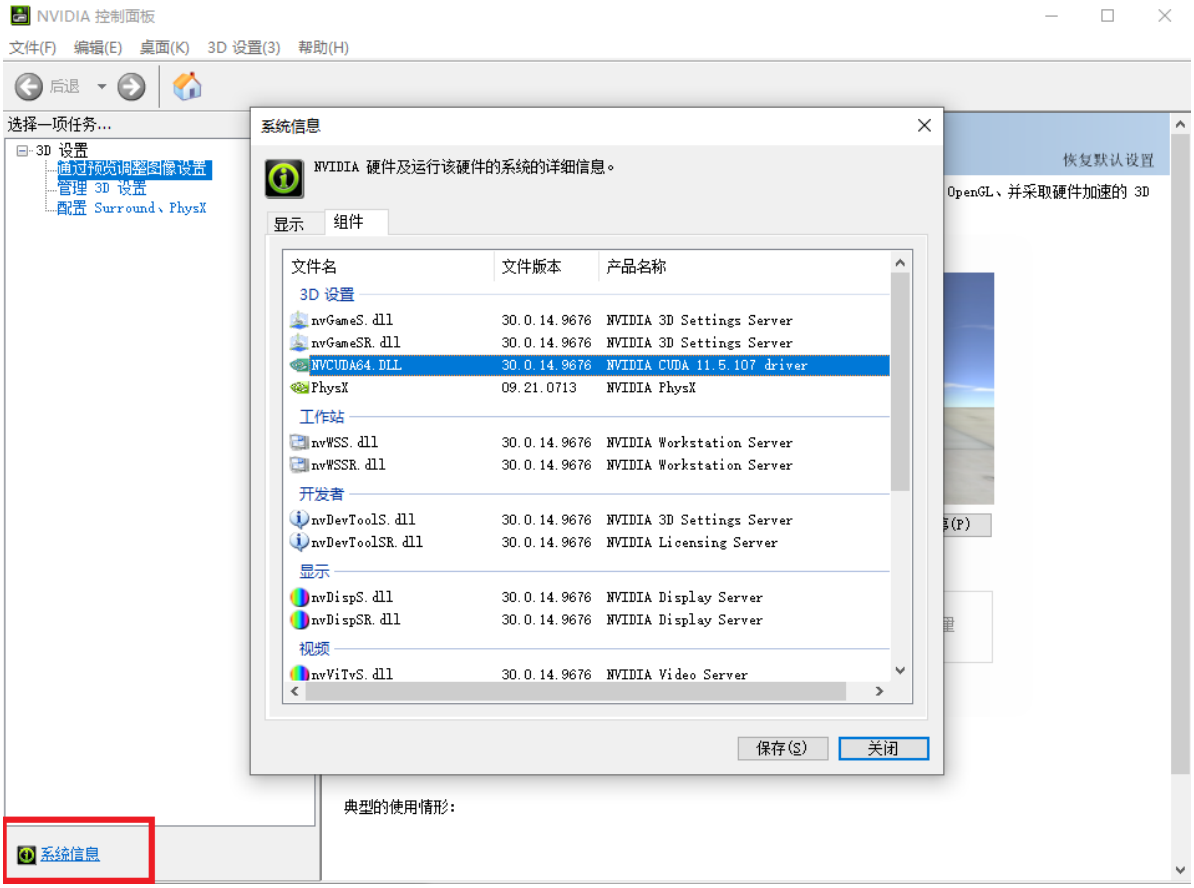
ReDraw-Mockup项目文档

运行要求

环境要求

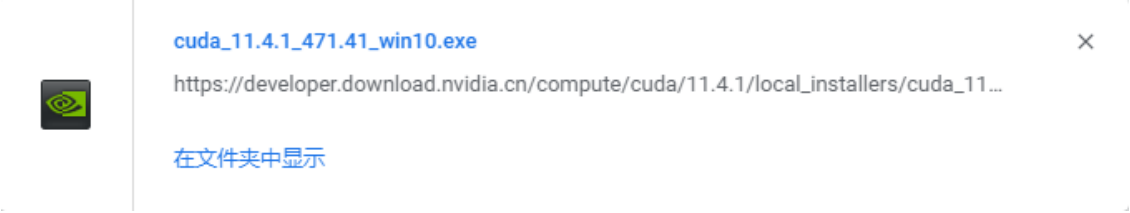
- Python >= 3.7
- NVIDIA CUDA驱动

  在NVIDIA控制面板中查看系统信息，找到组件中NVCUDA64.DLL，这就是NVIDIA CUDA驱动。查看驱动版本，下图为11.5.107



- 安装CUDA

  在CUDA官网下载，下载的时候要注意版本，不能高于NVIDIA CUDA驱动版本，否则可能不适配。



  复现中下载的是11.4.1，然后简单安装即可。安装完成后可以使用命令`nvidia-smi`来验证是否安装成功。

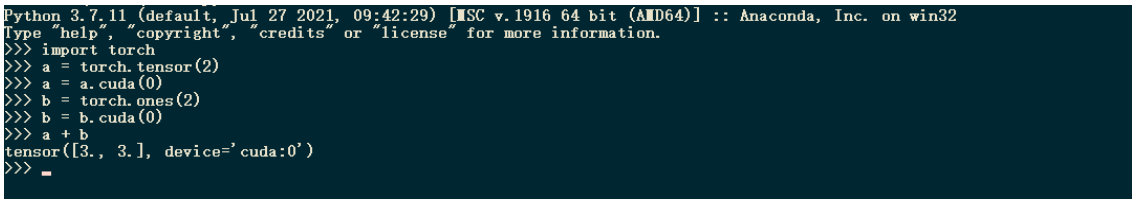
- Pytorch GPU版本

  训练CNN需要使用GPU加速。在Pytorch官网选择安装。

PyTorch Build	Stable (1.10)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 10.2	CUDA 11.3	ROCm 4.2 (beta)	CPU
Run this Command:	pip3 install torch==1.10.0+cu113 torchvision==0.11.1+cu113 torchaudio===0.10.0+cu113 -f https://download.pytorch.org/whl/cu113/torch_stable.html			

复现中选择了使用Pip安装，计算平台选择CUDA 11.3版本，复制下面`Run this Command`中的命令行运行安装即可。

