综合设计（二）

➢ 题目：

基于无人机的建筑图像识别

➢ 背景：

无人机已经被广泛应用于城市安防、建筑测绘等智慧城市建设中。在城市建设管理和改造中，无人机可以被应用于城市建筑测绘、违章建筑识别等城市管理中。本实验以无人机的城市建筑识别、跟踪、违章识别为应用场景，利用无人机边缘 AI 进行处理，实现统计建筑数量、跟踪建筑变化，对建筑变化的图像感兴趣高清编码回传等功能。

➢ 教学目标：

目标让学生掌握轻量型 AI 平台实现方法，掌握图像目标处理，目标检测方法，感兴趣区域编码方法，掌握 RTMP 推流以及直播整体系统环境搭建方法。

➢ 目的、要求：

1. 深度学习模型转换；
2. 建筑航拍视角图像检测，对所有建筑标注出轮廓（红色）；
3. 对比建筑数据库，对变化的建筑进行放大用蓝色进行标识；
4. 对变化区域的图像进行 ROI 编码；

5. 图像进行 HEVC 编码后，通过 RTMP 传输，接收端通过 VLC 可以看到视频；

6. 同时对编码后的视频流进行本地存储（每 5S 一段视频）；

➢ 实验器材与开发环境：

➢ 开发环境：

开发主机：Ubuntu

Docker 镜像：sophonsdk_v3.0.0_20220716

硬件：算能 SE5

➢ 实验器材：

开发主机 + 云平台（或 SE5 硬件）

➢ 程序文件：

项目由以下文件夹构成：

```
uva_building_proj
├──bin
├──code
├──docs
├──results
├──sources
└──video
```

其中，bin 包含可执行文件、执行脚本、深度学习模型等；code 文件夹包含源代码，results 中包含代码运行的结果，source 包含基于 pytorch 训练的模型、nginx 包，video 文件夹中是测试视频，docs 包含实验的指导文档。

➢ 实验步骤：

2.1 环境搭建

2.1.1 SDK 的 docker 开发环境

- 首先确保已经按基础实验的实验 1 的步骤完成了 docker 开发环境的搭建

2.1.2 SE5 盒子的网络设置

SE5 盒子上有 WAN 和 LAN2 个网口，可以选择任一种方式连接。其中 LAN 网口为固定 IP,其值为 192.168.150.1，使用网线连接 SE5 盒子的 LAN 口与电脑主机的网口，并设置电脑主机网口 ip 为 192.168.150.2/24，即可使电脑 ping 通 SE5 盒子（但 SE5 盒子不可上网）；而 WAN 口可动态 IP 分配，使用网线连接 SE5 盒子的 WAN 与路由器的 LAN 口，且在路由器上启动 DHCP 服务，SE5 盒子即可自动获取 IP.

2.1.3 通过 SSH 连接到 SE5 盒子

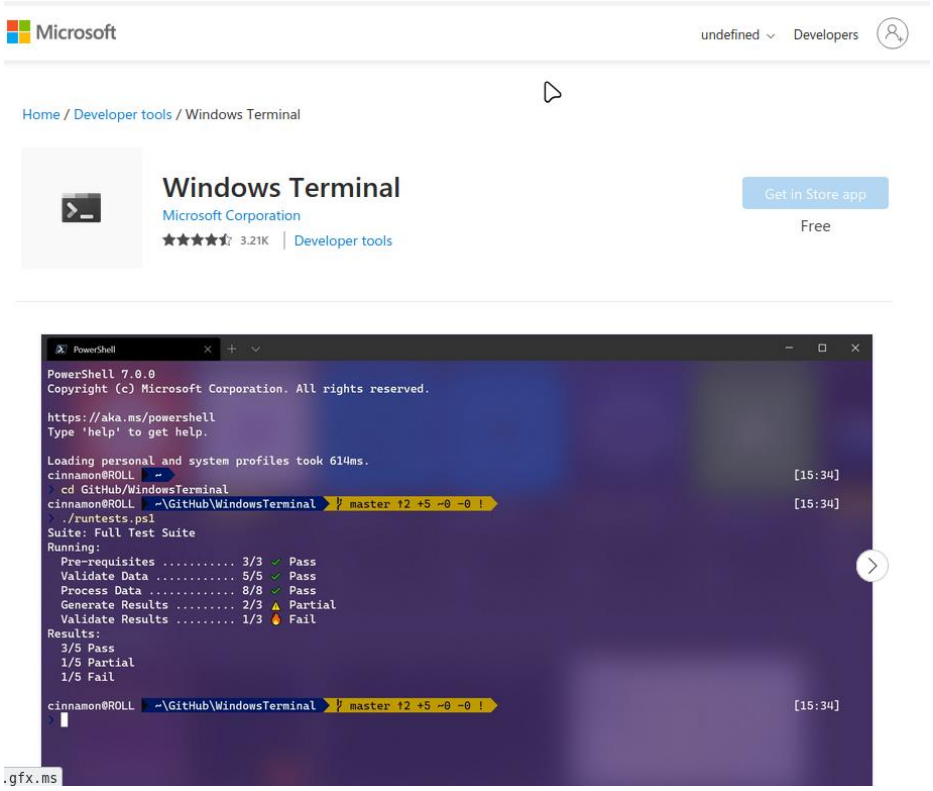
可以通过任何自己喜欢的 ssh 客户端软件连接到 SE5 盒子上，这里使用 ssh 命令连接，当然也可以通过 xshell,finalshell 等软件进行连接。

```
#这里的 ip 地址换成你的 se5 盒子的 ip 地址
ssh linaro@192.168.1.100
```

SE5 盒子默认的账号密码都是 linaro

大部分 linux 发行版都会自带 ssh 命令，在 linux 平台上直接使用上述命令即可，windows 原生的 cmd 命令行没有 ssh 命令，有 2 种解决方法：

1.在 windows 商城中下载更为现代化且命令向 unix 兼容的 Windows Terminal

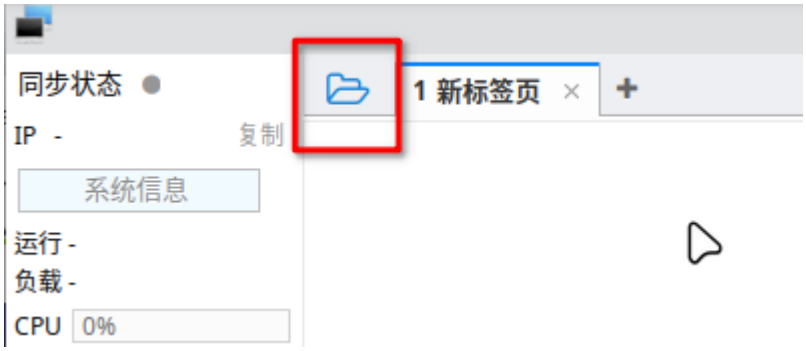


该终端中含有 ssh 命令

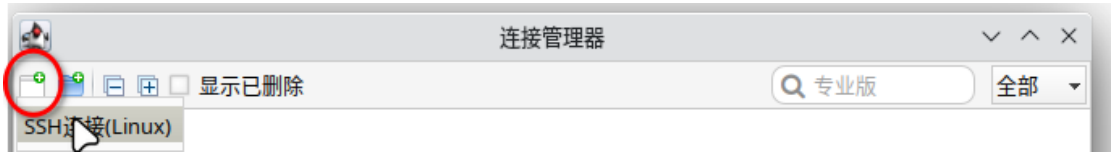
2.使用集成了远程连接功能的外部软件，如 xshell,finalshell 等,这里使用的是 finalshell

