CS473 GPU 计算及深度学习
课程大作业
项目报告

完成者：刁义嘉

目　　录

1. 项目内容与要求

1. 调研 TensorFlow-Graphics， 重点关注语义网络分割这个 tutorial。
2. 将语义网络分割的 tutorial 中的神经网络进行修改。要求：

   （1）加入残差设计一个网络结构；（2）加入 network in network（可参考 GoogLeNet）设计一个网络结构。这两个网络分别在原有基础上进行精度提升，画出网络的结果图，并算出来整个网络的参数。
3. 在人物模型上进一步实现曲面细分。可使用 Geometry Shader，Tesselation 等图形学方法。

2. 项目实验一：修改 mesh segmentation 网络

本实验中，我的主要工作内容是：对 TensorFlow-Graphics 中的 mesh segmentation demo 进行学习，基于课堂中学习的 ResNet 以及 GoogLeNet 网络的残差、Inception 网络的设计 Idea，对 demo 的网络架构进行修改，提升了网络的预测精度。

2.1 残差结构、Inception 的学习与理解

2.1.1 残差结构

残差的结构如下图所示，即相当于将该段神经网络带有可计算权重层的输出与输入相加，作为该段网络的整体输出。“残差”指的就是带有可计算权重层的部分。

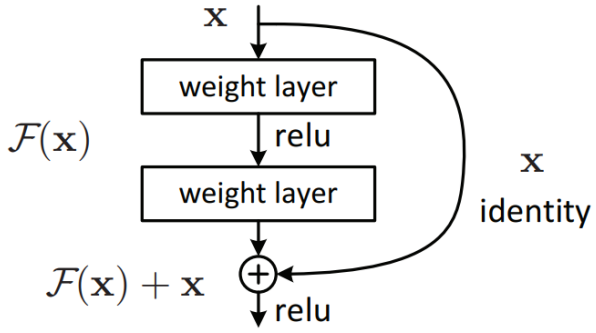


Figure 2. Residual learning: a building block.

残差网络结构的设计理念是解决神经网络在深度加深过程中遇到的模型退化问题。对于不同的任务，我们无法预知多少层深度的网络是足够的；过深的网络会带来梯度消散或爆炸的问题，也会导致整体的训练结果下降。

残差网络能够解决过深网络所带来的问题的基本原因是：在浅层输出已经足够成熟时，深层网络会更多地通过恒等路径进行前向传播，权重层基本不发挥实质性作用，效果会趋向于 0。

残差网络的另一个好处是，能够使反向传播过程对于输出的变化更敏感。通过恒等路径传播的部分会相对减少可训练部分的参数值，使得相对于同等量的输出变化，残差结构更加敏感，梯度下

降更快。

以上提到的两个优点，在**本项目中，较为适用的是：残差网络有利于反向传播**。本网络深度较小，并不很需要解决网络过深引来的模型退化问题，但本项目的网络设计也考虑了该思想。

2.1.2 Inception

Inception 的最关键理念是融合图像中多尺度的特征。不同大小的卷积核的感受野不同，在图像的运算过程中能够获取不同尺度的信息。添加多种尺度的卷积核到网络中，并整合信息，能够使网络感知到更多的图像中多种尺度下的特征信息。

Inception 的实现有许多版本的优化。这些优化的理念来源是，添加多路的卷积之后，训练参数会大幅上升，影响网络性能。通过多层小 Kernel 代替一个大 Kernel、大 Kernel 非对称拆分、用 1*1Kernel 降通道数或 Bottleneck 等技术，能够减少参数的个数，提升训练效率。

本项目主要采用 **Inception 的融合多尺度感受野**的理念进行 Network-in-Network 的修改与设计。

2.2 网络设计结构

示例代码的网络架构图如下图所示：

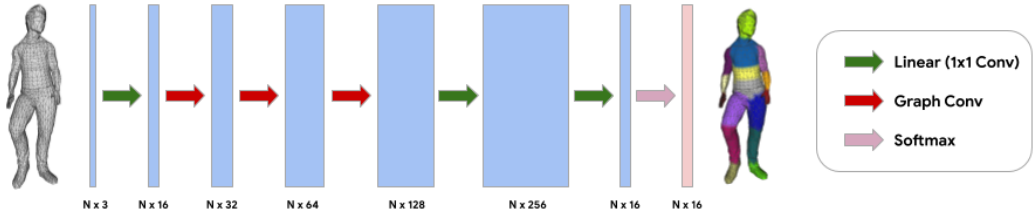


图 1 网络原结构

本部分的网络设计主要基于此网络进行部分修改，该网络整体的结构基本保持不变。

2.2.1 加入残差网络结构

在本部分，我设计了三种加入残差结构的方式。对于所有通道数不相同的残差起止模块，全部使用 Conv1 卷积来进行升通道或降通道的操作。

1. **对整个网络结构添加残差**

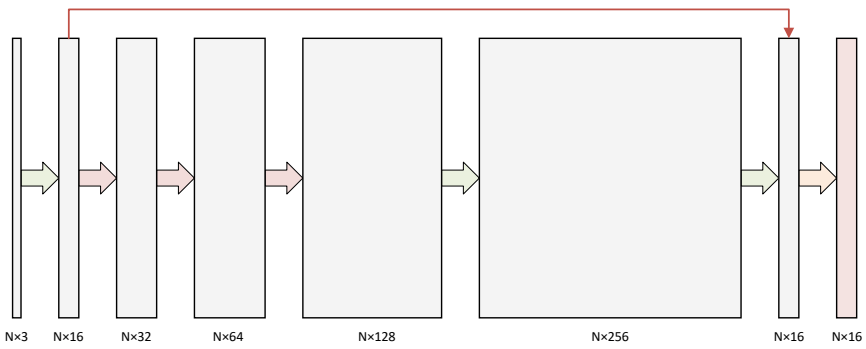


图 2 残差结构 Version0

对于整个网络添加残差的结构图如上图所示，红色箭头即为残差网络结构中恒等路径的传递方向。考虑到网络中有两个 N×16 的层，不用进行额外的提升或降低通道数的操作，因此我选择直接将两个层连接在一起：把第一个 N×16 层与第二个 N×16 层相加形成对于整个网络的残差结构。

