目录

[1 绪论 3](#_Toc134802806)

[1.1 项目背景与意义 3](#_Toc134802807)

[1.2 研究与产业现状 3](#_Toc134802808)

[1.2.1 研究现状 3](#_Toc134802809)

[1.2.2 产业现状 4](#_Toc134802810)

[1.2.3 现状分析与项目目标 4](#_Toc134802811)

[1.3 文章结构 4](#_Toc134802812)

[2 手势运动估计的关键问题与解决方案 5](#_Toc134802813)

[2.1 关键问题分析 5](#_Toc134802814)

[2.1.1 深度学习模型原理 5](#_Toc134802815)

[2.1.2 数据集选择 5](#_Toc134802816)

[2.1.3 实际场景展示 5](#_Toc134802817)

[2.2 方案设计 6](#_Toc134802818)

[2.3 技术选型 6](#_Toc134802819)

[3 手部运动数据集设计 7](#_Toc134802820)

[3.1 环境说明 7](#_Toc134802821)

[3.2 7](#_Toc134802822)

[3.2.1 Artiboost模型 7](#_Toc134802823)

[3.2.2 TriHorn-Net模型 9](#_Toc134802824)

[3.2.3 模型对比 11](#_Toc134802825)

[3.2.4 模型训练和测试 11](#_Toc134802826)

[3.3 数据输入与输出 11](#_Toc134802827)

[3.3.1 数据集输入 11](#_Toc134802828)

[3.3.2 关键数据输出 11](#_Toc134802829)

[3.4 模型封装与优化 11](#_Toc134802830)

[4 应用软件设计 12](#_Toc134802831)

[4.1 软件架构 12](#_Toc134802832)

[4.2 数据集生成 12](#_Toc134802833)

[4.3 Socket通信与优化 12](#_Toc134802834)

[4.3.1 Socket通信设计 12](#_Toc134802835)

[4.3.2 Socket通信优化。 13](#_Toc134802836)

[4.4 相机画面截取 14](#_Toc134802837)

[5 结果展示与分析 14](#_Toc134802838)

[5.1 模型训练与估计结果 14](#_Toc134802839)

[5.1.1 Artiboost模型。 14](#_Toc134802840)

[5.1.2 TriHorn-Net模型 14](#_Toc134802841)

[5.2 数据集对比实验结果 15](#_Toc134802842)

[5.2.1 模型默认数据集 15](#_Toc134802843)

[5.2.2 70%源数据集和30%合成数据集 15](#_Toc134802844)

[5.2.3 50%源数据集和50%合成数据集 15](#_Toc134802845)

[5.2.4 100%合成数据集 15](#_Toc134802846)

[6 总结 15](#_Toc134802847)

[6.1 总结 15](#_Toc134802848)

[6.2 趋势展望 15](#_Toc134802849)

# ****绪论****

## 1.1 项目背景与意义

手势作为一种新型的交互手段可以实际应用于一些具有交互性质的场景，例如：文化遗址、博物馆等等。传统的展馆多数是以图片和文字作为交互的载体，能够传达给的信息十分有限。可能由于时代或环境因素，人们无法体会到图片和文字信息背后所包含的内容，而手势作为一种新型的交互工具，可以很大程度地弥补图片和文字的不足。随着技术的发展以及人们对于信息获取需求的不断多样化，基于AR、VR的交互式体验逐渐开始流行起来。然而手势交互最大的难点就在于姿态估计，所以如何提升手部信息的精准性能够大幅提高交互式体验在实际场景的可行性。

在现有的体感交互设备中，大多都是专业且昂贵的动作捕捉设备，可以通过识别人体动作、手势等信息，最后反馈到计算机中实时显示，从而实现更加自然的人机交互方式。但是体感交互设备操作繁琐，通常需要在手部绑定特定的传感器来识别手势。而依靠神经网络模型估计手势姿态可以做到无需专业设备，只需要采集训练数据来不断学习，提高手势估计的精度，大大降低了交互成本，同时还具有较高的泛用性。

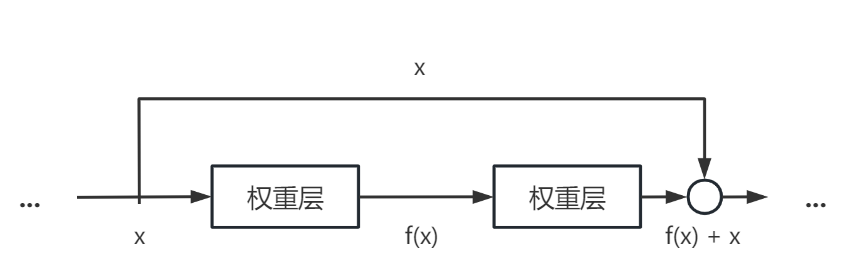
网络模型设计时，除了神经网络算法重要以外，训练和测试数据集同样也非常重要。采集真实世界的数据集可能会受到环境、设备的制约，而使用虚拟场景合成手部数据集的方法能够快速、高效地获取到大量的手部数据，避免了复杂的专业设备的使用。并且这些虚拟空间手部数据同样具有有效的手部信息，从而保证了真实场景数据集和合成数据集的有效性，避免了只通过现实数据采集的限制和约束。