下载地址:http://www.hostbuf.com/downloads/finalshell_install.exe

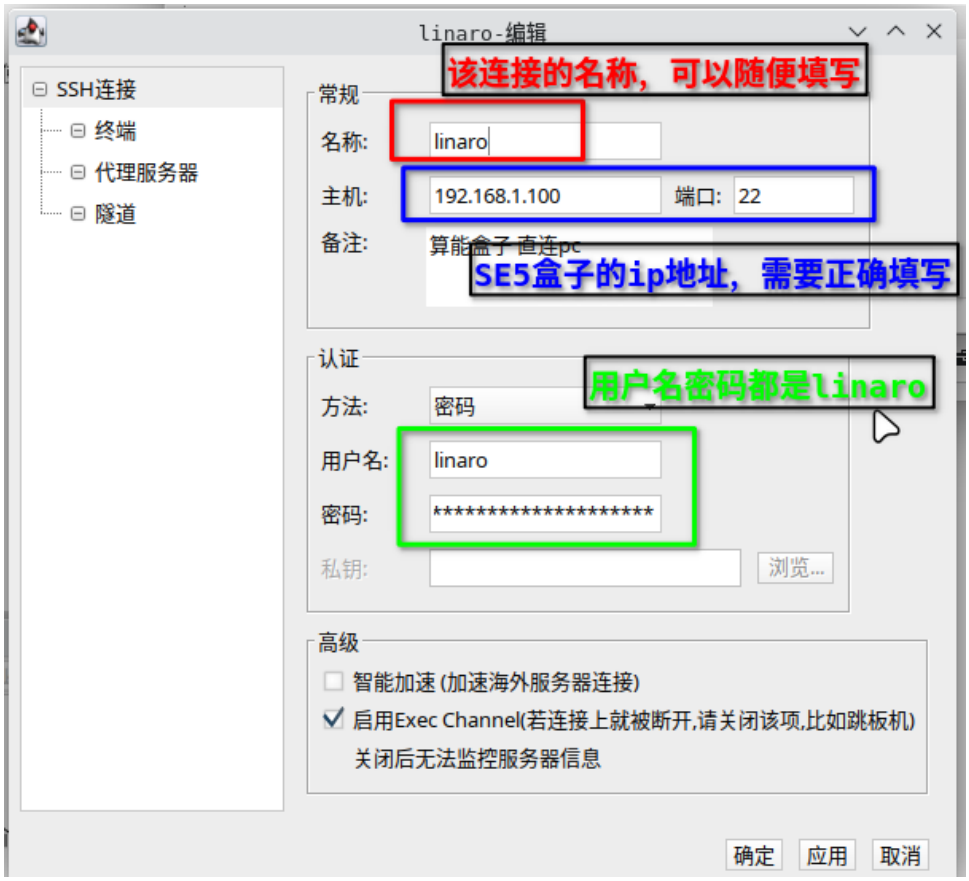
安装后启动



点击这个文件夹，然后再点击左上角图标，创建一个 ssh 连接



填入相关信息即可



首次连接会要求接收密钥，点永远接受。

2.1.4 SE5 盒子的文件上传下载设置

在 PC 与 SE5 盒子之间互传文件有多种方式，这里简单介绍三种，分别是使用 samba 共享文件夹、finalshell 上传下载文件、scp 命令上传下载文件，读者可自己选择一种

2.1.4.1 使用 samba 共享文件夹

优点：启动 samba 服务后，将 SE5 盒子的指定文件夹共享到局域网内，PC 机可以通过网络直接打开该文件夹，实现拖拽文件上传下载，直观方便

首先在 ***se5 盒子***上安装 samba：

```
sudo apt install samba
```

之后进行简单配置：

```
mkdir ~/project
chmod 777 -R ~/project
```

之后使用自己习惯的文本编辑器打开 samba 配置文件：

```
sudo vim /etc/samba/smb.conf
# 或使用 nano
# sudo nano /etc/samba/smb.conf
```

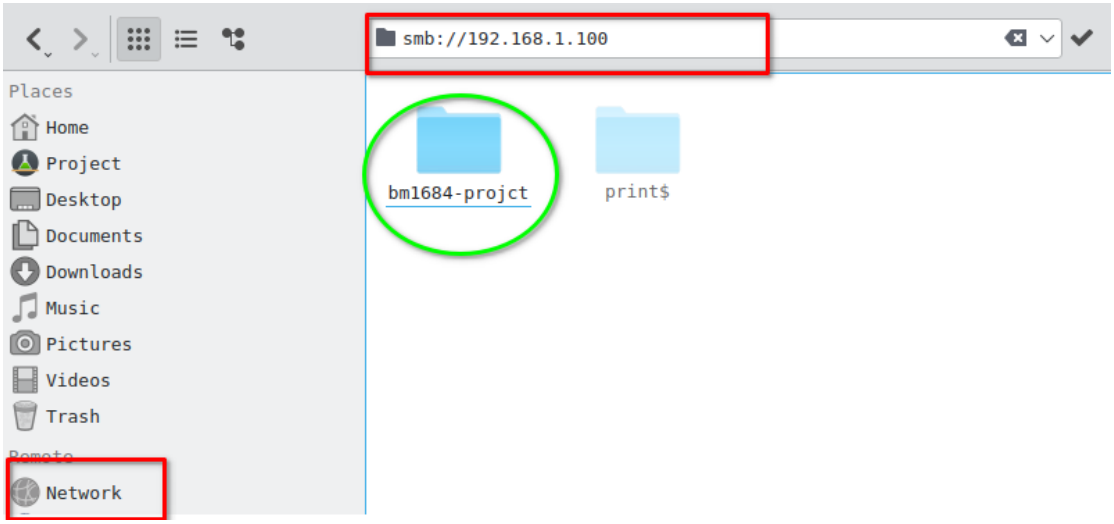
在配置文件的底部追加如下内容：

```
[bm1684-project]
path=/home/linaro/project
security=share
writeable=yes
create mask=0777
directory mask=0777
public=yes
browseable=yes
```