2. 对于整个图卷积部分添加残差

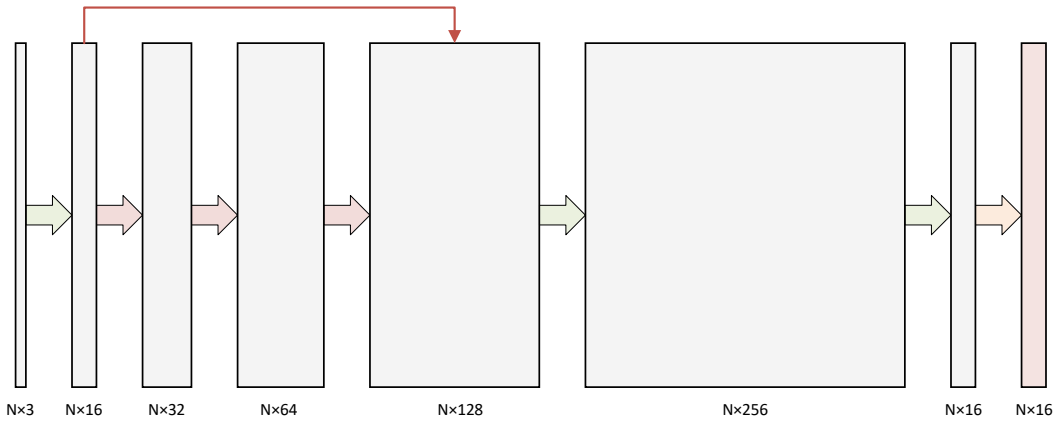


图 3 残差结构 Version1

对于图卷积部分添加残差的结构图如上图所示，红色箭头即为残差网络结构中恒等路径的传递方向。考虑到本网络的重点特征提取部分是图卷积部分，因此对残差结构进行了这个改变。

在此基础上，由于后半部分的卷积层并没有残差结构，于是对该结构进行了改进，对网络的剩余部分也添加了残差结构，如下图所示：

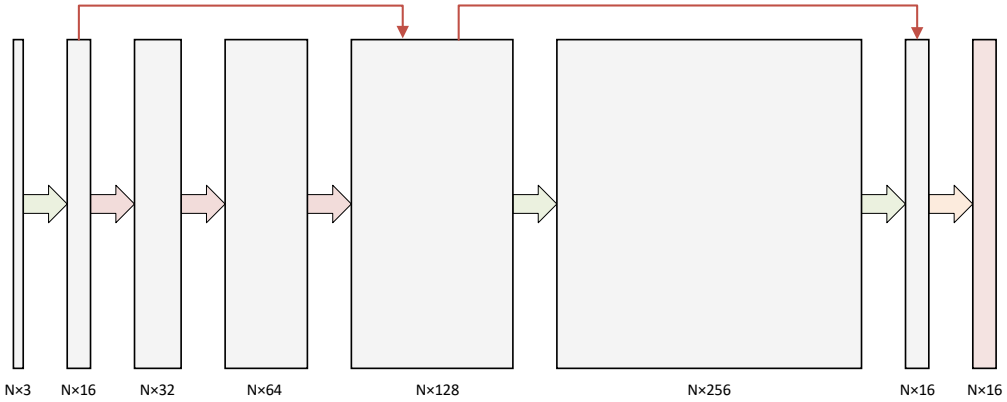


图 4 残差结构 Version1.5

3. 对每个图卷积层添加残差结构

对于图卷积部分添加残差的结构图如下图所示，红色箭头即为残差网络结构中恒等路径的传递方向。该结构的设计考虑是，每一个图卷积操作是否冗余都是未知的，细化残差网络不仅可以加快每个图卷积操作的收敛，更能降低图卷积操作冗余的可能。

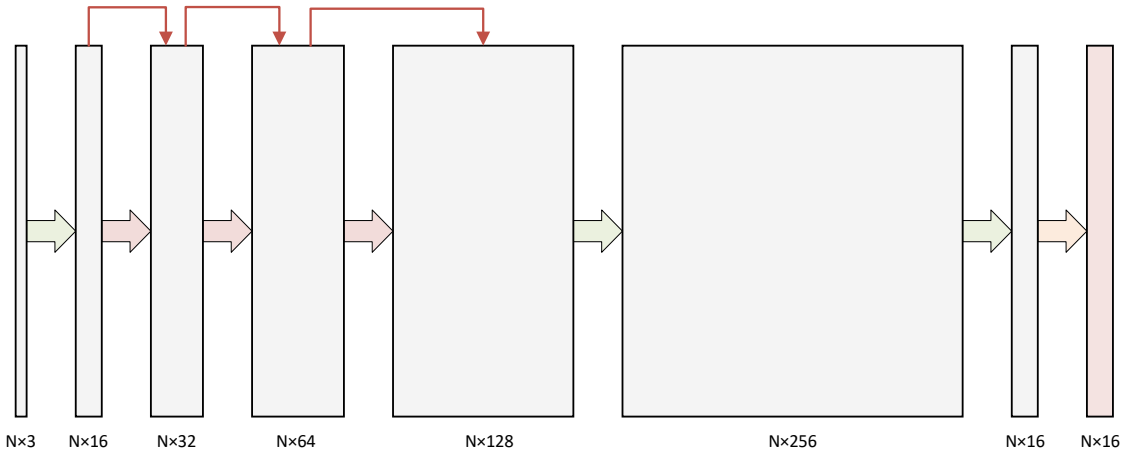


图 5 残差结构 Version2

2.2.2 加入 Network-in-Network 结构

本部分，我设计了两种添加 Network-in-Network 的方式。

1. 非图计算部分的仿 Inception 网络。

对于原网络进行观察，发现非图计算部分的通道数可以有更多种的选择。因此选择设计该网络，整合更多种非图卷积的通道数的大小来汇总图卷积信息。红色框内即为子网络结构。

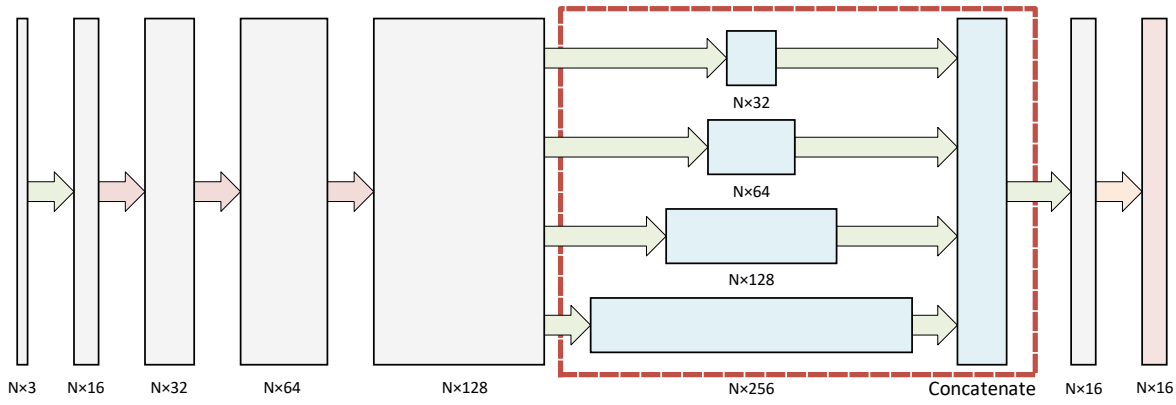


图 6 Inception Version 0

2. 对于图卷积操作的仿 Inception 网络

图卷积部分，实力网络中使用 32-64-128 的通道数顺序。可以在该部分尝试更多的通道数顺序和组合，因此设计了对于图卷积操作的 Inception 网络，如红色框内所示。