## 1.2 研究与产业现状

### ****1.2.1 研究现状****

随着深度学习、神经网络和图像处理的不断发展，近年来，计算机视觉领域出现许多优秀的算法架构，使得计算机视觉技术正在被广泛使用，横跨了多个领域，例如：自动驾驶、工业领域和医疗系统等等。计算机逐渐可以做一些更为精准的图像预测与数据增强方面的应用。

**卷积神经网络。**卷积神经网络避免了对图像的复杂前期预处理，可以将原始图像作为输入数据放入模型中，通过梯度下降和更新的方式训练模型。Alex Krizhevsky、Hinton等人在2012年提出的AlexNet[1], 首次提出了Relu激活函数以及最大化池。相比于Sigmoid激活函数，训练效率成倍提升，从而获得了当时ImageNet[2]竞赛冠军。Kaiming He等人于2015年提出的Resnet[3]开创了残差网络模型, 解决了深度神经网络的“退化”问题。“退化”是指使用浅层直接堆叠成深层网络，不仅难以利用深层网络强大的特征提取能力，而且准确率会下降。由于Resnet[3]的泛用性较高，因此许多数据增强模型会优先选择Resnet作为预训练模型，减少大量的训练资源和时间。下文中的Artiboost模型同样也是使用Resnet-34作为预训练模型。



**手势识别。**手势识别在整个计算机视觉中一直是一个热门的领域，但是由于手势在人体学中手势属于比较复杂的部分，因此手势识别相关的网络模型一直处于研究阶段。Siyuan Yang等人在2020年提出使用弱监督的三维姿态估计技术通过手势识别和3D手部姿势相互评估[4]，使得手势识别和手部姿态可以共同学习，相互促进和影响。Anil Armagan等人提出HANDS’19[5]，评估深度和颜色模态对三维手部姿态估计的影响。在以深度图作为数据集方面，还有使用不同的手摆出不同姿势的噪声深度图像的手部姿态模型[6]。

### 1.2.2 产业现状

近年来随着体感交互设备的发展，基于AR、VR的交互性大大提升，其中也包括手势姿态与交互的应用。例如LeapMotion设备，产品等同于一个小型摄像头，但是内部使用了光学部件来捕捉跟踪手势，并且可以生成对应的虚拟手来模拟捕捉到的真实手的动作。针对不同的虚拟动作，开发者可以自行开发不同的手势，用作手势识别或者物体交互。但也正是只使用了光学部件，手部不需要绑定任何的传感器，识别的手势模型经常会出现关节的错位甚至模型的丢失，更加无法处理手部有遮挡的情况。如果想要更稳定的手势识别设备，就需要穿戴传感设备传输精确数据。

（leapMotion 手势识别图、识别出现问题图）

而计算机视觉现阶段主要还停留在科研阶段，实际实质化产出内容较少。但是未来会有更多的实际场景中许多地方都需要使用计算机视觉相关的技术，例如：工业领域、医疗诊断、图像识别等等。在人脸识别领域，通过计算机进行支付验证、人脸跟踪的技术已经相对成熟。在自然语言处理领域，由于GPT-3[7]的出现，基于深度学习的大模型基本已经能够与人正常交流，并开始逐步进入了市场应用，被人们广泛使用。现今随着类似于gpt-3[7]的大模型越来越流行，未来会有更多领域使用大模型的方法进行训练和测试，其中也包括计算机视觉。但是由于手势的复杂性导致目前暂时没有一个非常优秀的网络模型来解决姿态估计的问题，因此整个手势识别领域现阶段还只是处于研究阶段。

### 1.2.3 现状分析与项目目标

在计算机视觉学术领域提出的许多前瞻性的新方法，造就了手势识别的飞速发展。虽然现阶段已经有许多开源的手势数据集，但是真实手部数据集生成工作依然是费时费力的，无法像合成数据集一样做到自由高效地批量生成数据集。同时多数模型的研究都只停留在理论层面的结论，没有考虑在实际场景中的应用。从而导致与实际应用对接较为繁琐，甚至可能需要进行大量的修改才能满足工程上的需求。因此我们的方法是在前人的网络模型基础上进行重构，使用其网络模型的核心思想，针对模型的输入改造数据集，然后将整个模型进行封装，从而能够应用到一些实际场景中。