最后重启 samba 服务：

```
sudo systemctl restart smbd
```

随后 linux 下可以打开文件管理器，找到“网络”（或是“其他位置”），在搜索框中输入 smb://SE5 的 ip 地址，回车，即可打开共享文件夹



windows 下可以右键 我的电脑->映射网络位置，然后按提示输入地址即可打开。

2.1.4.2 使用 finalshell 上传下载文件

FinalShell 是一体化的的服务器,网络管理软件,不仅是 ssh 客户端,还是功能强大的开发,运维工具,充分满足开发,运维需求。

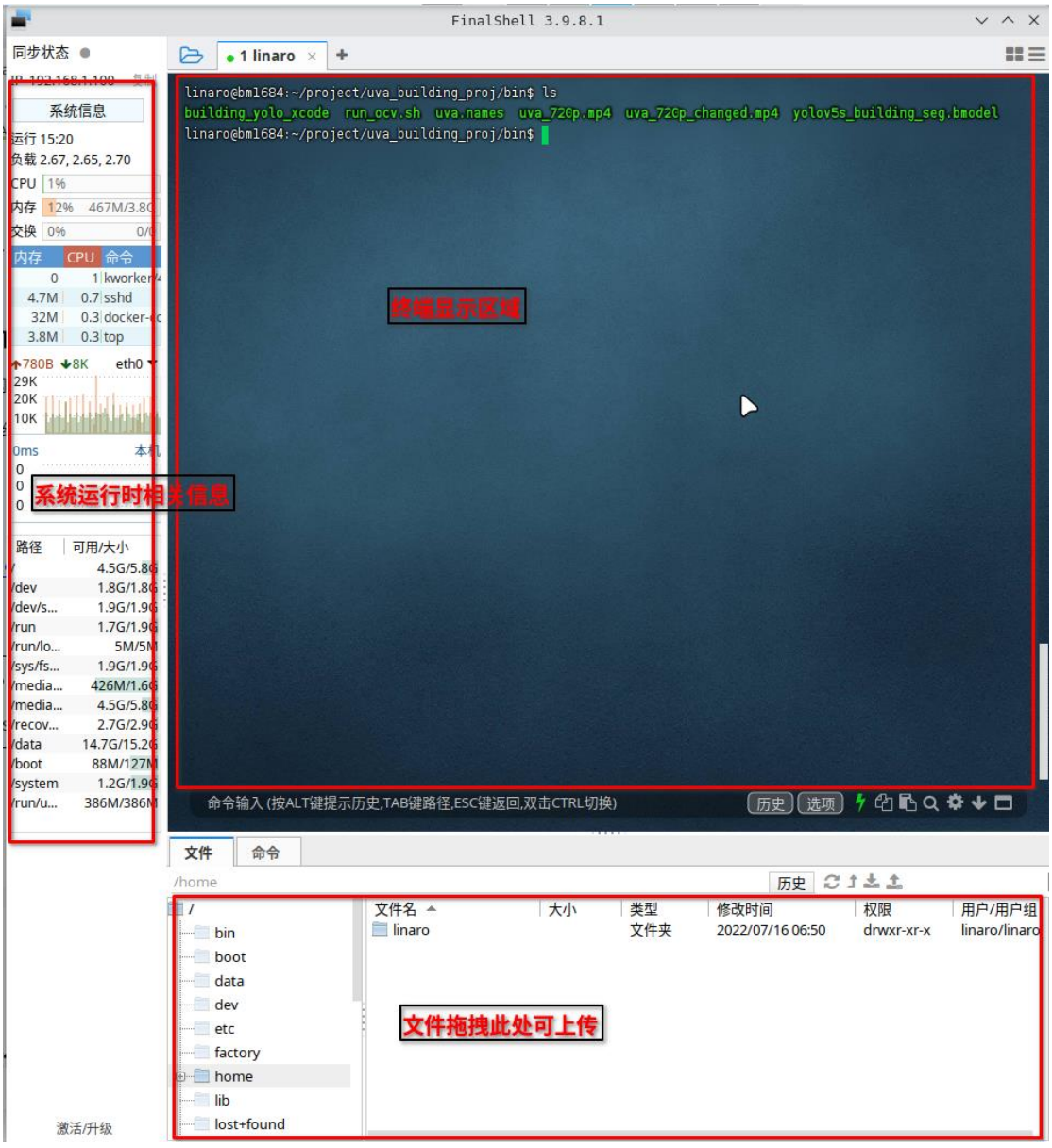
优点：基础功能免费，安装简便，无需复杂配置，支持文件拖拽上传下载。

windows 版下载地址：http://www.hostbuf.com/downloads/finalshell_install.exe

linux 一键安装命令：

```
wget www.hostbuf.com/downloads/finalshell_install_linux.sh;chmod +x finalshell_install_linux.sh;./finalshell_install_linux.sh;
```

其连接后界面如下：



2.1.4.3 使用 scp 命令传送文件

这种方法最简单最便捷，无需下载无需配置

使用如下命令进行传送：

```
scp 本地文件路径 linaro@se5 盒子 ip:欲存放在 se5 上的路径
```

如：传送当前目录下的 test.txt 文件到 se5 盒子的 test 文件夹下：

```
scp ./test.txt linaro@192.168.1.100:~/test/
```

如果要传输的是文件夹，加上-r 标志即可，如：

```
scp -r ./testFolder linaro@192.168.1.100:~/
```

2.1.5 配置 rtmp 服务器

这里的 rtmp 服务器可以自己使用 nginx 来搭建，也可以将视频流推送到市场主流的直播平台上。

2.1.5.1 安装 nginx 作为 rtmp 服务器

根据自己 pc 的系统类型选择相应的平台安装

windows 平台：

1. 下载 nginx：

下载 nginx(http://nginx-win.ecsds.eu/download/)，注意，一定要选择 nginx 1.7.11.3 Gryphon.zip 这个版本，或者 http://nginx-win.ecsds.eu/download/nginx 1.7.11.3 Gryphon.zip 直接下载，据说只有这个版本的 nginx 在编译时是加入了 rtmp 模块的，其他版本的都没有，包括 nginx 官方(http://nginx.org/)下载的也是没有包含 rtmp 模块的。

可在此 https://github.com/arut/nginx-rtmp-module.git 单独下载 rtmp 模块。

2. 解压

3. 配置 nginx

打开 nginx/conf 目录，新建一个文件：nginx.conf，然后在这个文件中输入如下内容：

```
worker_processes  1;

events {
    worker_connections  1024;
}

rtmp {
    server {
        listen 1935;
        chunk_size 4000;
        application live {
            live on;
            allow publish 127.0.0.1;
            allow play all;
        }
    }
}
```

Linux 平台：

linux 平台中通过包管理器下载 nginx 不含 rtmp 模块，需要自己编译，构建过程如下：

```
    # 获取 nginx-1.20.2 的源码
    export nginx_dir=${PWD}
    wget https://nginx.org/download/nginx-1.20.2.tar.gz
    git clone https://github.com/arut/nginx-rtmp-module
    tar -xvf nginx-1.20.2.tar.gz
    cd nginx-1.20.2

    ./configure --add-module=../nginx-rtmp-module --prefix=${nginx_dir}/build

    make && make install
```

构建完成后，同样进行配置修改，配置文件位于${nginx_dir}/build/conf/nginx.conf,删除原有内容，添加如下内容：

```
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;
events {
    worker_connections  8192;
}
rtmp {
    server {
        listen 1935;
        chunk_size 4000;
        application live {
            live on;
        }
    }
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        off;
    server_names_hash_bucket_size 128;
    client_body_timeout   10;
    client_header_timeout 10;
    keepalive_timeout     30;
    send_timeout          10;
    keepalive_requests    10;
    server {
        listen       80;
        server_name  localhost;
        location /stat {
            rtmp_stat all;
            rtmp_stat_stylesheet stat.xsl;
        }
        location /stat.xsl {
            root nginx-rtmp-module/;
        }
        location /control {
            rtmp_control all;
```

```
            }
            location / {
                root    html;
                index   index.html index.htm;
            }
            error_page      500 502 503 504    /50x.html;
            location = /50x.html {
                root    html;
            }
        }
    }
```

2.1.5.2 推送视频流至直播平台上

这里的直播平台以 bilibili 为例。

首先注册一个 bilibili 账号，找到个人中心进入->直播中心->我的直播间->开播设置

随后选择直播类型以及填写房间名称

点击开始直播后，会需要进行一次实名认证，如实填写后等待审核通过即可。
若审核通过，再次点击开始直播，会得到 bilibili 提供的 rtmp 地址以及串流密钥，如下：

记住服务器地址与串流密钥即可，后续步骤会使用到。

2.2 准备模型

- 获取 pt 模型 使用‘uva_building_proj\sources’里的’yolov5s_building_seg.pt’文件

2.2.1 下载 yolov5 源码

- 以下操作都在 ***docker 容器***中进行

```
# 首先创建一个存放 yolo 相关的文件夹,以及一个项目文件夹用以存放代码
mkdir /workspace/YOLOv5
mkdir /workspace/project
# 将该文件夹的路径通过 export 声明为环境变量
export YOLOv5=/workspace/YOLOv5
cd ${YOLOv5}

# 克隆源代码
git clone https://github.com/ultralytics/yolov5.git yolov5_github



cd yolov5_github
# 使用 tag 从远程创建本地 v6.1 分支
git branch v6.1 v6.1
# 下载 yolov5s v6.1 版本
wget https://github.com/ultralytics/yolov5/releases/download/v6.1/yolov5s.pt
```