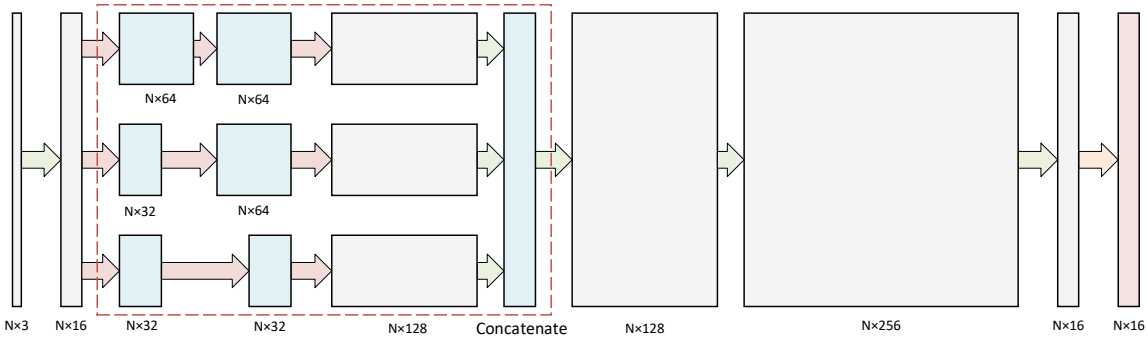


图 7 Inception Version 1

考虑到该网络中存在将三个 128 通道图卷积输出拼接后再进行降通道数的 1×1 卷积操作，将该 Inception 网络简化如下：

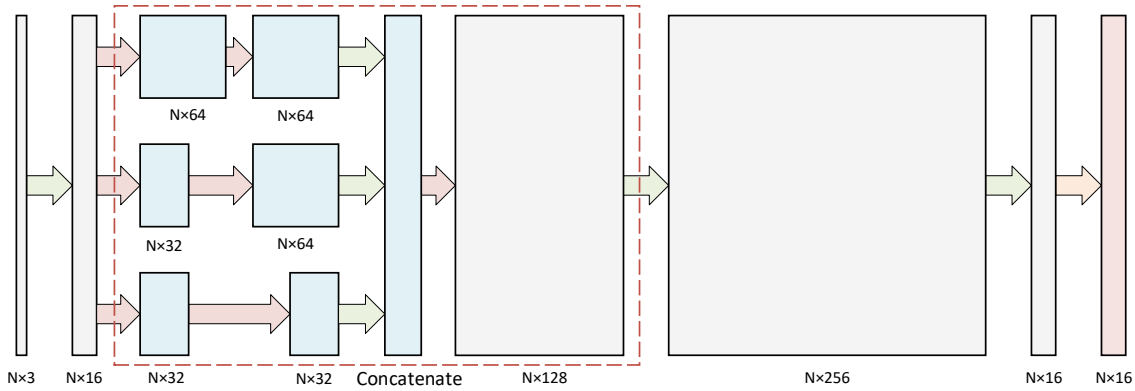


图 8 Inception Version 1.5

2. 2. 3 两种结构结合

两种结构的结合即将 2.2.1 与 2.2.2 部分所设计的网络组合。组合后的网络主要有如下几种：

1. 仅包含图卷积的两种结构结合

第一种情况是将残差结构包在 network-in-network 外面：

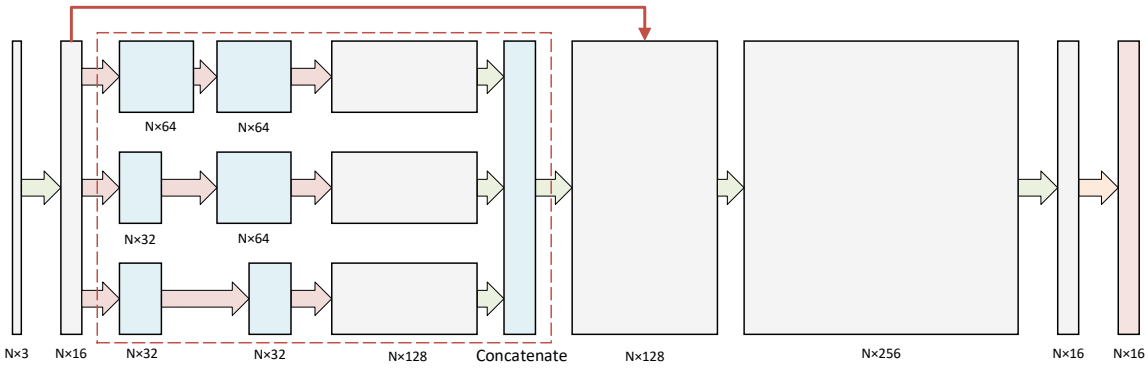


图 9 组合结构 Version 0

第二种情况是将残差结构加入 network-in-network 结构中：

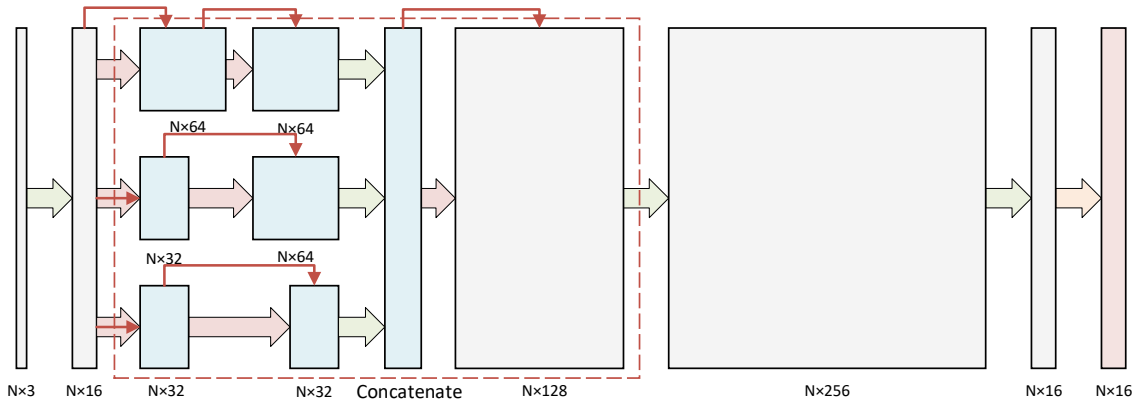


图 10 组合结构 Version 1

2. 整个网络层面的两种结构结合

结构图呈现如下。

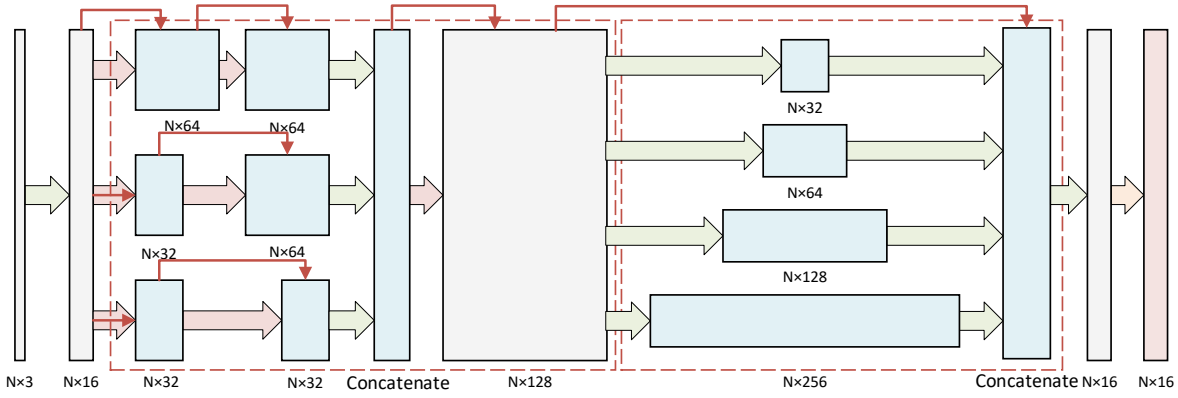


图 11 组合结构 Version 2

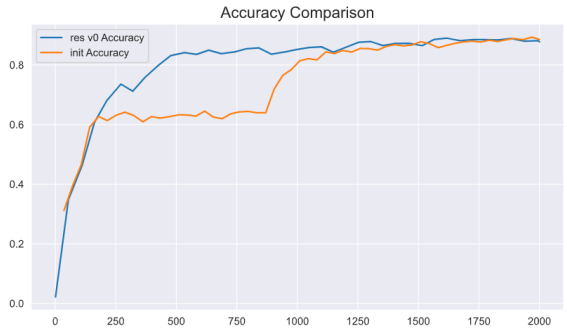
2.3 实验结果与分析

本项目的实验环境是 Google Colab，使用 GPU 运行时。通过摘取训练过程中的 Loss 信息以及测试集上的 Accuracy 信息来进行实验结果的分析。

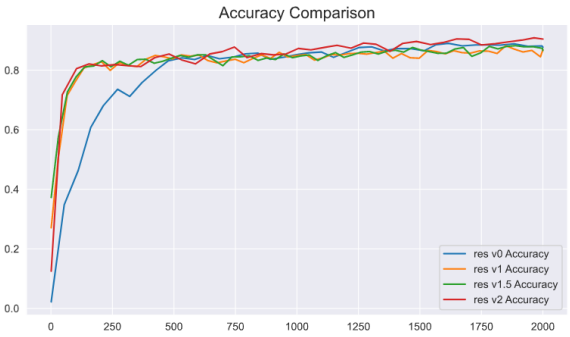
本项目的所有网络版本训练超参数完全相同，并且为节省时间以及更清楚地看到实验对比效果，训练次数都为 2000 次。