本项目的目标是构造一个面向虚拟场景的应用。首先需要建立数据集生成工具，该工具针对神经网络模型的输入要求使用骨骼动画化驱动虚拟手批量渲染生成数据集，作为原来模型数据集的补充数据集。基于如上数据集，交由手部姿态估计算法模型进行训练，然后封装成一个手部姿态估计工具。最后应用于下游的实际应用中，根据输入的手势数据在应用的虚拟场景中实时显示和交互。

## 1.3 文章结构

本文总共分为六个章节

第一章：绪论。

第二章：关键问题分析与解决

第三章：神经网络模型设计

第四章：模块设计

第五章：实验对比与分析

第六章：总结与展望

# 2 手势运动估计的关键问题与解决方案

## 2.1 关键问题分析

手势运动估计的数据集生成与应用一共分为三个模块，分别为数据集生成、手势估计模型以及实际场景应用。而其中，手势估计模型是三者的核心。数据集的生成需要首先分析模型的输入参数，修改输入参数，根据输入参数批量生成补充数据集，然后放入模型中进行训练。实际场景应用则是在模型训练完成后，使用最新迭代的模型进行估计，最后得到的结果放入场景中进行展示和交互。因此，手势估计模型设计至关重要，只有完成整个模型的训练、测试以及重构才能继续完成其他两个模块的工作。

（模块图）

### 2.1.1 深度学习模型原理

由于在此之前并未学习过深度学习的模型原理，所参考的论文也没有对基础知识的详细描述，因此对模型和论文中的大量方法和术语并不清楚。在前期工作中需要了解大量的深度学习的基础知识，从最简单的有监督学习开始，再到需要使用到神经网络来解决的复杂问题。之后再进一步了解各种神经网络模型，以及网络每一层所使用的方法。

### 2.1.2 数据集选择

神经网络模型通常都会使用一些开源的数据集进行训练和测试，从而有助于将自身的网络模型与他人的模型相比较，以此来说明自身模型的误差率和稳定性。但是通常这些开源的数据集并非是直接将图片的信息完整地输入至网络模型中，相反，往往在输入前已经对图片进行了一些预处理操作，例如：在输入前对图像预分割，在训练过程中模型只会关注边界框内的像素信息，减少图像其他区域的信息干扰。但是在没有其他工具的帮助下，使用摄影设备截取的画面不仅仅包含手势信息，还包含其他物体和环境元素。当截取到的图片信息直接交由网络模型训练时，模型将无法准确得到其中的手势信息，导致最后训练的结果与实际值相差甚远。

### 2.1.3 实际场景展示

在实际场景中首先需要保证估计得出的结果的准确性，在训练过程中数据集大小问题、数据集本身的问题、训练周期问题以及损失函数选择的问题都可能会影响模型最后训练的结果。其次还需要保证整个项目的实时性，模型的处理过于复杂、数据传输方式的选择不当等等都会耗费大量的时间，造成最后实时的效率低下。

## 2.2 方案设计

首先是模型框架原理问题。在项目前期工作中，需要清楚了解基础的深度学习基本框架、基本原理，了解整个模型的流程以及每一个步骤的详细内容。在掌握了一些基础知识后，结合论文，搭建环境，进行整个模型的可行性分析和测试结果的准确性分析。若模型的可行性出现问题，缺少相关的文件或者参数，那么应该尝试寻找相应文件或是更换模型。这一问题普遍存在，虽然所选的项目模型是开源、可供下载的，但可能存在只提供最后的测试模型，而如何训练模型的部分并没有开源。当完成可行性和准确性分析后，就需要结合论文，详细分析模型的代码。对于一些封装性较好的模型，则更加需要仔细分析，寻找模型输入数据的传输方式，以及神经网络最后输出结果的方式。当输入输出都完成后，就可以开始下一步生成数据集和对接下游应用的工作。

其次是数据集生成问题。在虚拟场景中生成数据集很好地解决图片预处理的问题。通过在虚拟场景中渲染手势来批量生成数据集，与实际场景的数据集生成相比，虚拟场景可以根据模型的特定需求，更加方便地调整渲染背景，避免了其他元素的干扰。同时在虚拟场景中无需实际的摄影设备和场地，也避免了标签数据的标注成本，大大减少了数据集制作的难度。另外，虚拟场景的合成数据集可能同样具有相同的语义信息。如果与真实数据集一同使用，可以大大增加数据集范围，挖掘数据的“困难样本”，提高数据的多样性，从而使得估计的效果更加精确。

最后是实际场景展示问题。通过大量的实验测试调整训练模型的参数，从而使得最后得到一个较好的估计模型。神经网络模型在训练时普遍都是使用2D的数据集进行训练，而实际场景中展示手势关节信息需要3D数据，因此需要将2D的关节数据转换为3D的关节数据。对于项目的实时性，尽可能地提高网络模型的估计效率以及降低与应用传输数据所消耗的时间。

