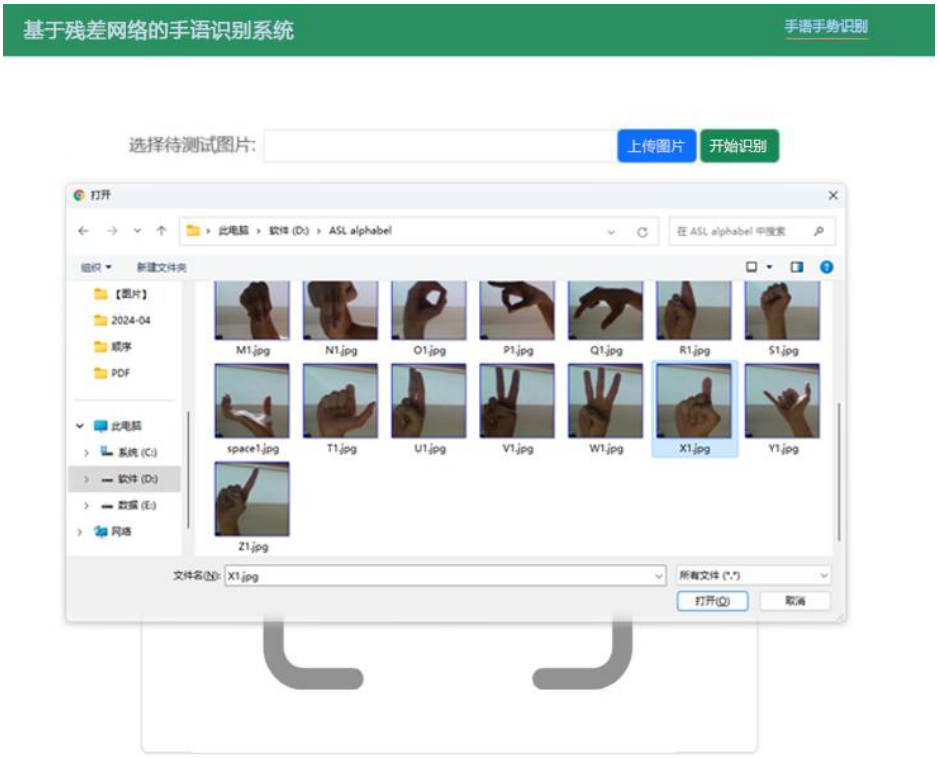

图9 手势图片上传

2. 检测界面

基于残差网络的手语识别系统功能共分为两类，分别为分类和识别，在手势图片识别完毕后，会在手势图片的下方以文字语言形式显示分类结果和识别准确

率，如图 10、图 11 所示。

图10　检测效果图一

图11　检测效果图二

五、程序说明

1. 预处理

数据预处理，首先定义图像处理函数，然后读取图片和label，将读取的图片数据加载到imgs[]列表中，最后将图片的label加载到labels[]中，与上方的imgs索引对应，核心代码如图 12 所示。

```
# 定义图像处理函数
def read_img(path):
    print("数据集地址: "+path)
    imgs = []
    labels = []
    for root, dirs, files in tqdm(os.walk(path)):
        for file in files:
            # print(path+'/'+file+'/'+folder)
            # 读取的图片
            img = cv2.imread(os.path.join(root, file))
            # 将读取的图片数据加载到imgs[]列表中
            imgs.append(img)
            # 将图片的label加载到labels[]中，与上方的imgs索引对应
            labels.append(str(os.path.basename(root)))
    return imgs,labels
```

图12 预处理代码

2. 模型训练

模型训练，共计训练轮次为30轮，即30个epoch，设置批处理batch_size为64，核心代码如图 13 所示。

```
from tensorflow.keras.callbacks import (
    ReduceLROnPlateau,
    EarlyStopping,
    ModelCheckpoint,
    TensorBoard)

# 编译模型来配置学习过程
ResNet_model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',metrics=['accuracy'])
callbacks = [
#     ReduceLROnPlateau(verbose=1),
    # 提前结束解决过拟合
    # EarlyStopping(patience=10, verbose=1),
    # 保存模型
    ModelCheckpoint(checkpoints + 'resnet_train_{epoch}.tf', monitor='accuracy',verbose=0,
                    # 当设置为True时，将只保存在验证集上性能最好的模型
                    save_best_only=True, save_weights_only=True,
                    # CheckPoint之间的间隔的epoch数
                    period=2),
    TensorBoard(log_dir='logs')
]
# 模型训练
history = ResNet_model.fit(data_train, epochs = epoch,callbacks=callbacks,validation_data = data_test)
```

图13 模型训练参数设置

3. 训练曲线

训练和测试集的准确率如下，在第4个epoch时准确率开始快速增加，在第16个epoch时趋于稳定，达到百分之97.55的准确率,具体如图 14 所示。

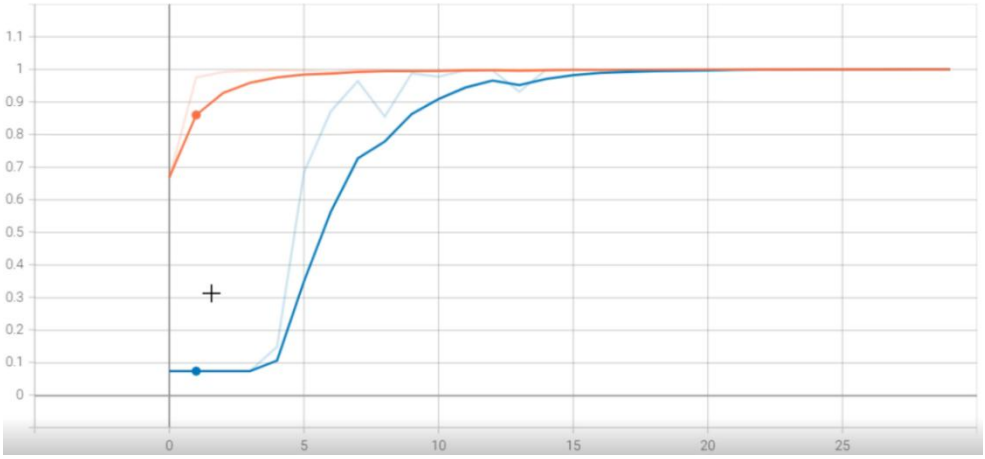

图14 训练曲线