2.3.1 残差结构实验与分析

对于图 2 残差结构 Version0 中的网络结构，其 Loss 与 Accuracy 与原始网络对比图如下所示。可以看出，残差结构有效地使 Loss 降得更快，Accuracy 更早达到较好的效果。残差结构的引入较好地实现了 2.1.1 部分所属特性。

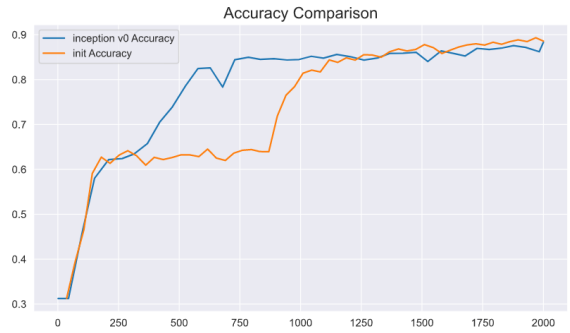


下图对比了三种残差网络设计的训练效果。可以看出，图 3 残差结构 Version1 中的网络结构与图 4 残差结构 Version1.5 中的网路结构的效果相似，都比粗粒度的残差效果更好。对网络训练精度提升效果最好的是图 5 残差结构 Version2 中的网络。说明每个图卷积操作有残差网络的辅助训练效果更好。

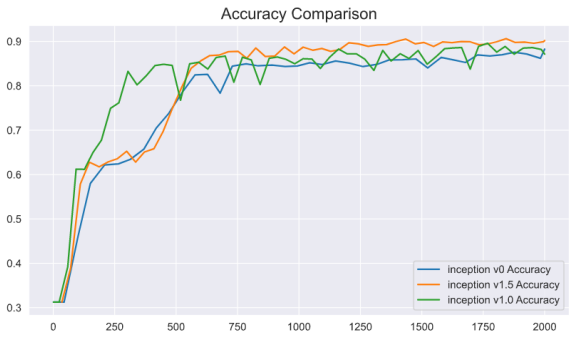


2. 3. 2 Network-in-Network 结构

首先对比非图卷积部分的仿 inception 网络与原始网络的训练情况。可以看出，Inception Version 0 中的网络的训练效果略强于原始网络，但是仅仅在中期收敛更快，精度并未有显著提升。



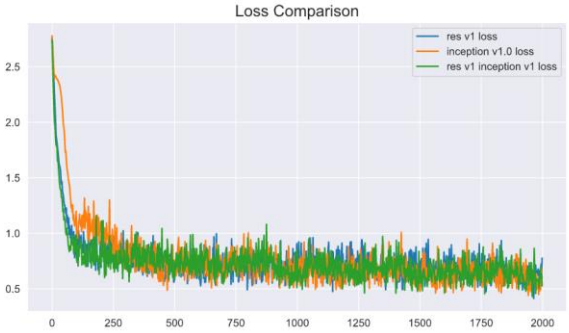
接下来对比三种不同的 network-in-network 结构所带来的效果。如下图所示，Inception Version 1 的收敛速度更快，但是精度提升效果与 Inception Version 0 较为相近。Inception Version 1.5 对网络的精度有一定的提升效果，但是中期收敛速度较慢。此实验结果说明采用 inception 的思想设计的 network-in-network 结构具有一定的加快模型收敛的效果，但提升精度效果很有限。