2.2.2 修改 models/yolo.py

- 通过修改该文件中的 Detect 类的 forward 函数的最后 return 语句，实现不同的输出。在本次实验中，需要使用`4 个输出`

```
def forward(self, x):
    z = [] # inference output
    for i in range(self.nl):
        x[i] = self.m[i](x[i]) # conv
        bs, _, ny, nx = x[i].shape # x(bs,255,20,20) to x(bs,3,20,20,85)
        x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

        if not self.training: # inference
            if self.dynamic or self.grid[i].shape[2:4] != x[i].shape[2:4]:
                self.grid[i], self.anchor_grid[i] = self._make_grid(nx, ny, i)

            if isinstance(self, Segment): # (boxes + masks)
                xy, wh, conf, mask = x[i].split((2, 2, self.nc + 1, self.no - self.nc - 5), 4)
                xy = (xy.sigmoid() * 2 + self.grid[i]) * self.stride[i] # xy
                wh = (wh.sigmoid() * 2) ** 2 * self.anchor_grid[i] # wh
                y = torch.cat((xy, wh, conf.sigmoid(), mask), 4)
            else: # Detect (boxes only)
                xy, wh, conf = x[i].sigmoid().split((2, 2, self.nc + 1), 4)
                xy = (xy * 2 + self.grid[i]) * self.stride[i] # xy
                wh = (wh * 2) ** 2 * self.anchor_grid[i] # wh
```

```
                y = torch.cat((xy, wh, conf), 4)
            z.append(y.view(bs, self.na * nx * ny, self.no))

        #return x if self.training else x #3 个输出
        #return x if self.training else (torch.cat(z, 1)) #1 个输出
        return x if self.training else (torch.cat(z, 1),) if self.export else (torch.cat(z, 1), x) #4 个输出
```

2.2.3 导出 JIT 模型

- 要运行导出 JIT 模型的 py 文件需要使用 pip 安装许多的依赖和包，为了不污染容器的 python 的原生包环境，下面会使用 python 的虚拟环境`virtualenv`
- SophonSDK 中的 PyTorch 模型编译工具 BMNETP 只接受 PyTorch 的 JIT 模型（TorchScript 模型）。
- JIT（Just-In-Time）是一组编译工具，用于弥合 PyTorch 研究与生产之间的差距。它允许创建可以在不依赖 Python 解释器的情况下运行的模型，并且可以更积极地进行优化。在已有 PyTorch 的 Python 模型（基类为 torch.nn.Module）的情况下，通过 torch.jit.trace 就可以得到 JIT 模型，如 torch.jit.trace(python_model, torch.rand(input_shape)).save('jit_model')。BMNETP 暂时不支持带有控制流操作（如 if 语句或循环）的 JIT 模型，因此不能使用 torch.jit.script，而要使用 torch.jit.trace，它仅跟踪和记录张量上的操作，不会记录任何控制流操作。这部分操作 yolov5 已经为我们写好，只需运行如下命令即可导出符合要求的 JIT 模型：

```
cd ${YOLOv5}/yolov5_github
#安装  python 虚拟环境 virtualenv
pip3 install virtualenv

#  切换到虚拟环境
virtualenv -p python3 --system-site-packages env_yolov5
source env_yolov5/bin/activate

#  安装依赖
pip3 install -r requirements.txt#  此过程遇到依赖冲突或者错误属正常现象
#  导出 jit 模型
python3 export.py --weights yolov5s_building_seg.pt --include    torchscript
#  退出虚拟环境
deactivate
#  将生成好的 jit 模型 yolov5s.torchscript 拷贝到${YOLOv5}/build 文件夹下
mkdir ${YOLOv5}/build

cp yolov5s_building_seg.torchscript   ${YOLOv5}/build/yolov5s_coco_v6.1_4output.trace.pt

#  拷贝一份到${YOLOv5}/data/models 文件夹下
mkdir -p    ${YOLOv5}/data/models
```

```
cp                                                    yolov5s_building_seg.torchscript
${YOLOv5}/data/models/yolov5s_coco_v6.1_4output.trace.pt
```

2.3 模型转换

- 以下操作***都在 docker 容器***中进行

2.3.1 创建辅助脚本

- 首先创建几个 shell 脚本来辅助转换，以下的脚本应该创建于**${YOLOv5}/yolov5_github** 目录中

```
cd ${YOLOv5}/yolov5_github
```

- 首先**创建并编写** model_info.sh 文件,可以通过修改 model_info.sh 中的模型名称、生成模型目录和输入大小 shapes、使用的量化 LMDB 文件目录、batch_size、img_size 等参数：

```
#!/bin/bash
root_dir=$(cd `dirname $BASH_SOURCE[0]`/../ && pwd)
build_dir=$root_dir/build
src_model_file=$build_dir/"yolov5s_coco_v6.1_4output.trace.pt"
src_model_name=`basename ${src_model_file}`
dst_model_prefix="yolov5s"
dst_model_postfix="coco_v6.1_4output"
fp32model_dir="fp32model"
int8model_dir="int8model"
#lmdb_src_dir="${build_dir}/coco/images/val2017/"
image_src_dir="${build_dir}/coco_images_200"
# lmdb_src_dir="${build_dir}/coco2017val/coco/images/"
#lmdb_dst_dir="${build_dir}/lmdb/"
img_size=${1:-640}
batch_size=${2:-1}
iteration=${3:-2}
img_width=640
img_height=640

function check_file()
{
    if [ ! -f $1 ]; then
        echo "$1 not exist."
        exit 1
    fi
}

function check_dir()
```

```
{
    if [ ! -d $1 ]; then
        echo "$1 not exist."
        exit 1
    fi
}
```

- 然后**创建并编写** gen_fp32bmodel.sh 文件：

```
#!/bin/bash
source model_info.sh
pushd $build_dir
check_file $src_model_file
python3 -m bmnetp --mode="compile" \
        --model="${src_model_file}" \
        --outdir="${fp32model_dir}/${batch_size}" \
        --target="BM1684" \
        --shapes=[[${batch_size},3,${img_height},${img_width}]] \
        --net_name=$dst_model_prefix \
        --opt=2 \
        --dyn=False \
        --cmp=True \
        --enable_profile=True

dst_model_dir=${root_dir}/data/models
if [ ! -d "$dst_model_dir" ]; then
    echo "create data dir: $dst_model_dir"
    mkdir -p $dst_model_dir
fi
cp "${fp32model_dir}/${batch_size}/compilation.bmodel"
"${dst_model_dir}/${dst_model_prefix}_${img_size}_${dst_model_postfix}_fp32_${batch_size}b.bmo
del"

popd
```

2.3.2 生成 FP32 BModel

执行以下命令，使用 bmnetp 编译生成 FP32 BModel

```
chmod +x gen_fp32bmodel.sh

./gen_fp32bmodel.sh
```

若成功应有如下输出：

```
============================================================
*** Instruction generation process for subnet 0
============================================================
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
piler:I] subnet input tensor name=input.1
I0130 23:06:20.993304 190066 bmcompiler_context_subnet.cpp:231] [BMCompiler:I] subnet output tensor name=172
I0130 23:06:20.993312 190066 bmcompiler_context_subnet.cpp:231] [BMCompiler:I] subnet output tensor name=171
I0130 23:06:21.002658 190066 bmcompiler_context.cpp:356] [BMCompiler:I] set_stage param cur_net_idx = 0

============================================================
*** Store bmodel of BMCompiler...
============================================================
I0130 23:06:21.009158 190066 bmcompiler_bmodel.cpp:153] [BMCompiler:I] save_tensor input name [input.1]
I0130 23:06:21.009182 190066 bmcompiler_bmodel.cpp:153] [BMCompiler:I] save_tensor output name [172]
I0130 23:06:21.009188 190066 bmcompiler_bmodel.cpp:153] [BMCompiler:I] save_tensor output name [171]
I0130 23:06:21.230631 190066 bmcompiler_bmodel.cpp:153] [BMCompiler:I] save_tensor input name [input.1]
I0130 23:06:21.230670 190066 bmcompiler_bmodel.cpp:153] [BMCompiler:I] save_tensor output name [172]
I0130 23:06:21.230679 190066 bmcompiler_bmodel.cpp:153] [BMCompiler:I] save_tensor output name [171]
Compiling succeeded.
/workspace/YOLOv5/yolov5_github
```