## 2.3 技术选型

本项目涉及深度学习相关方法，结合论文所推荐配置，因此选择Linux操作系统。Linux系统具有更好的稳定性，可以更好的保证运行过程中的稳定性和安全性。其次，Linux系统可以实现更高效的GPU加速，尤其是在多GPU环境下，Linux系统可以更好地利用GPU资源进行并行计算和训练。此外，许多深度学习相关的工具和库都是基于Linux系统开发的，并且Linux环境下的命令行操作也是非常强大和灵活的，这些特性为深度学习的研究和开发提供了更强大的支持。

深度学习框架使用PyTorch框架。PyTorch框架更加适合开发者[8]：第一，PyTorch 使用了动态图框架。动态图框架中计算图在运行过程中构建，因此动态图框架更易调试，具有更高的灵活性。第二，设计简洁。PyTorch在设计上有张量、变量、模块三个抽象层次，设计更为简洁易于理解。第三，PyTorch具有速度优势。第四，极强的易用性。PyTorch框架在目前所有深度学习框架中易于学习，由于采用了命令式编程，其代码与Python等编程语言相似，学习门槛低，且继承了Torch框架灵活易用的接口设计，符合人类的思维方式。

（图）

链接：[(34 封私信 / 66 条消息) PyTorch到底好用在哪里? - 知乎 (zhihu.com)](https://www.zhihu.com/question/65578911/answer/2529374967)

# 3 手部运动数据集设计

## 3.1 环境说明

整体环境框架为Ubuntu 20.04 + Cuda 11.1 + PyTorch 1.8.1。

## 3.2

### 3.2.1 Artiboost模型

ArtiBoost是一种轻量级的在线数据增强学习模型，从数据的角度提高关节手-物体的姿态估计[9]。

从单独一个RGB图像中提取信息来估计关节三维手-物姿态是一个到目前为止都难以解决和具有挑战性的问题。由于模型在训练时需要尽可能地满足数据集的多样性，而手势-物体姿态的信息又比较繁多，这就需要包含不同手姿态、物体姿态和相机视点的大规模数据集。大多数真实世界的数据集缺乏这种多样性，最大原因在于物体的形状并非统一，使用手抓取不同的物体的姿态也是不同的，从而导致真实世界的书记无法包含太多的姿态特征，造成最后训练的模型泛用性较低，欠拟合。相比之下，合成数据集可以很容易地确保巨大的多样性，然后只从它们中学习是低效的，并且训练所需的时间成本非常大。但是先前探索合成数据集的工作已经多次证明了它在增强训练神经网络的真实数据方面的有效性[10][12]。因为合成数据可以很容易地涵盖不同的姿态和视角。这些工作的数据合成阶段与训练阶段分离，称为离线数据增强。然而，并不是每个手-物体结构都有助于训练手部-物体姿态估计模型。例如，类似的结构可能已经被观察过多次，那些容易识别的样本可能经常出现。因此，对于手部-物体姿态估计模型任务来说，在训练过程中不与模型重复交互的离线数据增强被认为是低效的。ArtiBoost将合成数据集与真实世界的数据集结合在一起，使用在线数据增强的方法，通过在训练过程中探索和合成两个步骤，有效地提高关节手-物姿态估计。在训练过程中，ArtiBoost交替执行数据探索和合成，基于合成手-物体配置和视点空间(CCV-space)覆盖各种手-物体姿态和摄像机视点，并通过挖掘策略自适应丰富当前难识别样本[9]。即使是一个简单的神经网络中也可以在HO3D数据集上超越之前基于Transformer[11]模型的效果。

问题定义。给定一个输入图像I∈RH×W×3，提取图像中的信息特征，观察单个手与某个对象交互。手势-物体姿态估计模型旨在学习预测3D手部关节位置的特定神经网络:Ph={pj}J j=1，对象质心位置:po和对象旋转:ro∈so(3) 三个关键信息，其中pj, po∈R3, J =21，H×W为分辨率。

模型框架。为了训练神经网络，首先需要准备一个真实世界的源数据集:Dreal作为初始数据。在训练过程中，ArtiBoost提出了合成手-物体结构和相机视角空间(称为CCV-Space)[9]来模拟手部与物体的交互动作，其中包含了模拟手势姿态的手部空间（C-Space）、模拟手势-物体交互的合成手部空间（CC-Space）以及使用相机记录手势-物体姿态的视角空间（V-Space），并从中采样手-物体-视角姿势三个维度。然后在合成阶段，将采样手和物体在采样视角方向上的多边形网格渲染为具有逼真纹理的RGB图像，形成一个采样权重池M。将采样权重池M与源数据集Dreal以一个权重值Mixture分批混合，得到的混合数据集放入手势-物体姿态估计学习框架中进行训练，以完成前向和后向传播。当训练完成时，ArtiBoost 根据数据挖掘模块更新采样权重池M，将训练损失作为反馈，进入探索步骤，挖掘当前难识别的样本，为下一轮训练做准备。