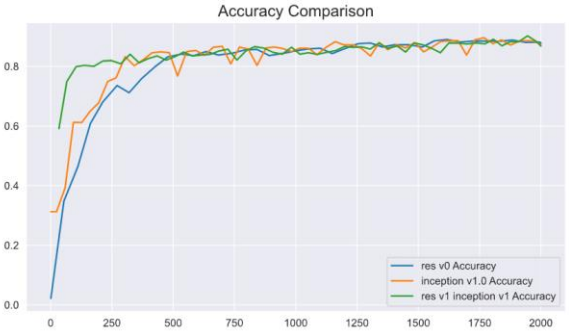


对于 Inception Version 0 非图计算模块添加 Inception 网络效果逊于图计算模块添加 Inception 网络，我的分析是，本项目中的仿 Inception 结构与 GoogLeNet 中的 Inception 结构的作用基本不同。对于非图计算部分，本网络除去通道区别，由于仅仅只有一维，无法做到多层次的信息感知，因此网络收敛情况和精度都逊于图计算 inception 结构。

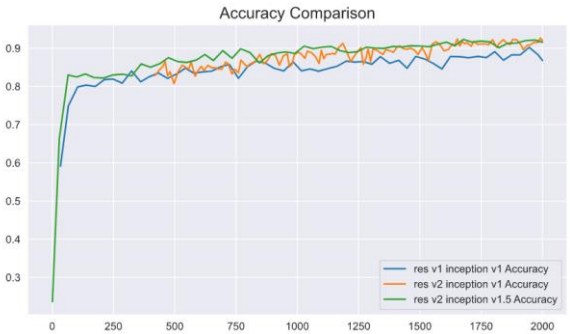
2.3.3 两种结构的结合

首先对比组合结构 Version 0 与被组合的残差结构 Version1 和 Inception Version 1 的模型效果。可以看出，组合结构 Version 0 在网络收敛程度上快于其单独的残差和 network-in-network 结构。



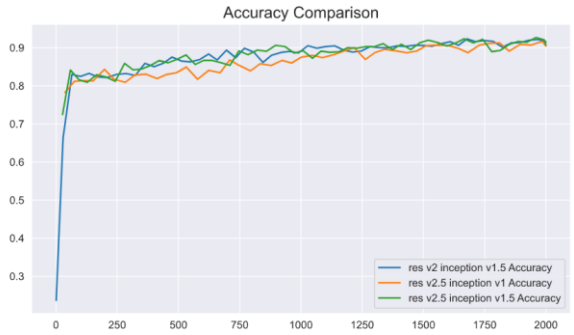


其次对比组合结构 Version 0 与组合结构 Version 1 两种仅在图计算部分整合两种网络结构的模型效果。由下图可以看出，将残差加在每一个图卷积层（res v2）能够使模型更有效，在训练 200 次内就能够将模型准确率提升至 0.9 以上。其中，使用简化版本的 network-in-network 结构 Inception Version 1.5）能够使模型准确率更加稳定,且收敛的相对更快。



最终对比组合结构 Version 2：网络的图计算和非图计算部分都应用了相应网络。可以观察到，三种模型的训练效果相似，其中采用较为复杂的图计算 inception 结构的网络（Inception Version 1）

效果差于简化版的两个网络（Inception Version 1.5）。三者在 2000 次训练之后模型准确率均能达到 0.9 以上，训练最佳结果接近 0.925.



2.4 实验结论

1. 残差网络、基于 inception 理念设计的 network-in-network 都对 mesh segmentation 网络有精度提升、收敛更快的效果。
2. mesh segmentation 网络的精度提升关键部分是图卷积部分，添加图卷积网络层粒度的残差和 inception 网络能够较大程度提升网络精度，在 2000 次训练之后模型预测准确率达到 0.9 以上。

3. 项目实验二：曲面细分

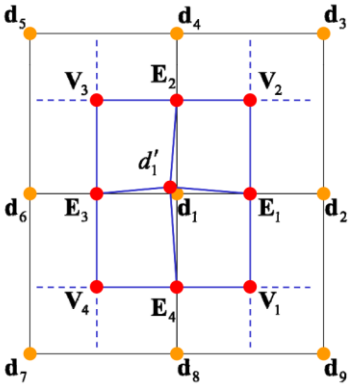
本部分参考网络资料，主要实现了 Catmull-Clark, Loop, DooSabin, Butterfly 的曲面细分算法；并且使用 DooSabin 和 Butterfly 算法，对于 mesh segmentation 中的部分模型进行了曲面细分的实验。

3.1 曲面细分算法

曲面细分算法的原理部分主要参考了 Stanford 的 CS468 课程资料。

3.1.1 Catmull-Clark 算法

该算法主要是对 B 样条的双三次插值的空间扩展，计算流程如下图：



$$\mathbf{V}_2 = \frac{1}{n} \times \sum_{j=1}^{n} \mathbf{d}_j$$

$$\mathbf{E}_i = \frac{1}{4}\left(\mathbf{d}_1 + \mathbf{d}_{2i} + \mathbf{V}_i + \mathbf{V}_{i+1}\right)$$

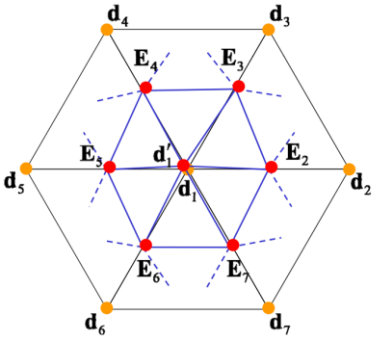
$$\mathbf{d}_1' = \frac{(n-3)}{n}\mathbf{d}_1 + \frac{2}{n}\mathbf{R} + \frac{1}{n}\mathbf{S}$$

$$\mathbf{R} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{E}_i \quad \mathbf{S} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{V}_i$$

图 12 Catmull-Clark 算法

3.1.2 Loop 算法

该算法主要将 Box 样条计算一般化，需要特别注意的是，该算法需要把 mesh 转换为三角网格进行计算。计算关键步骤为：



$$E_i = \frac{3}{8}(d_1 + d_i) + \frac{1}{8}(d_{i-1} + d_{i+1})$$

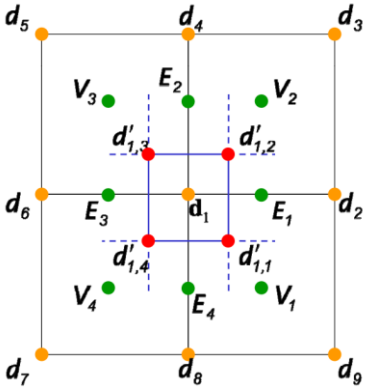
$$\mathbf{d}'_1 = \alpha_n \mathbf{d}_1 + \frac{(1-\alpha_n)}{n}\sum_{j=2}^{n+1}\mathbf{d}_j$$

$$\alpha_n = \frac{3}{8} + \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2$$