并在${YOLOv5}/data/models 下生成 yolov5s_640_coco_v6.1_4output_fp32_1b.bmodel，即转换好的 FP32 BModel，使用 bm_model.bin --info 查看的模型具体信息如下

```
root@archlinux:~# cd ${YOLOv5}/data/models/
root@archlinux:/workspace/YOLOv5/data/models# bm_model.bin --info yolov5s_640_coco_v6.1_4output_fp32_1b.bmodel
bmodel version: B.2.2
chip: BM1684
create time: Mon Jan 30 23:06:21 2023

=========================================
net 0: [yolov5s]  static
------------
stage 0:
input: input.1, [1, 3, 640, 640], float32, scale: 1
output: 172, [1, 32, 160, 160], float32, scale: 1
output: 171, [1, 25200, 38], float32, scale: 1

device mem size: 66084864 (coeff: 30238080, instruct: 252544, runtime: 35594240)
host mem size: 0 (coeff: 0, runtime: 0)
root@archlinux:/workspace/YOLOv5/data/models#
```

2.4 编译建筑识别与对比程序

2.4.1 获取源码包

- 以下操作在 ***docker 宿主机***中进行

首先将`uva_building_proj.zip`下载到 docker 宿主机，即虚拟机中。这里假设该文件被放到了~/Downloads/中。

然后查看算能开发环境 docker 的 ID：

```
docker ps
```

```
> $ docker ps                                                                                    [±master ●●]
CONTAINER ID   IMAGE                                          COMMAND   CREATED          STATUS          PORTS     NAMES
e6880c8ed039   sophgo/sophonsdk3:ubuntu18.04-py37-dev-22.06   "bash"    35 minutes ago   Up 35 minutes             modest_kel
ler
```

记住 CONTAINER ID 的内容，然后使用 docker cp 命令将文件拷入 docker 环境中：

```
docker cp ~/Downloads/uva_building_proj.zip e6880c:/workspace/project
```

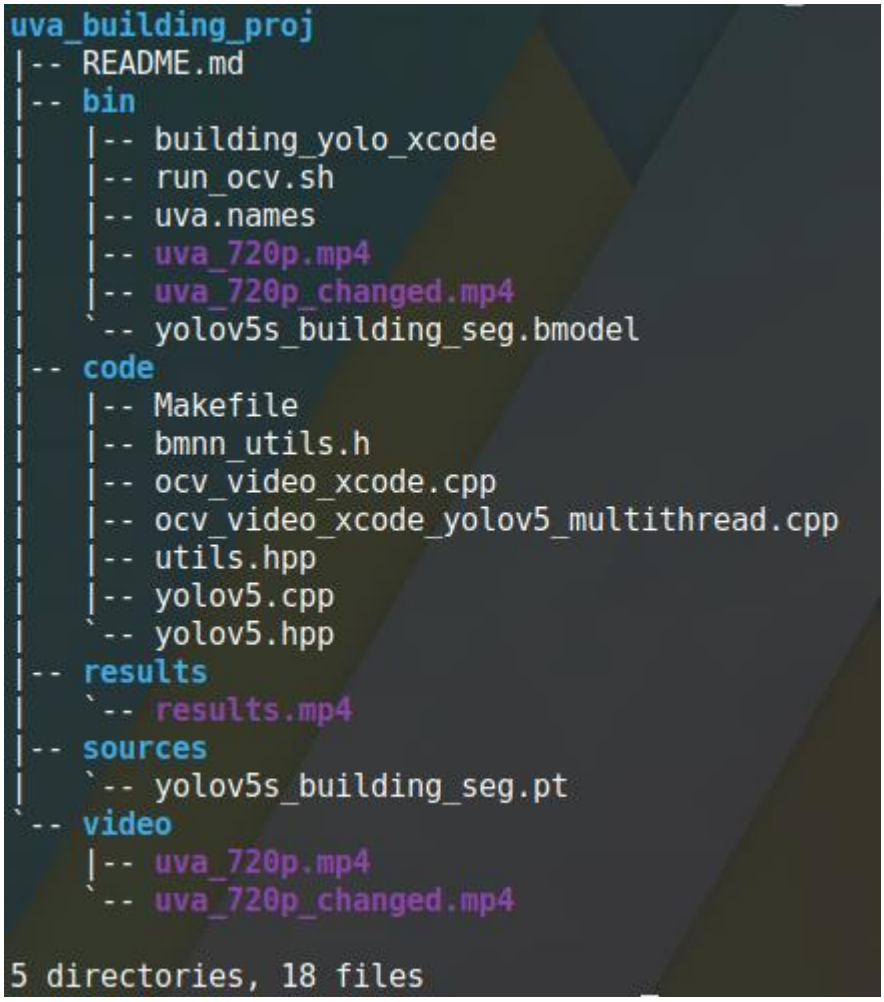
命令中的 e6880c 换成刚刚查看的 CONTAINER ID 的值，一般只需要输入前 6 位即可。

- 以下操作在 ***docker*** 中进行

进入 docker 后，首先解压刚刚拷贝进来的 uva_building_proj.zip 文件

```
cd /workspace/project
unzip uva_building_proj.zip
```

若解压成功，查看 uva_building_proj 文件夹应该是如下结构：



2.4.2 编译代码

- 以下操作在 ***docker*** 中进行

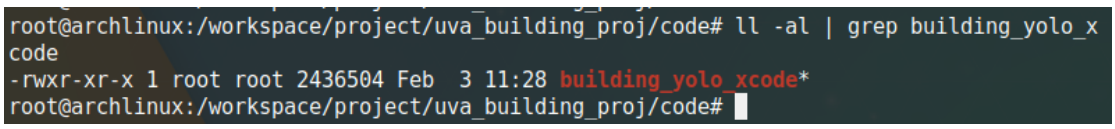
进入源码目录并编译：

```
cd /workspace/project/uva_building_proj/code
make
```

并编译过程中会产生部分 waring,无需理会，make 指令执行后查看当前目录下是否生产了 building_yolo_xcode 可执行文件：

```
ll -al | grep building_yolo_xcode
```

观察是否有如下输出，若有则 make 编译成功。



2.4.3 整理欲上传的文件

在/workspace/project/uva_building_proj/bin 中有已经编译好且测试通过了的文件，可以选择将整个 bin 文件夹上传至 SE5 盒子中运，然后按 2.5 节运行。也可以按如下说明整理自己的 rebuild-bin 文件夹，来测试刚刚编译的程序以及 2.2 节转换的模型是否可用。