（框架图）

3.2.1 模型训练与测试

HO3D数据集。Artiboost模型一共有两个可用的开源数据集，分别为HO3D数据集和DexYCB数据集。两个数据集都可以用来进行模型训练和测试，但是在使用DexYCB数据集分析整个模型代码时，发现其数据集的输入方式较为复杂。如果之后参照DexYCB数据集重构代码可能会比较繁琐，不利于之后数据集的替换，因此选择HO3D数据集作为整个模型框架的输入数据。HO3D数据集中的图像包含了手以及与物体交互的动作，因此在下载HO3D数据集本身的同时，还需要下载模型预处理好的物体相关信息。在HO3D数据集中，包含了分别用来训练和估计的数据以及用来控制输入数据集的txt文件。

（图）

模型训练。模型开放了其训练整个Artiboost数据增强部分的代码，但是并没有给出训练的相应初始参数配置信息，导致Artiboost模型部分的训练无法进行，只能够使用一个基于Resnet[3]已经预训练的基础神经网络进行测试，即模型的可行性存在问题。而如果忽略Artiboost提出的CCV-Space方法，只是使用神经网络模型进行测试，可能最后会出现估计的误差值会非常大，无法用于最后在实际场景实时展示手势。为了解决最后的场景应用问题，在分析完Artiboost模型框架后，重新更换了神经网络模型，寻找可以使用自身数据集进行训练和测试的模型设计。

（图）

模型测试。虽然模型的训练部分无法完成，但是整个项目提供了完整的测试流程。通过使用特定的开源数据集，可以使用其已经预训练好的神经网络模型验证数据估计的准确性。在Epoch 0中，通过直接将测试数据集导入网络模型，最后估计得到每张图像中手势节点信息，并且将得到的信息连同物体的节点信息一共渲染到图像中展示。

（图）

由此得到，模型测试部分可以正常运行，绘制的结果也基本正确，但由于此时的网络并没有训练，所以部分关节节点的误差值较大。

3.2.2 输入部分

HO3D数据集的输入参数包含了相机内参、手势边界框、根节点坐标变换值、关节3d坐标、物体坐标信息、物体旋转坐标值、图片像素信息等信息。其中有许多参数是之后数据集生成时所不需要的。例如：有关物体的所有参数信息都是非必要的。因为这些物体参数表示了图像中手部与之交互的特定物体，而在最后的手部姿态实时展示中，并没有特定的物体，那么输入物体的参数信息也就没有意义。剩下的参数信息主要为相机参数、手势边界框、根节点坐标变换值、关节3D坐标初始值以及图像像素信息。

相机参数：对于真实世界的数据集，每个摄像设备都有固定的相机内参；而使用合成数据集，引擎所使用的摄像机一般也都有对应的参数可供调用。

手势边界框：图像原先的分辨率为640 ×480像素。一般整个图像除了手和物体以外，还会包含其他的干扰因素，通过事先缩小需要学习的像素范围，可以大大减少其他物体的干扰。但也正是这个原因，对于前期数据集处理的要求较高，无法做到任意一个相同大小的图像无需处理就可以通过模型训练，造成数据集生成和替换难度的提升。

（图）

根节点坐标变换：手势可以在空间内自由变换旋转，给定根节点的变换信息可以使整个模型的训练和估计得到一个基准值，其他节点根据根节点的位置来估计相应的坐标。

关节3D坐标初始值：一共具有21个关节节点数据，在输入传入时所有关节都是默认为根节点坐标值，当使用模型估计后，才能得到真正估计得出的关节数据。

图像像素信息：包含了图像中每一个像素的RGB三元组信息。

3.2.3 输出部分

原先模型的输出只有一个已经封装好的测试效果图展示，而其中模型估计得到的节点信息并没有暴露对应的接口和函数供他人调用。寻找输出的部分花费了大量的时间和精力，原因在于首先需要对整个Artiboost模型代码有较为清楚的认识，知道每一个模块、每一个部分的的作用。其次需要对Python语言有较为详细的了解，特别是与其他主流语言有所不同的却又频繁使用的功能。经过大量的学习和研究，以及指导老师的帮助，最终寻找到了保存21个关节节点的数据，并将最终估计得出的数据通过Json的格式保存到了项目所在目录的文件中。在最终用来实时展示的Unity游戏引擎中编写调用文件读入的方法，将其中的Json格式的关节数据对应到虚拟场景中制作的手势中，查看估计得出的3D坐标的效果。

（图）

（介绍图片中的信息）

### 3.2.2 TriHorn-Net模型

之前在Artiboost模型训练部分提到，缺少相应的训练参数配置，导致无法进行模型训练。为了解决这一棘手的问题，重新寻找了一个能够正常训练和测试且相对较为轻量的神经网络模型：TriHorn-Net模型。