图 13 Loop 算法

3.1.3 DooSabin 算法

该算法与 Catmull-Clark 算法类似，都是将对 B 样条的双三次插值的空间扩展。本算法支持对多边形 mesh 进行计算。计算过程如下：



$$V_2 = \frac{1}{n} \times \sum_{j=1}^{n} d_j$$

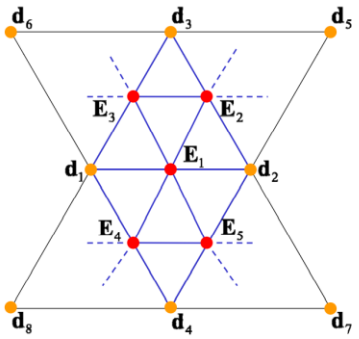
$$E_i = \frac{1}{2}(d_1 + d_{2i})$$

$$d'_{1,j} = \frac{1}{4}(d_1 + E_j + E_{j-1} + V_j)$$

图 14 DooSabin 算法

3.1.4 Butterfly 算法

该算法使用较为原始的插值方法，可以应用于三角 mesh。计算过程如下：



$$\mathbf{E}_1 = \frac{1}{2}(\mathbf{d}_1 + \mathbf{d}_2) + \omega(\mathbf{d}_3 + \mathbf{d}_4) - \frac{\omega}{2}(\mathbf{d}_5 + \mathbf{d}_6 + \mathbf{d}_7 + \mathbf{d}_8)$$

$$\mathbf{d}'_i = \mathbf{d}_i$$

图 15 Butterfly 算法

3.2 代码实现

曲面细分的代码实现主要分为两部分。一部分是，将 mesh segmentation demo 中的人体模型转换为.off 格式；另外一部分是实现上述算法，将输入.off 格式的 mesh 进行细分，输出细分后.off 格式的 mesh。

3.2.1 模型格式转换

本部分主要在 mesh segmentation 代码的基础上进行更改；将经过读入处理后的 tfrecords 输出伪.off 格式文件即可。

3.2.2 曲面细分算法实现

本部分主要参考了 ozkanyumsak 的工作。

本部分的算法实现是，从输入.off mesh 文件，进行曲面细分后输出.off mesh 文件。代码实现语言是 C++，使用 Visual Studio 进行编译。

代码的数据结构参考 ozkanyumsak 的工作，建立一个 mesh 数据结构。该数据结构主要组成是自定义的点、线、面三个类；三个类型分别都会存储与自己相关的点线面的信息并且及时更新。

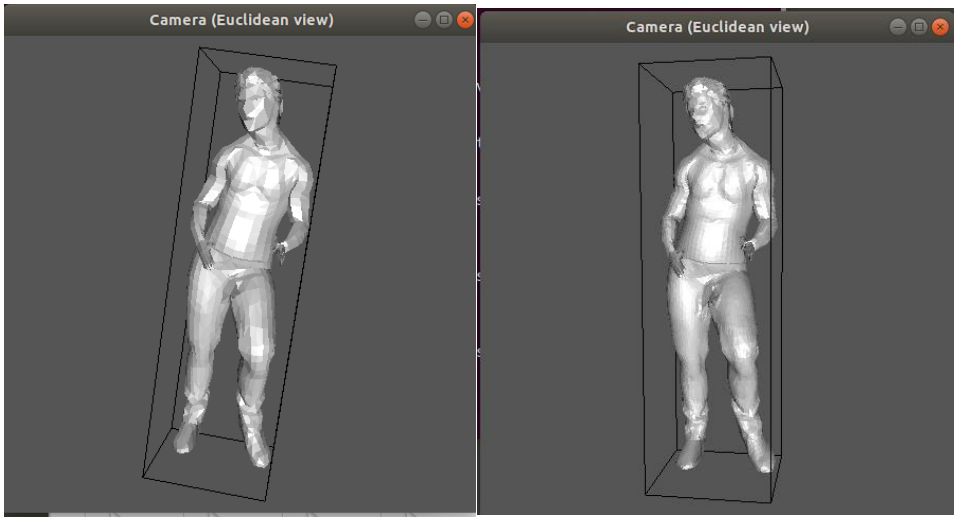
代码的算法的实现分别定义了四个算法类来实现。每一次做曲面细分，都会根据当前数据结构存储数据，按照 3.1 中所述算法进行计算。

