**3588板模型和算法移植流程**

3588算法移植流程有多个步骤，现总结流程和注意点于此。

SDK工程位置：**http://192.168.1.173:8081/huboheng/android\_sdk.git**

**一. 原始pt模型转为onnx模型**

下载yolov5工程，进入工程

python export.py --weights your-weight.pt --include onnx

注意设置模型输入尺寸。

生成onnx模型会和pt模型在同一文件夹下。

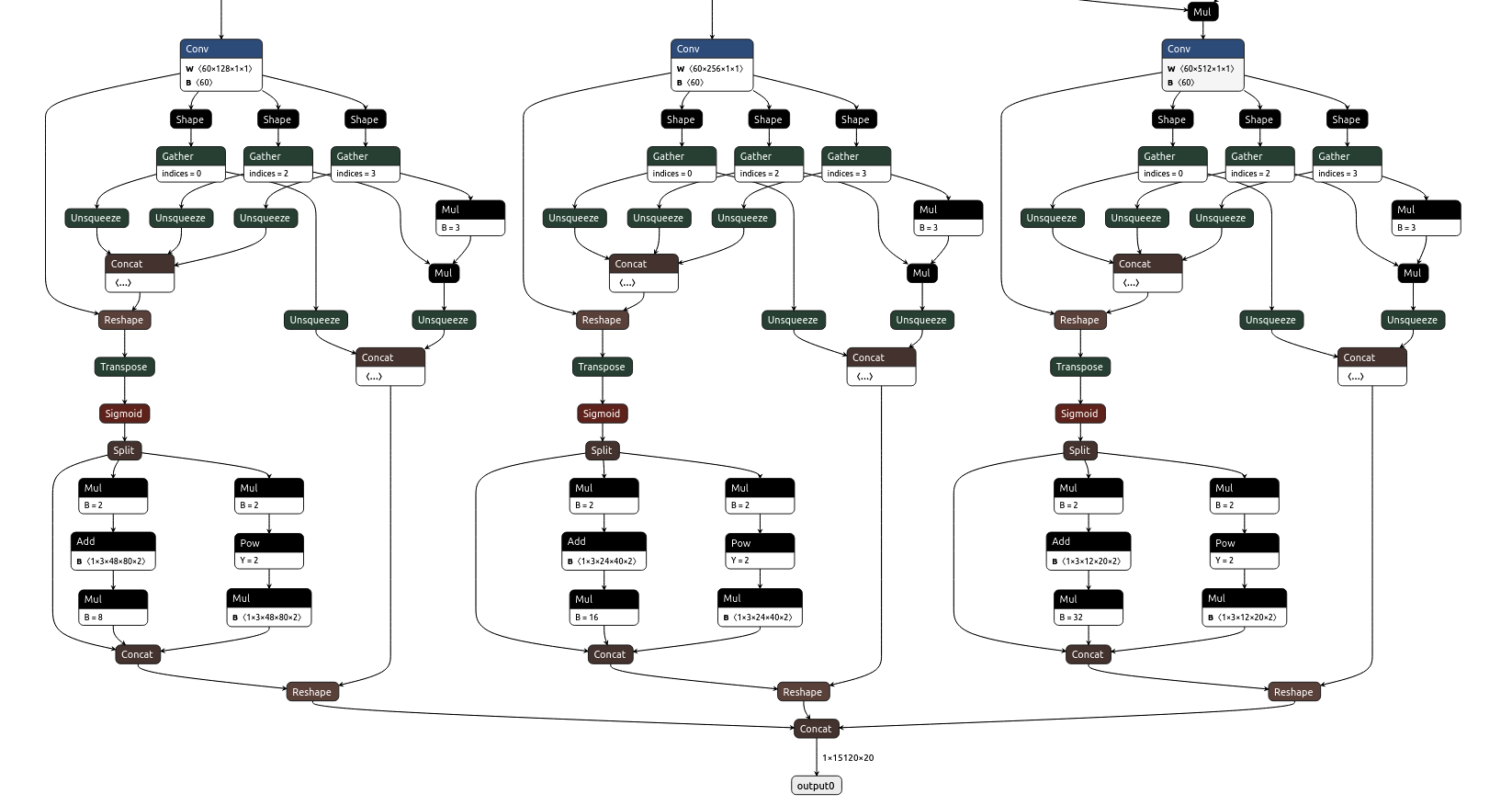
Use the line above in yolov5 project to generate onnx model

**二.onnx模型结构修改**

3588上的推理SDK现在使用的后处理逻辑，其要求RKNN模型有三个输出，维度为1\*3\*特征图W\*特征图H\*(classnum+5), 因此需要对应onnx模型结构一致。

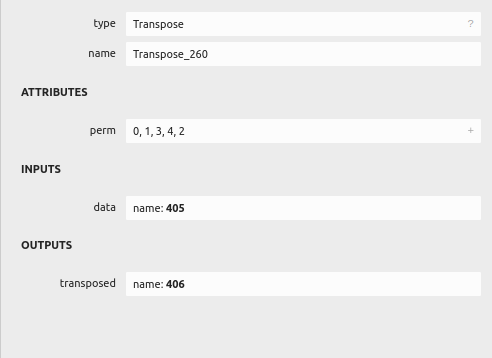
在浏览器中打开https://netron.app以查看onnx模型结构

Open onnx model within [***https://netron.app***](https://netron.app/)to see the model structure