TriHorn-Net模型使一种能够提高深度图像手部姿态估计精度的新型神经网络模型[13]。

与Artiboost[9]模型相同的是，两者都是为了提高手势姿态精度的模型，都使用了数据增强来更精确地估计手势。数据增强目的是通过随机生成新数据来增加扩充训练数据的数量和多样性以及使数据样本接近实际数据。近年来，在数据增强方法方面取得了重大进展，主要用于视觉[14][15]、自然语言处理[16]和语音[17]。

但是不同的地方在于，Artiboost模型的侧重点在于手部-物体交互的姿态估计，通过实际场景的数据集与合成数据集结合的方式作为输入数据集，不断扩充原有的数据样本，达到数据增强的目标，输入信息也为通常的RGB三元组，而TriHorn-Net模型则是重点关注单独手部的姿态估计，并没有涉及到物体以及其他场景元素，输入信息是手势的深度图，除了手以外图像中没有其他任何干扰因素。通过将深度图拆分成包含关键信息和不包含关键信息但是可能存在潜在信息的注意力图，以此来构建特征池。TriHorn-Net模型第一个创新是将三维手姿估计分解转化为深度图像空间中二维关节位置的估计，并在两个互补的注意力图的帮助下估计其相应的深度。这种分解可以防止深度估计在预测和特征提取上受到干扰，而深度估计同样也是一项较为困难的任务。第二个创新是PixDropout，这是第一个基于外观的数据增强手部深度图像的方法[13]。PixDropout的原理为在深度图上先随机采样，然后屏蔽输入图像中一个矩形区域，用来模拟图像识别任务的遮挡。

问题定义。三维手部姿态估计的任务定义如下:给定一个输入深度图像DI∈RH×W，任务是在相机坐标系中估计一组预定义的手部关节P∈R J×3的三维位置。H, W表示输入深度图像的高度和宽度。J表示待估计的关节总数[13]。

模型框架。TriHorn-Net模型架构包括两个阶段部分。在第一阶段，输入深度图像通过编码器网络f运行。编码器提取并组合手部的低级和高级特征，并输出高分辨率特征体，该特征体传递给三个独立的分支。UV分支计算每个关节的注意力图，这些图集中关注在相应关节发生的像素上。这种过程是通过将注意力图传递到一个特殊的softmax层，再由2D带监督的应用程序计算得到的。第二个分支，称为注意力增强分支，同样也是计算每个关节的注意力图，但没有任何约束，允许它自由学习检测对于不同场景下关节深度值最重要的手部像素。该注意图通过融合操作增强了由UV分支计算的注意图。融合操作由每个关节可学习参数控制的线性插值执行。因此，融合注意图不仅关注关节像素，还关注非关节的手部像素，这些像素包含了估计关节深度值的潜在有用信息。然后使用融合后的注意力图作为指导，从深度分支计算的深度特征图中构建特征池。最后，利用权重共享线性层从每个关节计算的特征向量中估计关节深度值。

（图）

3.3.1 模型训练与测试

MSRA数据集。TriHorn-Net模型一共有三个可用的开源数据集。MSRA数据集相比其他两个数据集结构更加清晰。ICVL数据集在输入方面使用的并非普通的.txt文件，而是使用已经封装好的.pickle文件，在之后数据集替换时可能会出现一些问题，所以最后采用MSRA数据集。MSRA数据集中包括了多个包含了深度图二进制信息的文件和参考图像信息，以及用来控制训练数据大小和测试数据大小的文件，结构简单，易于之后的数据集生成工作。

（图）

模型训练。相比与之前的Artiboost模型，TriHorn-Net模型的训练部分可以正常运行，其代码也更加的简洁，没有做大量的模块封装。在训练过程中，可以看到每一个Epoch的训练情况，包括训练的损失值，从而能够清楚地看到整个模型的收敛效果。

（图）

最后也会生成对应每一个Epoch的训练结果，即网络模型和参数信息。

（图）

模型测试。模型测试使根据之前训练好的神经网络模型进行测试。首先需要选择某个Epoch的网络训练结果，再添加测试数据集，估计得到最后2D关节点的UVD坐标值。

3.3.2 输入部分

MSRA数据集输入参数只有深度图的图像像素信息，没有其他任何需要额外添加的参数。这有助于之后数据集生成的工作，模型默认使用的是图像已经封装好的二进制像素信息，考虑到应用中最后需要输入的是图像信息，所以需要添加将图像转化为对应像素矩阵的功能。另外，因为2D关节信息转换为3D关节信息还需要相机的参数信息，而在模型中这一部分信息是作为变量直接在代码中保存的，因此之后需要根据数据集生成的相机参数信息进行调整。

（图）

3.3.3 输出部分

TriHorn-Net模型的输出参数同样也非常简单，默认只有估计得出的2D关节信息和估计误差值分析。但是在模型中已经保存了有关相机参数的信息，因此可以将2D关节信息转换为3D关节信息，然后同样使用json格式保存到本地，便于之后Unity端数据的读取。