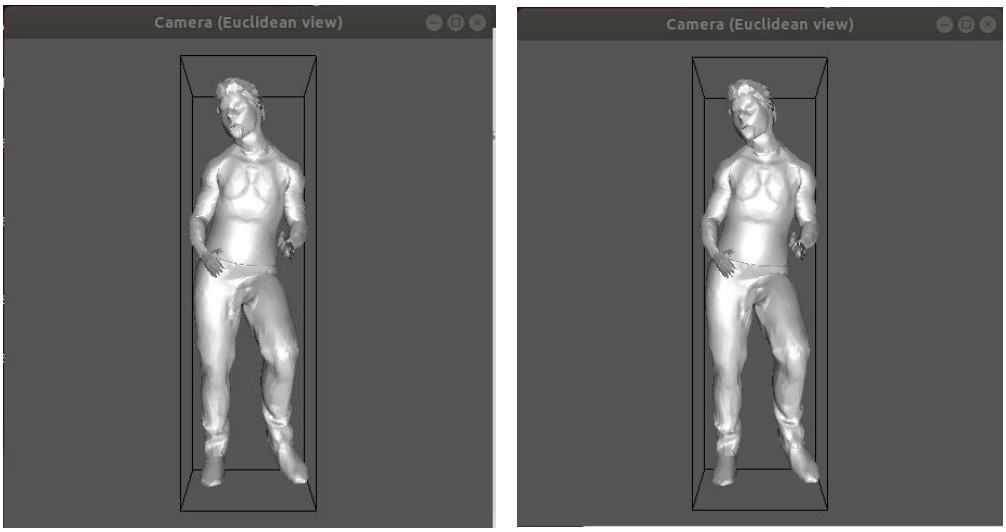
3.3 效果展示

3.3.1 人体曲面细分效果展示

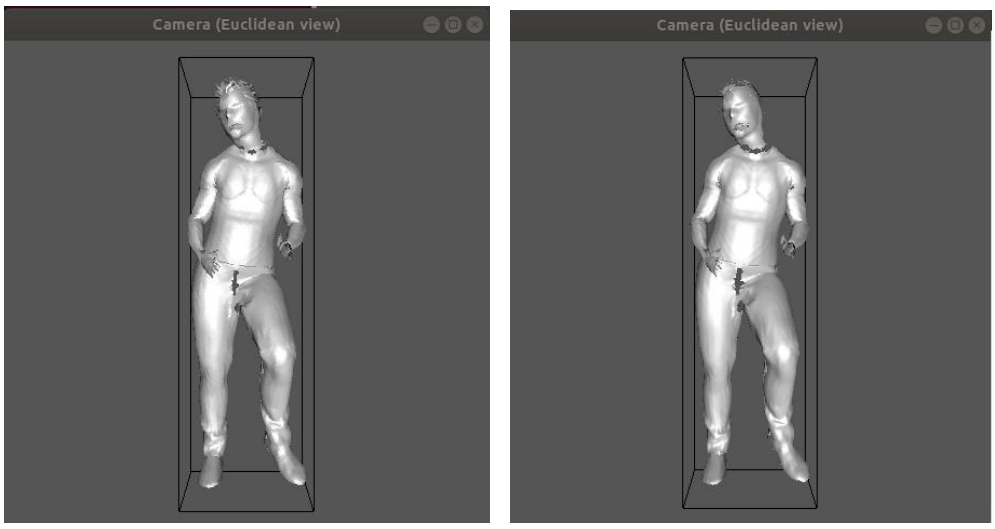
由于 Catmull-Clark 算法不支持三角网格的细分；实验过程中 Loop 算法会产生 inf, nan 的错误值，所以曲面细分仅采用 DooSabin 和 Butterfly 算法进行计算。

以下四张图按照从左至右、从上至下的顺序，分别为原始 mesh、Butterfly 算法迭代 1 次、Butterfly 算法迭代 2 次和 Butterfly 算法迭代 5 次的人体模型：

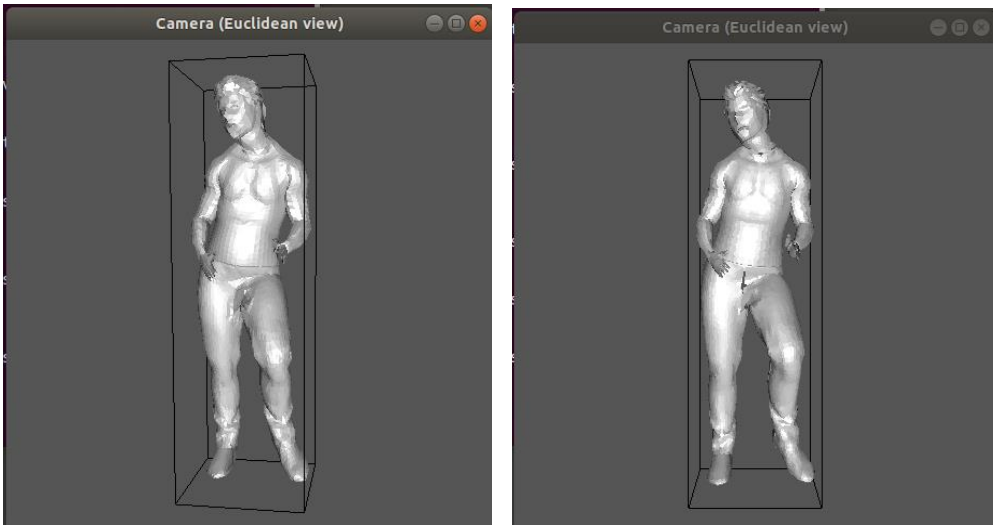




以下两张图，左侧右侧分别是 DooSabin 算法迭代 2 次和 DooSabin 算法迭代 5 次的人体模型。DooSabin 中的部分缺陷是没有完整地绘制出所有 mesh 点和平面。



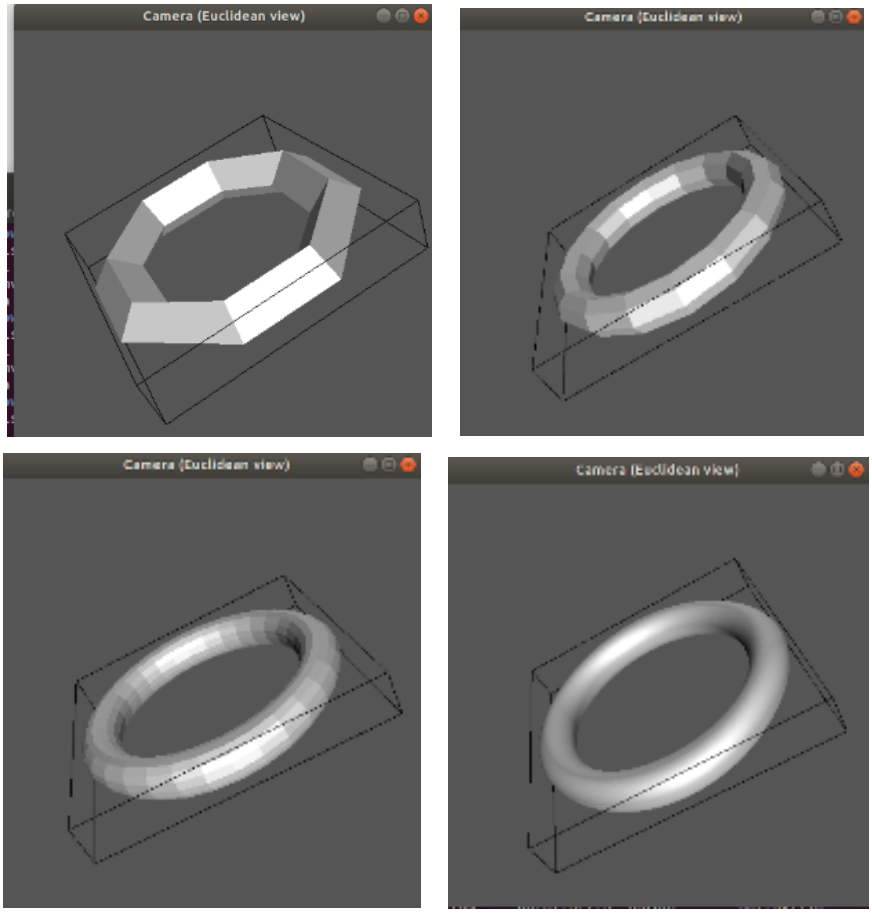
以下两张图分别为 Butterfly 算法迭代 1 次和 DooSabin 算法迭代 1 次的人体模型。可以明显地看出，DooSabin 算法是多边形的曲面细分算法，相同迭代次数达到的细分效果更好。