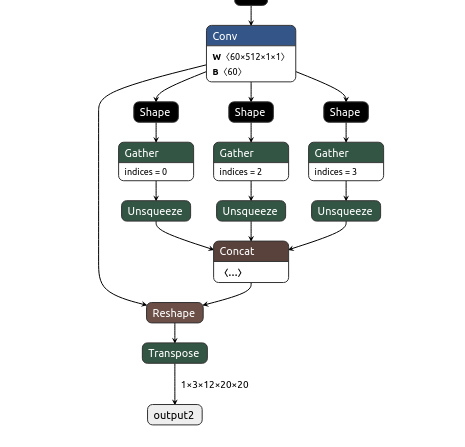


生成的onnx模型如上图，需要截取到3个transpose节点后进行输出。

在netron.app中点击3个transpose节点并查看其节点编号，输入数据编号。查看最后concat节点和其3个reshape节点的编号。

Check the index and input number of all the 3 transpose node,Check the index of last concat node, and 3 last reshape node.

使用工程中rknn-toolkit2-1.3.0/onnx\_change.py重构onnx模型。

 转换后的onnx模型结构（其中一个输出）如上图所示

Use ***rknn-toolkit2-1.3.0/onnx\_change.py*** to change the onnx structure and get a model with 3 outputs , looks like the pic above each.

**三、onnx模型转rknn模型**

使用rknn-toolkit2-1.3.0/examples/onnx/yolov5/test.py 进行模型转换注意target\_platform='rk3588' 不要改变。

Make sure the RKNN model is build for 3588 before using ***rknn-toolkit2-1.3.0/examples/onnx/yolov5/test.py*** to generate RKNN model.

**四、SDK的编译**

4.1 库的交叉编译（目前暂时不需要再次编译）

4.1 is In case one day the libs are lost again. Now there is no need to build the libs again.

需要百度并下载安卓ndk交叉编译器

**protubuf 需要使用3.17版本**（/opt/android-ndk-r17c为你的交叉编译器位置）**, 如果使用其他版本，除非进行一定量对应改动，否则SDK无法编译。**

下载解压后使用

*cmake \*

*-Dprotobuf\_BUILD\_SHARED\_LIBS=OFF \*

*-Dprotobuf\_BUILD\_STATIC\_LIBS=ON \*

*-DCMAKE\_VERBOSE\_MAKEFILE=ON \*

*-Dprotobuf\_BUILD\_TEST=OFF \*

*-Dprotobuf\_BUILD\_EXAMPLES=OFF \*

*-DCMAKE\_TOOLCHAIN\_FILE=/opt/android-ndk-r17c/build/cmake/android.toolchain.cmake \*

*-DCMAKE\_BUILD\_TYPE=Release \*

*-DCMAKE\_INSTALL\_PREFIX= \*

*-DANDROID\_NDK= \*

*-DANDROID\_TOOLCHAIN=clang \*

*-DANDROID\_ABI=arm64-v8a \*

*-DANDROID\_NATIVE\_API\_LEVEL=24 \*

*-DANDROID\_STL=c++\_shared \*

*-DANDROID\_LINKER\_FLAGS="-landroid -llog" \*

*-DANDROID\_CPP\_FEATURES="rtti exceptions" \*

*./cmake*

即可获得库和include文件

Use the code above to build protobuf lib. Version 3.17 of protobuf shall be used, otherwise you will get yourself into trouble of bugs.

opencv+ffmpeg+tbb 联合编译：

根据这篇博客的操作进行联合编译

https://blog.csdn.net/qq\_34231329/article/details/126953330

注意最终编译opencv时，命令中增加

*-DBUILD\_TBB=ON\*

*-DWITH\_TBB=ON\*

以使其联合编译TBB

Follow the blog above to build ffmpeg+opencv. Add the above two lines in the building command mentioned in the blog to build TBB lib with opencv.

4.2 libxiding.so 的编译

下载SDK工程后，bash build.sh （注意设置好自己的交叉编译器位置），即可得到 libxiding.so

注意要在CMakeLists.txt 里面链接所有之前编译出来的库。（目前不需要修改只需要保持）

Make sure every library you generate in step 4.1 is linked in CmakeLists.txt like the pic down. No need to change now.



4.3 C++版算法的调试

bash build.sh 同时还会生成 test\_video 和test\_pic 两个可执行文件，一般直接使用test\_video. 编译前要先修改test\_video.cc中的配置以选择不同实验。

Specify your experiment and your test videos in test/test\_video.cc before doing bash build.sh.



build.sh 会自动复制相关文件到板子上。

使用*adb shell* 进入板子，调试：

*cd /data/test*

*export LD\_LIBRARY\_PATH=./*

*./test\_video*

Use above lines for local video test.

注意：在进行本地调试需要将注释掉工程中analyzer/utils.cc 最后一行memcpy(output.score\_image, frame.data, frame.cols \* frame.rows \* frame.channels()) ; 保留倒数第二行cv::imwrite(score\_info + ".jpg", frame);

Comment the last line of **analyzer/utils.cc** and keep the second last line when do local testing. Do the opposite when generating lib.so for APK usage.

提供给APK方面时相反。