3.3.4 模型封装与优化

当模型的训练、测试、输入部分和输出部分全部完成后，就需要将整个TriHorn-Net模型封装。首先删除了原先的通过bash命令调用方式，而是采用命令行或PyCharm编译器直接运行的方式，使得整个模型能够常驻在后台，等待应用软件的调用。其次重新编写了模型估计部分的代码，删除了原先对不同数据集的操作以及对估计得到的误差值的统计和分析的内容。添加了线程同步的功能使得模型可以在后台挂载，等待调用。

（图）

为了保证实际应用场景的实时性，理论上应达到每秒30帧才能保证人眼视觉上不会发现明显的卡顿现象。但是这也就对网络模型估计的效率要求非常高。经过在模型封装期间的大量测试，发现主要的时间占用在对输入数据的处理阶段。

（数据处理时间 图）

（模型估计时间 图）

原因一是模型本身除了输入需要估计的图像信息外，还需要输入对应图像的标签数据用来计算误差率。而对于误差率的计算是非必要的，所以需要删减对标签数据的读入和计算的模块。原因二是模型的输入图像为320×240像素，但是实际网络模型需要的图像为128×128像素，所以需要对输出数据进行一定的图像裁剪与缩放，需要大量的计算。

（图）

### 3.2.3 模型对比

### 3.2.4 模型训练和测试

## 3.3 数据输入与输出

### 3.3.1 数据集输入

Artiboost模型

TriHorn-Net模型

### 3.3.2 关键数据输出

Artiboost模型

TriHorn-Net模型

## 3.4 模型封装与优化

# 4 应用软件设计

## 4.1 软件架构

应用模块一共分为三个部分：数据集模块、通信模块以及摄像画面截取模块。数据集模块主要解决了模型所需要的训练数据集和测试数据集的问题；通讯模块主要解决了模型与Unity应用数据传输的问题；摄像画面截取模型主要用来对比真实数据集与合成数据集放入模型估计的效果的区别。

## 4.2 数据集生成

虚拟场景生成数据集。若使用默认的数据集训练TriHorn-Net模型，再用自己真实拍摄得到的手部作为数据集进行估计时，误差非常大。这其中最大的原因是因为TriHorn-Net模型的输入数据是深度图，图像中除了手部以外没有其他任何环境因素的干扰，而使用摄像机拍摄的图像很难不受到外界因素的影响，并且手部像素的灰度值也较低。如果使用灯光的话势必也会造成手部周围像素的灰度值也会较高，同样也会影响模型的训练。因此必须在虚拟场景中制作数据集。使用Unity游戏引擎自身的摄像机可以将整个摄像机画面背景从一般的天空盒调整为深度图，符合模型输入的所有要求。即使手部的灰度值较低，也可以添加点光源，在不影响其他区域的同时增加手部的灰度值。

（现实图）

（虚拟图）

但是当把测试数据集替换为虚拟场景中的手部渲染的数据，再此交由模型进行估计后，手势节点依旧不理想。经过分析，可能是模型默认的数据集与自身生成的数据集两者本身差别就比较大，从而造成原先模型所训练的结果无法应用到不同的数据集中。为了能够使得最后的结果基本可以拟合手势姿态，弃用了原本的训练数据集，而是改成自己制作训练数据集。

首先在Unity引擎中选择了一个可供自定义节点信息的手势资产。

（图）

然后制作了一些比较简单的手势姿态

（图）

再对应制作一些驱动手势的动画

（图）

最后编写摄像机脚本。当开始执行时，设置摄像机截取每一帧固定大小的图像信息，作为训练数据集和测试数据集。

（图）

## 4.3 Socket通信与优化

### 4.3.1 Socket通信设计

在之前的模型设计阶段，有提到模型的输出数据都是使用json格式保存到本地的文件中，再由Unity端读取文件里的内容。此方法虽然十分简单易懂，但同时会造成时效性的大幅降低。每一次估计得出的数据，都需要先保存，然后Unity通过循环扫描文件的方式读取内容。为了提升传输效率，选择使用基于Tcp协议的Socket通信。

Python端通信。Python端封装了整个网络模型，应为整个项目的服务端，始终保持运行的状态，直到有Socket连接后才开始估计测试数据集，并将估计得出的结果再发回客户端。

因此在第一次运行网络模型后，应创建一个线程用来单独用作Socket通信。

（图）

而在线程中，首先需要与客户端约定好套接字，即ip和端口号，然后开始监听，等待客户端的连接。

（图）

建立一个队列池。当接收到客户端的连接后，开始实时运行网络模型估计测试数据集，将估计得到的节点信息放入队列池中，而通信线程则循环访问队列，将队列中的数据按比特流的方式传输到Unity客户端。直到模型估计完成且队列池中已经没有需要传输的数据后才结束通信。