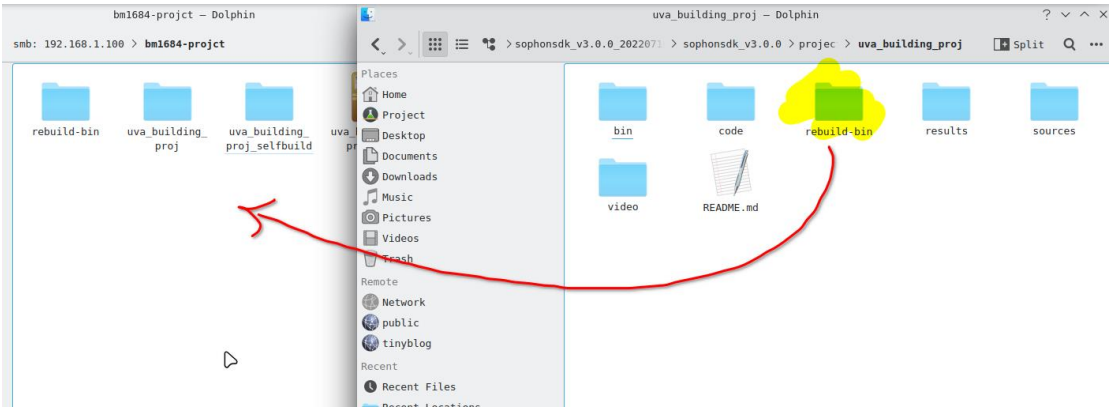
```
cd /workspace/project/uva_building_proj/

mkdir rebuild-bin

cp video/* rebuild-bin

cp ${YOLOv5}/data/models/yolov5s_640_coco_v6.1_4output_fp32_1b.bmodel rebuild-bin/yolov5s_building_seg.bmodel

cp code/building_yolo_xcode rebuild-bin

echo “building” >> rebuild-bin/uva.names
```

2.5 运行建筑识别与对比程序

2.5.1 将整理好的文件上传至 SE5 算能盒子中

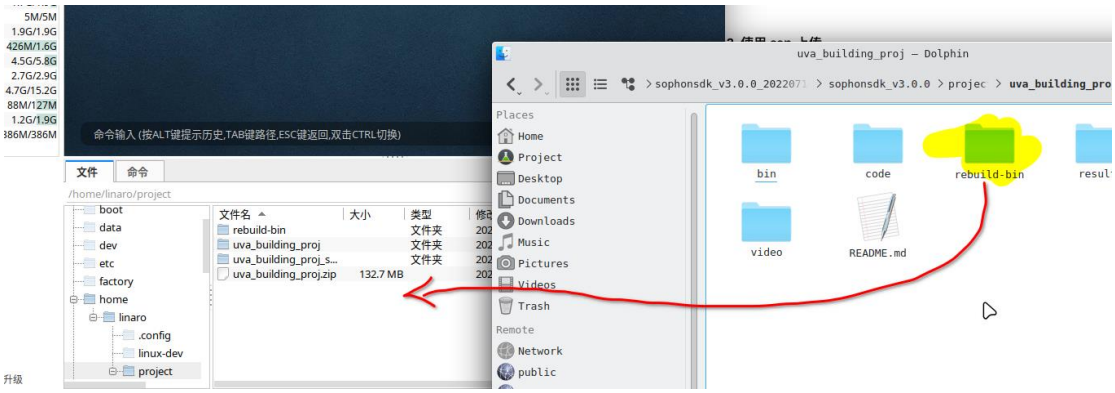
2.5.1.1 使用 samba 共享文件夹上传

打开共享文件夹和 docker 挂在的/workspace 的目录，直接将整理好的文件拖拽或是复制粘贴过去



2.5.1.2 使用 finalshell 上传

同样是直接拖进去



2.5.1.3 使用 scp 上传

在 ***docker*** 中执行如下命令：

```
cd /workspace/project/uva_building_proj/

# linaro@ip   ip 换成算能盒子的 ip 地址
scp -r rebuild-bin linaro@192.168.1.100:~/project
```

2.5.2 运行前准备

首先需要启动 nginx 服务，确保按照 2.1 节完成了 nginx 的安装配置。

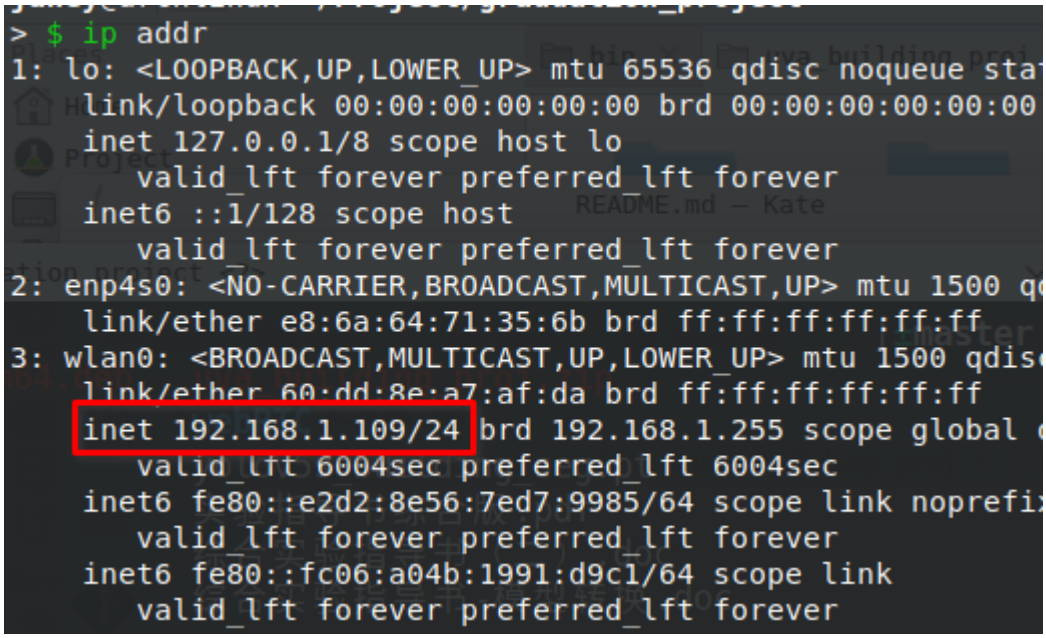
以下操作在 ***PC 中***完成，根据自己 pc 的系统类型选择**相应**的平台

- linux 平台

```
cd ${nginx_dir}/build/sbin
sudo ./nginx -c ../conf/nginx.conf
# 运行 sudo ./nginx -s stop 关闭 nginx
```

查看当前 ip 地址

```
ip addr
```



那么我的推拉流地址是 rtmp://192.168.1.109:1935/live/test,依据自己的 ip 进行修改

此时可以使用 ffplay 对该地址进行拉流

```
ffplay rtmp://192.168.1.109:1935/live/test
```

```
> $ ffplay rtmp://127.0.0.1:1935/live/test                                    [±master ●●]
ffplay version n5.1.2 Copyright (c) 2003-2022 the FFmpeg developers
  built with gcc 12.2.0 (GCC)
  configuration: --prefix=/usr --disable-debug --disable-static --disable-stripping --enable-amf --enable
-avisynth --enable-cuda-llvm --enable-lto --enable-fontconfig --enable-gmp --enable-gnutls --enable-gpl -
-enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libbs2b --enable-libdav1d --en
able-libdrm --enable-libfreetype --enable-libfribidi --enable-libgsm --enable-libiec61883 --enable-libjac
k --enable-libmfx --enable-libmodplug --enable-libmp3lame --enable-libopencore_amrnb --enable-libopencore
_amrwb --enable-libopenjpeg --enable-libopus --enable-libpulse --enable-librav1e --enable-librsvg --enabl
e-libsoxr --enable-libspeex --enable-libsrt --enable-libssh --enable-libsvtav1 --enable-libtheora --enabl
e-libv4l2 --enable-libvidstab --enable-libvmaf --enable-libvorbis --enable-libvpx --enable-libwebp --enab
le-libx264 --enable-libx265 --enable-libxcb --enable-libxml2 --enable-libxvid --enable-libzimg --enable-n
vdec --enable-nvenc --enable-opencl --enable-opengl --enable-shared --enable-version3 --enable-vulkan
  libavutil      57. 28.100 / 57. 28.100
  libavcodec     59. 37.100 / 59. 37.100
  libavformat    59. 27.100 / 59. 27.100
  libavdevice    59.  7.100 / 59.  7.100
  libavfilter     8. 44.100 /  8. 44.100
  libswscale      6.  7.100 /  6.  7.100
  libswresample   4.  7.100 /  4.  7.100
  libpostproc    56.  6.100 / 56.  6.100
    nan    :  0.000 fd=   0 aq=    0KB vq=    0KB sq=    0B f=0/0
```