安装完成后进行验证：



可以看到在`cuda:0`(GPU)上可以进行tensor创建和运算，Pytorch GPU版本安装成功。

- 安装必要的库`pip install -U d2l`

其他要求

- 由于训练出来的CNN体积过大，无法上传至Github，因此需要额外下载。
  - 百度云盘链接：https://pan.baidu.com/s/1PGk3_xJ7kklDwNRHNjhZLQ 提取码：zjzh
  - 下载完成后，需要放在项目的`CNNClassifier`目录下。
- 需要下载KNN层次聚类的数据仓库，链接为ReDraw-Final-Google-Play-Dataset.tar.gz，下载完成后，需要解压，并且将项目`KNNAlgo.Aggregate.py`文件中的`MINED_DATA_DIR`改为解压后的数据仓库本机地址。
- 在项目下直接运行，其中`screenfile`为GUI截图的文件路径（最好使用绝对路径）

```
python run.py screenfile
```

功能模块及交互

CNNClassifier

由于ReDraw工具核心算法中的核心是使用了软件数据仓库挖掘以及自动化动态程序分析技术得到的大规模数据集。因此数据集对于复现十分重要。

CNN训练数据下载地址 https://zenodo.org/record/2530277#.YaHyELgzaHs

目录`CNNClassifier`下为CNN训练代码和以及GUI组件分类的接口。`DataLoader.py`负责创建数据集（测试集，验证集以及测试集）。`GNet.py`负责创建CNN以及训练。两个模块都使用了Pytorch框架来实现。`classifier.py`提供了`predict`接口负责预测GUI组件的类别。详细代码实现见项目。