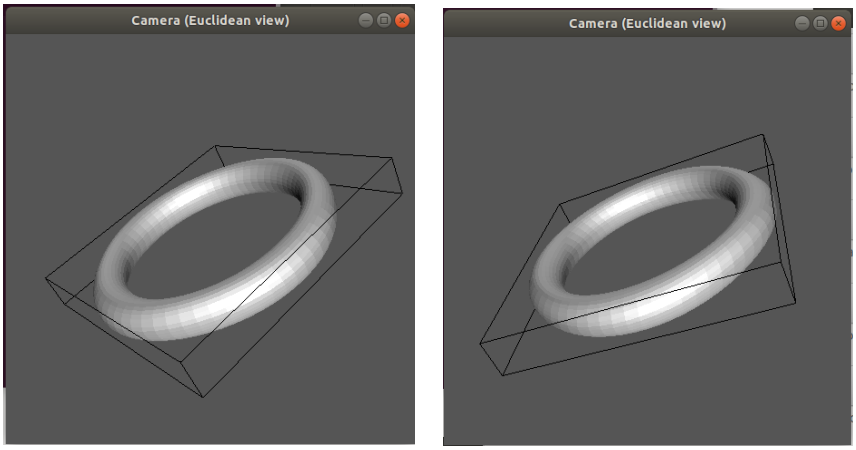
3.3.2 曲面细分算法效果展示

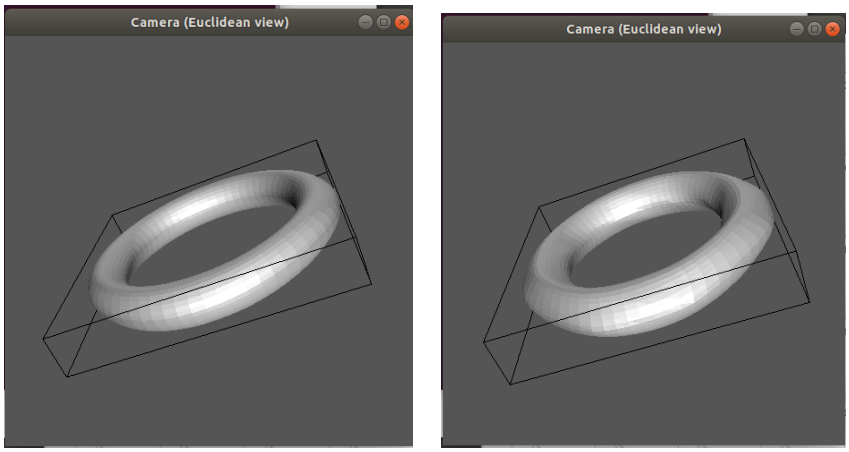
本项目的效果展示使用了 geomview 工具；运行环境为 VMWare 虚拟环境中的 Ubuntu18.04 系统。使用 sudo apt-get install geomview 即可安装；输入 geomview ./{file name}.off 即可可视化.off 文件。

原始环形 mesh 与 Catmull-Clark 算法迭代 1，2，5 次的曲面细分效果对比：



使用 Catmull-Clark, Loop, DooSabin, Butterfly 算法迭代三次的 mesh：





4. 实验过程中遇到的问题与解决方案

Mesh segmentation 网络修改部分主要遇到两个问题。

第一个问题是 tensorflow 中 estimator 的使用问题：如何保存运行过程中的 loss 与 accuracy 信息。我目前的解决方法是，控制 train_and_evaluate 过程中输出 loss 的 accuracy 信息，在训练结束之后，将该文本信息输入自己编写的 python 脚本，获取具体值并且绘制图像以供分析。选择这个并不是很漂亮的解决方法，一是因为 Colab 的使用自由度较低，无法方便的获取运行路径下的数据信息；另外一个是，在尝试对代码运行路径进行更改时，会产生无法找到问题原因所在的报错。由于时间有限，我并未仔细阅读 mesh_segmentation_dataio.py 文件，因此都采用了 notebook 中默认的读写方式。

第二个问题是程序中 tensorflow 中 API 调用不当导致的报错。我之前并未使用过 tensorflow 训练网络，所以对其中 API 不是很熟悉，因此在修改网络过程中遇到不少无法直接查到问题所在的报错。这些问题通过阅读官方文档、仔细阅读示例代码能够解决。

曲面细分部分主要遇到的问题是环境配置。为减少环境配置可能遇到的各种浪费大量时间的问题，我选择参考基本纯 C++实现的项目，并且修改成为以文件为输入输出的格式。在配置好 geomview 环境之后，我遇到了将.off 文件拷贝之后文件格式没有问题但无法运行展示的问题。在经过各种尝试之后，该问题是由于该软件不支持 windows 下生成的文件造成的。

5. 项目总结与心得

本次课程大作业的 mesh segmentation 网络修改部分让我收获很大，首先是通过实践让我对残差网络结构、Inception 的 idea 有了更深入的理解；通过对模型更改的尝试、实验与结果整理，我对于语义分割网络也有了更加清楚的认知。该部分也让我熟悉了 tensorflow 以及 colab 的使用。

本次课程大作业的曲面细分部分让我学习到了基本的曲面细分算法，并且通过阅读、配置并修改算法代码的过程对该算法理解更深刻。

6. 参考资料

1. https://www.tensorflow.org/graphics/api_docs/python/tfg/nn/layer/graph_convolution/feature_steered_convolution_layer
2. https://tensorflow.google.cn/api_docs/python/tf?hl=zh-cn
3. https://tensorflow.google.cn/api_docs/python/tf/estimator?hl=zh-cn
4. Targ, Sasha, Diogo Almeida, and Kevin Lyman. "Resnet in resnet: Generalizing residual architectures." arXiv preprint arXiv:1603.08029 (2016).
5. Sam, Suriani Mohd, et al. "Offline signature verification using deep learning convolutional neural network (CNN) architectures GoogLeNet inception-v1 and inception-v3." Procedia Computer Science 161 (2019): 475-483.
6. https://github.com/PanJinquan/tensorflow_models_learning
7. http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/10_Subdivision.pdf
8. https://github.com/ozkanyumsak/mesh-subdivision
9. http://www.geomview.org/

7. 致谢

感谢老师们精彩的课堂讲授与安排的平时作业辅导。尤其是学期后期的课程与此次大作业内容高度相关，老师简洁明了的让我对 ImageNet 各种网络有了更深入的理解，用生动的例子让我对 Computer Graphics 的工作有了更多的了解和兴趣。与实践高度相关的课程使我收获颇多。

感谢课程的各位助教，帮助我对课程所讲授知识有了更细致的理解，也帮助我解决写作业过程中遇到的各种技术细节问题。