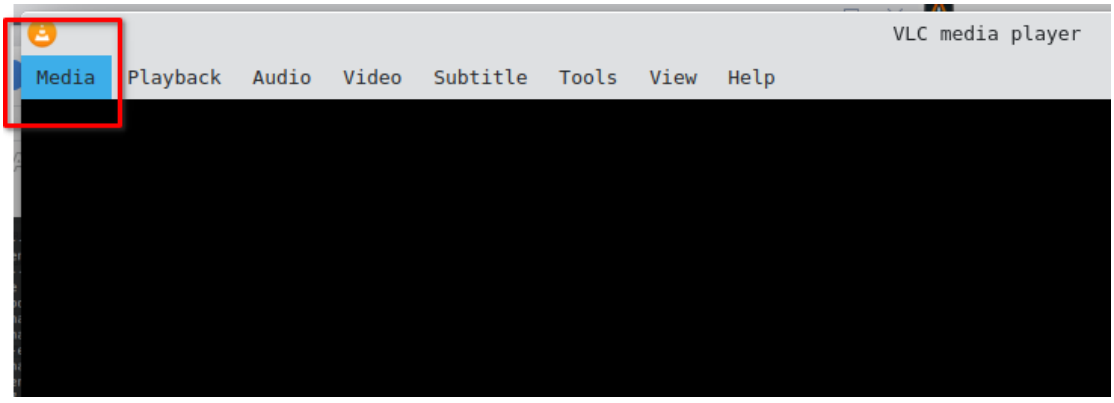
此时就开始拉流，但暂时还没有任何画面，等到待会 SE5 盒子运行我们编译的程序后，便会向该 rtmp 地址推流，fflpay 就会有所显示。

- windows 平台

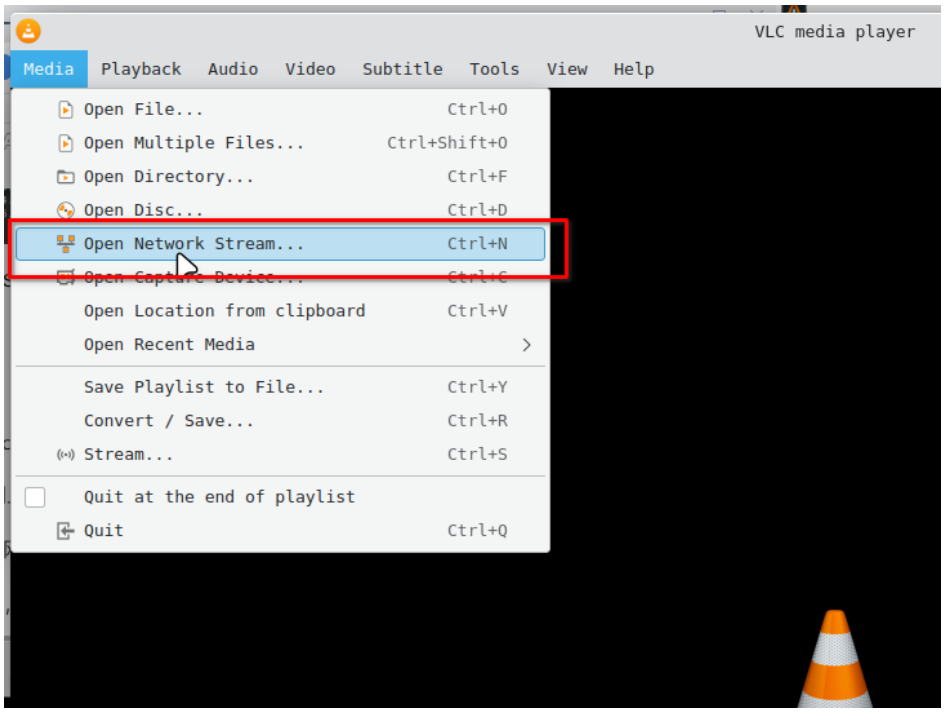
在 nginx 目录打开 cmd，输入`nginx`或`nginx -c conf/nginx.conf`来启动 nginx，推拉流的地址同上，获取电脑 ip 后自己进行改写即可。

然后打开 vlc，开启网络串流并进行拉流：

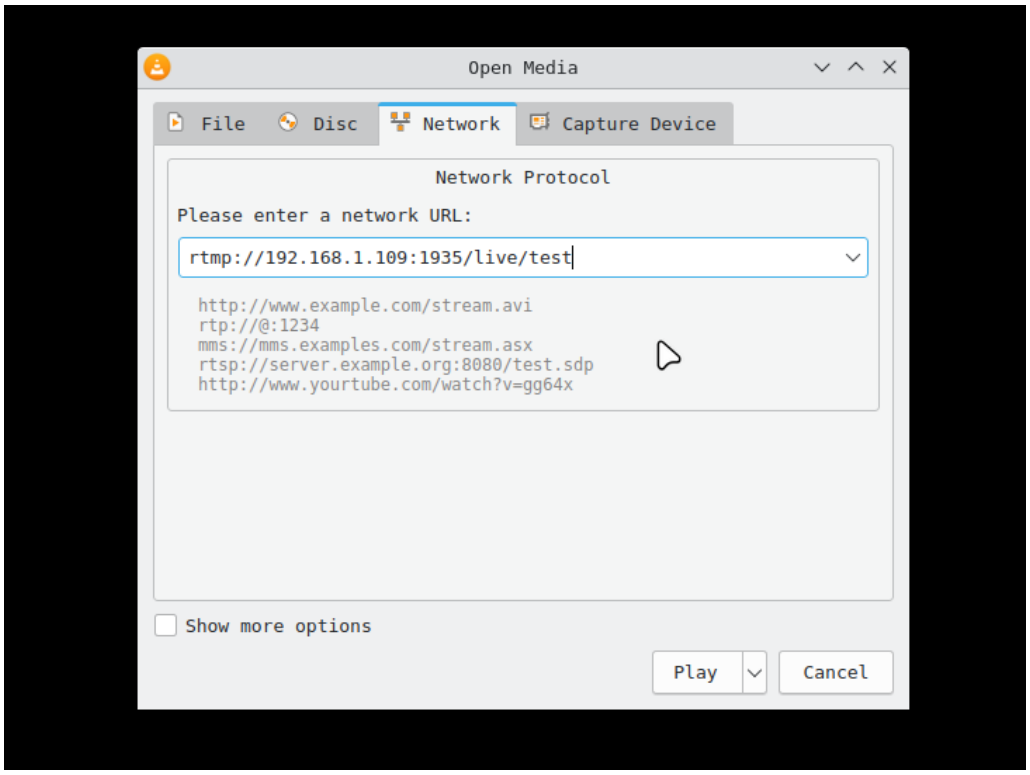
1.点击左上角 Media,打开 Media 菜单



2. 选中 Open Network Stream



3. 完成如下填写



然后点击 Play 即可，同样是暂时没有画面，等到待会 SE5 盒子运行我们编译的程序后，便会向该 rtmp 地址推流，vlc 就会有所显示。

- Android 平台

安卓平台需要先下载软件 VLC for Android,随后打开该 app,点击右下角的“更多”选项

卡，在串流栏点击“新建串流”，然后输入网络地址，即 rtmp 地址，点击右边的箭头开始拉流

2.5.3 运行

以下操作*在 SE5 盒子中*完成

进入刚刚传过来的文件目录

```
cd ~/project/rebuild-bin
 #或是 cd ~/project/bin
```

生成运行脚本 run_ocv.sh，命令行输入如下命令，按自己的 rtmp 推流地址修改 --outputName 参数的值，如：

使用自建的 nginx 推拉流服务器则输入以下命令：

```
echo chmod +x ./building_yolo_xcode;./building_yolo_xcode --input1="./uva_720p.mp4" --input2="./uva_720p_changed.mp4" --bmodel=./yolov5s_building_seg.bmodel --classnames=./uva.names --code_type="H264enc" --frameNum=1200 --outputName="rtmp://192.168.1.109:1935/live/test" --encodeparams="bitrate=4000" --roi_enable=1 --videoSavePath="./video_clips"
```