训练CNN

以下是训练CNN前10个epoch的信息。`test acc`是指在验证集上的准确率，用来评估模型的泛化效果。

```
epoch 1==========================================================================================
train loss 1.416, train acc 0.561, test acc 0.635
epoch 2==========================================================================================
train loss 0.838, train acc 0.741, test acc 0.721
model saved in epoch: 2
epoch 3==========================================================================================
train loss 0.677, train acc 0.783, test acc 0.771
epoch 4==========================================================================================
train loss 0.572, train acc 0.814, test acc 0.781
model saved in epoch: 4
epoch 5==========================================================================================
train loss 0.486, train acc 0.840, test acc 0.804
epoch 6==========================================================================================
train loss 0.417, train acc 0.862, test acc 0.819
model saved in epoch: 6
epoch 7==========================================================================================
train loss 0.358, train acc 0.883, test acc 0.825
epoch 8==========================================================================================
train loss 0.318, train acc 0.894, test acc 0.820
model saved in epoch: 8
epoch 9==========================================================================================
train loss 0.275, train acc 0.908, test acc 0.829
epoch 10=========================================================================================
train loss 0.246, train acc 0.918, test acc 0.838
model saved in epoch: 10
model saved in epoch: 10
final: train loss 0.246, train acc 0.918, test acc 0.838
279.0 examples/sec on cuda:0
```

由于设备性能的限制，没有按照文章中的训练数据选取方法。复现选取训练数据时，如果GUI组件种类的图片数量超过1k张，则不使用扩增的数据进行补全，并且令其数量不超过1w张。反之，如果图片数量没有超过1k张，则使用扩增的数据进行补全至1k张。得到52517个GUI组件的训练集。

可能训练数据量不够大以及CNN的能力过强，出现了一定程度的过拟合现象。

GUIDetection

在`GUIDetection`目录下为GUI组件检测的代码实现。`procAppScreenshot.py`下的接口`processScreenshot`处理输入图片检测GUI组件并且返回原子的GUI组件。

在子目录`RectUtils`下，实现了对Canny轮廓检测得到BoundingBox的数据结构，Rect表示一个矩形的BoundingBox，RectView表示这些Rect之间的树层次结构关系。RectUtil为处理Rect代码逻辑的辅助函数。

在子目录`Utils`下，定义了一些通用的辅助类和辅助函数。有图片处理展示的`ImageUtil.py`，颜色相关的`ColorUtil.py`以及定义Canny边缘检测的常量的`Constant.py`。

在子目录`ViewProcess`下，是使用Canny边缘检测得到轮廓的BoundingBox的代码逻辑。`Canny.py`封装了Canny边缘检测算法的具体实现，`ContourAnalysis.py`实现了从`Canny.py`得到的轮廓层次结构中产生BoundingBox的轮廓层次结构的代码逻辑。由于Canny边缘检测算法的精度问题，`HierarchyInfo.py`中的`ViewHierarchyProcessor`对得到BoundingBox的层次结构中的区域做了处理，包括删除重叠的BoundingBox，对轮廓的层次结构做进一步处理等。

KNNAlgo

在`KNNAlgo`目录下为KNN层次聚类算法的具体实现。`Aggregate.py`下的接口`aggregate`实现了对InputNodes的KNN层次性聚类，利用数据仓库中的GUI截图以及运行时层次结构，获得匹配的GUI截图并使用匹配GUI组件的父容器来构造InputNodes的层次结构。`aggregate`返回的是一个RootNode，是整个GUI组件层次的根节点，详细见项目代码。

子目录`Utils`下实现了描述GUI组件和容器的类以及一些辅助函数。最关键的`NodeUtils.py`下的`rectViewsToNodes`接口将`GUIDetection`模块`procAppScreenshot.py`中的接口`processScreenshot`返回的RectView转换成描述GUI层次信息的GUINode。GUINode中的成员如下，定义了描述一个GUI组件或容器的信息：

```
class GUINode:
    def __init__(self, xml_node = None, rect = Rect()):
        self.classType = ""
        self.rect = rect
        self.children = []
        self.dep = 0
        self.parent = None
        self.text = ""
        self.img = None
        self.x, self.y, self.height, self.width = rect.x, rect.y, rect.height,
rect.width
        ......
```

`rectViewsToNodes`接口将RectView转换成GUINode，其中`node.classType = classifier.predict(node_image)`使用了前面`CNNClassifier`中的`predict`接口来预测GUI组件类型。GUINode同时也可以表示数据仓库GUI截图的层次信息（通过解析xml文件），例如不同的Layout，不同的GUI组件类型等。

run.py

`run.py`中的`runApplication`为运行接口，使用整合了前面模块所提供接口函数来运行。使用`processScreenshot`得到GUI组件BoundingBox等信息，使用`rectViewsToNodes`得到组件的种类信息以及GUI截图图像等。最后调用`aggregate`进行层次性聚类得到GUI的层次结构。