（图）

Unity端通信。Unity客户端同样需要新创建一个线程，使用与服务端一样的ip和端口号来请求与服务端的连接。当连接成功时就可以向服务端发送数据或是接收来自服务端的数据，所有的数据都是通过比特流的形式传输。

（发送图）（接收图）

为了防止出现传输的内容丢失，同样需要建立一个队列池，将通信线程接收到的数据保存到队列中，而主线程则不断获取队列中的数据来实时显示手势节点。

（图）

### 4.3.2 Socket通信优化。

Python端

Unity 端

虽然通信的功能已经全部实现，但是在测试过程中发现整个通信的实时性非常低，甚至只能达到每秒传输一次节点数据的速度。原因在于Python服务端创建的通信线程会与主线程抢占CPU时间，通信线程会不断循环等待主线程完成网络模型的数据估计，直到队列中获取到数据才会执行通信，然后继续循环等待。Unity客户端也是如此，在创建了通信线程后，控制手势节点的线程需要不断循环等待，直到通信线程接收到数据并保存至队列中，才能显示当前帧的手势。由此大大影响了整个流程的效率。为此分别在Python服务端和Unity客户端引入了线程同步和事件系统的功能。在Python服务端添加线程池，在同一时间只能有一个线程获得CPU使用权，其他线程被阻塞。使得在通信期间，主线程与通信线程能够交替运行，互不干扰。在Unity客户端添加事件池，实现只有通信线程接收并处理好数据后才会告知手势节点的线程运行，更新当前帧的手势姿态。

（Python图）（Unity图）

### 4.4 相机画面截取

原先设想使用真实的手势作为测试数据集，所以增加了调用笔记本自带的摄像头的功能模块。只要设置好调用的设备、图像高度和宽度，就可以实时调用摄像头的画面，并直接生成为图像的像素矩阵作为输入数据集。同时采集画面模块也是通过新增一个线程实现的，所以也要做与主线程的同步关系。但是在测试的过程中发现，由于TriHorn-Net模型对于数据集的要求较高，真实数据集没有经过预处理无法通过模型正确估计，因而测试数据集主要都替换为了合成数据集，相机画面截取的少量数据集用于之后的数据集对比实验。

# 5 结果展示与分析

## 5.1 模型训练与估计结果

### 5.1.1 Artiboost模型。

Artiboost模型估计结果。Artiboost模型无法训练，因此只进行了测试部分的实验。通过实验说明

### 5.1.2 TriHorn-Net模型

TriHorn-Net模型训练结果。

（Loss折线图）

TriHorn-Net模型估计结果。

## 5.2 数据集对比实验结果

### 5.2.1 模型默认数据集

### 5.2.2 70%源数据集和30%合成数据集

### 5.2.3 50%源数据集和50%合成数据集

### 5.2.4 100%合成数据集

# 6 总结

## 6.1 总结

## 6.2 趋势展望

参考文献

1. Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.
2. Deng J, Dong W, Socher R, et al. Imagenet: A large-scale hierarchical image database[C]//2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009: 248-255.
3. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.
4. Yang S, Liu J, Lu S, et al. Collaborative learning of gesture recognition and 3D hand pose estimation with multi-order feature analysis[C]//Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16. Springer International Publishing, 2020: 769-786.
5. Armagan A, Garcia-Hernando G, Baek S, et al. Measuring generalisation to unseen viewpoints, articulations, shapes and objects for 3D hand pose estimation under hand-object interaction[C]//Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16. Springer International Publishing, 2020: 85-101.
6. Khamis S, Taylor J, Shotton J, et al. Learning an efficient model of hand shape variation from depth images[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 2540-2548.
7. Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.
8. 唐晓彬,沈童.深度学习框架发展综述[J].调研世界,2023,No.355(04):83-88.DOI:10.13778/j.cnki.11-3705/c.2023.04.009.
9. Artiboost
10. Hasson Y, Varol G, Tzionas D, et al. Learning joint reconstruction of hands and manipulated objects[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019: 11807-11816.
11. Transformer
12. Zimmermann C, Ceylan D, Yang J, et al. Freihand: A dataset for markerless capture of hand pose and shape from single rgb images[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 813-822.
13. TriHorn-Net
14. Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.
15. Cubuk E D, Zoph B, Mane D, et al. Autoaugment: Learning augmentation policies from data[J]. arXiv preprint arXiv:1805.09501, 2018.
16. Yu A W, Dohan D, Luong M T, et al. Qanet: Combining local convolution with global self-attention for reading comprehension[J]. arXiv preprint arXiv:1804.09541, 2018.
17. Hannun A, Case C, Casper J, et al. Deep speech: Scaling up end-to-end speech recognition[J]. arXiv preprint arXiv:1412.5567, 2014.