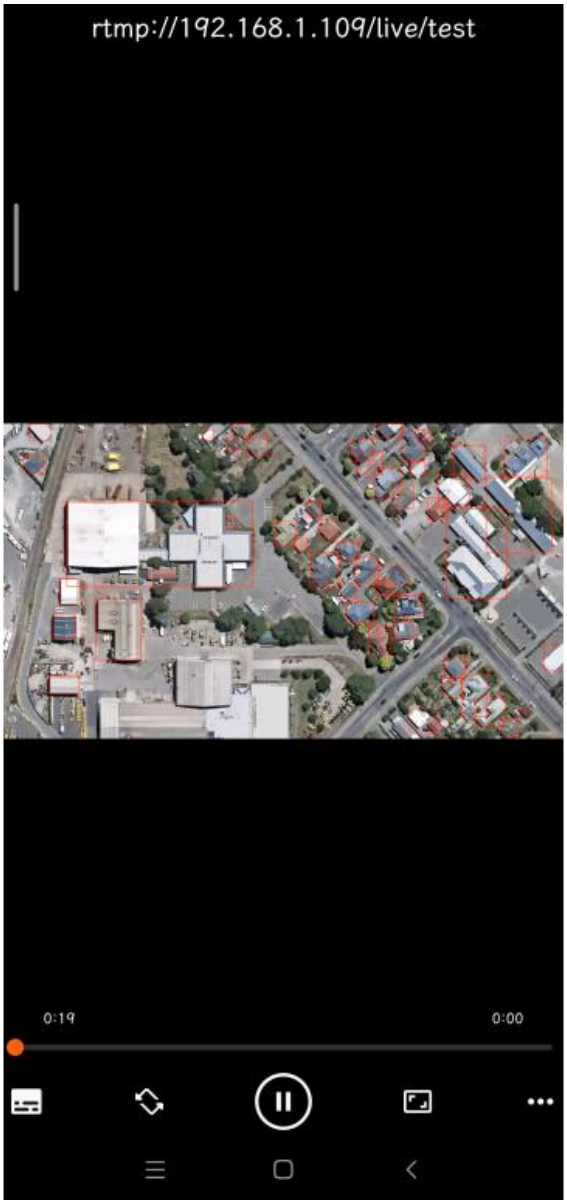
命令行输入如下命令来运行：

```
sh ./run_ocv.sh
```

运行之后若无报错，稍等数秒之后，之前打开的 ffplay 或 vlc 就会开始显示画面，例如

下：


rtmp://127.0.0.1:1935/live/test


rtmp://192.168.1.109/live/test
0:19
0:00

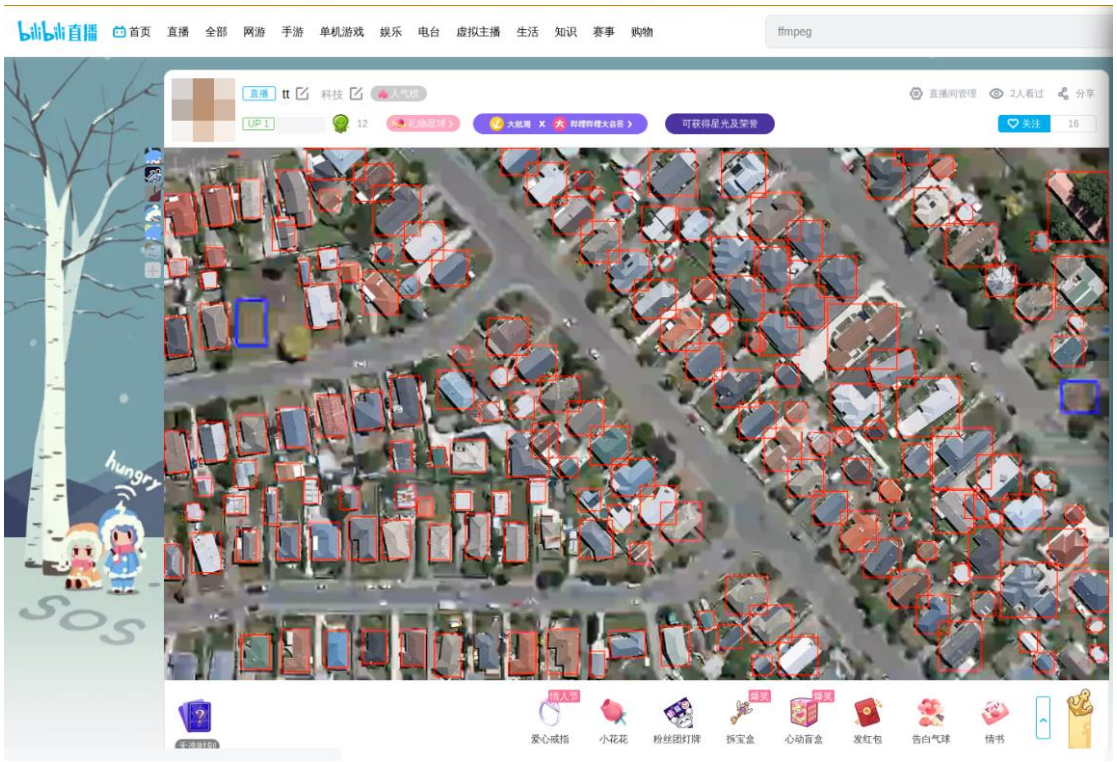
若使用第三方推拉流服务器则输入以下命令：

```
echo chmod +x ./building_yolo_xcode;./building_yolo_xcode --input1="./uva_720p.mp4" --input2="./uva_720p_changed.mp4" --bmodel=./yolov5s_building_seg.bmodel --classnames=./uva.names --code_type="H264enc" --frameNum=1200 --outputName="rtmp 服务器地址+串流密钥" --encodeparams="bitrate=4000" --roi_enable=1 --videoSavePath="./video_clips"
```

命令行输入如下命令来运行：

```
sh ./run_ocv.sh
```

运行之后若无报错，稍等数秒之后，打开自己的直播间即可看到如下：



SE5 保存结果视频 5S 一个，如下图所示


202326_205336.mp4
202326_205342.mp4
202326_205348.mp4
202326_205353.mp4
202326_205359.mp4
202326_205412.mp4
202326_205418.mp4
202326_205424.mp4
202326_205429.mp4
202326_205435.mp4
202326_205440.mp4
202326_205445.mp4
202326_205450.mp4
202326_205455.mp4
202326_20546.mp4
202326_20550.mp4
202326_205510.mp4
202326_205515.mp4
202326_205521.mp4
202326_205526.mp4
202326_205531.mp4
202326_205536.mp4
202326_205542.mp4
202326_205547.mp4
202326_20555.mp4