Trt\_wheels 各API参数详解

1. Input

ITensor\* **trt\_input**(INetworkDefinition\* network, const char\* inputName,int C,int H,int W);

构建网络的起始层，数据从这个API输入到网络中

InputName: 输入层的名字，在推理阶段需要通过它找到输入数据的位置

E,g!

auto input = trt\_input(network,”data”,3, 640, 640);

1. Conv

ITensor\* **trt\_conv**(INetworkDefinition\* network, ITensor\* input,vector<float> wgt,vector<float> bias,int outputC,

int kernel = 1,int stride = 1,int padding = 0,

int group =1,int dilations = 1);

Wgt: conv层权值

Bias: conv层的偏置，当训练时设置bias = false，bias为空

outputC: 输出的channels

Kernel: 卷积核的大小，默认是kernel\_h = kernel\_w = kernel；如果kernel\_h != kernel\_w的情况可以根据代码修改即可

Stride: stride\_h = stride\_w = stride，同上理解

Padding: 同上理解

Group: 默认值是1，当不为1 的时候需要输入的通道数inputC % group == 0; 否则会错

Dilations: 默认是1，当不为1的时候，通常情况下padding == dilations

E,g!

auto conv = trt\_conv(network , input , wgt, bias, 64 ,7,2,3)

1. Pool

ITensor\* **trt\_pool**(INetworkDefinition\* network, ITensor\*input, PoolingType pType,

int kernel = 1,int stride = 1,int padding = 0);

enum class **PoolingType** : int

{

kMAX = 0, // 最大池化

kAVERAGE = 1, // 平均池化

kMAX\_AVERAGE\_BLEND = 2

};

E,g:

Auto pool = trt\_pool(network, input, PoolingType::kMAX, 2,2);

1. batchNorm

ITensor\* **trt\_batchNorm**(INetworkDefinition\* network, ITensor\*input, vector<float> alpha, vector<float> gamma,vector<float> mean,vector<float> var, float eps = 1.0e-5);

tensorRT中本身并没有类似addBatchsize()这样的API，需要借助Scale层实现。Scale算法公式如下：

y= (x \* scale + shift)^power;

batchsize算法公式如下：

y = ((x-mean)/sqrt(var+1e-5))\*w + b

= x\*w/sqrt(var+1e-5) +(b – (mean\*w)/sqrt(var+1e-5))

所以通过变换可以得到：

scale = w/sqrt(var+1e-5)

shift = b – (mean\*w)/sqrt(var+1e-5)

power = 1.0

注意：在pytorch中eps默认等于1.0e-5,在其他神经网络框架中并不一定，请注意观察

E,g:

Auto bn = trt\_batchNorm(network , input, alpha, gamma, mean, var);

五：activation

ITensor\* **trt\_activ**(INetworkDefinition\* network, ITensor\* input,ActivationType Type,float alpha = 0.0,float beta = 0.0);

激活层，包括很多的激活类型，有些需要设置alpha和beta的值。

E,g:

Auto Lrelu =trt\_activ(network, input, ActivationType::kLEAKY\_RELU,0.1);

六：FC( == linear)

ITensor\* **trt\_FC**(INetworkDefinition\* network, ITensor\* input,

int outputC, vector<float> wgt, vector<float> bias);

全连接层

E,g:

Auto fc = trt\_FC(network, input,1000,wgt,bias);

七：elementWise(sum/sub/prod/div/power...)

ITensor\* **trt\_elt**(INetworkDefinition\* network, ITensor\* input1, ITensor\* input2, ElementWiseOperation Type);

enum class **ElementWiseOperation** : int

{

kSUM = 0, //!< Sum of the two elements.

kPROD = 1, //!< Product of the two elements.

kMAX = 2, //!< Maximum of the two elements.

kMIN = 3, //!< Minimum of the two elements.

kSUB = 4, //!< Substract the second element from the first.

kDIV = 5, //!< Divide the first element by the second.

kPOW = 6, //!< The first element to the power of the second element.

kFLOOR\_DIV = 7,//!< Floor division of the first element by the second.

kAND = 8, //!< Logical AND of two elements.

kOR = 9, //!< Logical OR of two elements.

kXOR = 10, //!< Logical XOR of two elements.

kEQUAL = 11, //!< Check if two elements are equal.

kGREATER = 12, //!< Check if element in first tensor is greater than corresponding element in second tensor.

kLESS = 13 //!< Check if element in first tensor is less than corresponding element in second tensor.

};

有些**ElementWiseOperation** 需要注意input1 和input2，如果放错位置会得到错误的结果，例如：kSUB/dDIV等，input1 - input2 和 input2 -input1

E,g:

Auto elt = trt\_elt(network,input1,input2,**ElementWiseOperation::kSUM);**

八：concat

ITensor\* **trt\_concat**(INetworkDefinition\* network,vector<ITensor\*>

inputs, int axis = 0);

将多个input沿axis通道拼接起来，注意：当拼接的时候，所有输入的其他通道的结果要相等，否则会出错

E,g:

Auto cat = trt\_concat(network, inputs, axis);

九：slice（切分）

ITensor\* **trt\_slice**(INetworkDefinition\* network , ITensor\* input, Dims start,Dims size,Dims step);

Start: 切分的起始位置

Size: 切分后的输出尺寸

Step: 每个维度切分的步长

Eg:

Auto slice = trt\_slice(network,input,dims3{0,0,0},dims3{1,640,640},

dims3{1,1,1});

十：softmax

ITensor\* **trt\_softmax**(INetworkDefinition\* network, ITensor\* input);

E,g:

Auto softmax = trt\_softmax(network,input);

十一：matmul

ITensor\* **trt\_MatMul**(INetworkDefinition\* network, ITensor\* input1, bool transpose1,ITensor\* input2,bool transpose2);

矩阵相乘，注意input1和input2不能交换位置,且input1(transpose1)的列要等于input2(transpose2)的行

bool transpose1: 是否将input1转置，如果等于True则需要转置

bool transpose2：是否将input2转置，。。。

E,g:

Auto mm = trt\_MatMul(network,input1,false,input2,false)

十二：shuffle\_reshape( just reshape)

ITensor\* **trt\_shuffle\_reshape**(INetworkDefinition\* network,

ITensor\* input,Dims dims);

对input进行reshape操作,dims的所有值的乘积必须和input的所有通道数乘积相等，也就是只改变形状不改变大小

Eg:

Auto reshape = trt\_shuffle\_reshape(network,input,dims);

十三：shuffle\_permute( just permute)

ITensor\* **trt\_shuffle\_premute**(INetworkDefinition\* network, ITensor\* input, Permutation pmt);

对输入进行通道转置，例如输入是三通道，仅交换0，1通道：

Permutation Pmt ;

Pmt.order[0] = 1;

Pmt.order[1] = 0;

Pmt.order[2] = 2;

Eg:

Auto premute = trt\_shuffle\_premute(network,input,pmt);

十四：shuffle ( reshape and permute)

ITensor\* **trt\_shuffle**(INetworkDefinition\* network, ITensor\* input, Dims dims,Permutation pmt, bool Reshape\_first);

既交换通道又进行转置，需要注意的是当bool Reshape\_first设置为true的时候先进行reshape，之后做permute,反之则先permute,然后reshape

十五：reduce

ITensor\* **trt\_reduce**(INetworkDefinition\* network, ITensor\* input, uint32\_t reduceAxes,ReduceOperation op,bool keepDims = false);

\*reduceAxes W H C

\* 0X01 0 0 1

\* 0X02 0 1 0

\* 0X03 0 1 1

\* 0X04 1 0 0

\* 0X05 1 0 1

\* 0X06 1 1 0

\* 0X07 1 1 1

在等于1的所有通道上做reduceOperation操作

enum class **ReduceOperation** : int

{

kSUM = 0, // +

kPROD = 1, // X

kMAX = 2, // 取最大值

kMIN = 3, //取最小值

kAVG = 4 // 取平均值

};

bool keepDims ：结果是否保留等于1的维度，如果设置为true，则保留；否则不保留。例如输出通道是{64，1，1}，如果设置为true,则输出通道是{64}

例如：AdaptiveMaxPool2d((1,1))

Auto adaptPool = trt\_reduce(network,input,0x06,ReduceOperation::kMAX);

十六：topk (argmax/argmin)

ITensor\* **trt\_topK**(INetworkDefinition\* network, ITensor\* input,

int k = 1, TopKOperation op = TopKOperation::kMAX ,

uint32\_t reduceAxes = 0x01,int outputAxes = 0);

可以借助topk层实现argmax/argmin操作，即按顺序输出

K: 输出前多少个

Op: 输出的类型

reduceAxes:类似reduce层的操作，不过仅仅可以在一个通道上输出

outputAxes: 当等于0的时候输出排序后的结果，

等于1的时候输出排序后的index

Eg:

Auto argmax3 = trt\_topK(network,input,3,TopKOperation::kMAX ,0x01,0);

十七：upsample (nerest/bilinear)

ITensor\* **trt\_upsample**(INetworkDefinition\* network, ITensor\* input, ResizeMode mode, Dims outdims);

Mode：上采用的类型，目前tensorRT的官方支持nereast和bilinear的采样方式，但是当使用bilinear的采样时候，只支持ali\_corners = true的情况

Outdims: 采样后输出的通道

Eg:

Auto up = trt\_upsample(network,input,ResizeMode::kNEAREST,Dims3{1,20,20});

十八：constant ("The most important layer!!!")

ITensor\* **trt\_constant**(INetworkDefinition\* network, Dims dims, vector<float> wgt);

个人认为最重要的API，这个API的主要作用是添加一个tensor\*到整个network中，可以理解为向网络中添加了一个layer

Dims: 输出的维度

Wgt：添加的数据，也是这个API输出的tensor

当遇到output = alpha \* input的情况下，在tensorRT中是无法直接相乘的，需要把alpha添加到网络中才可以直接参与运算，这个时候就可以用这个API把alpha添加进去，例如dims = Dims3{2,10,10}

For(int i = 0 ;i < 2\*10\*10; i++)

wgt.at[i] = alpha;

Auto add\_alpha = trt\_constant(network,dims,wgt);

灵活运用这个API，可以间件实现很多的原来trt中没有的操作

十九：PReLU

ITensor\* **trt\_PReLU**(INetworkDefinition\* network,

ITensor\* input,vector<float> wgt);

Auto prelu = trt\_PReLU(network,input, wgt);

二十：groupNorm

ITensor\* **trt\_groupNorm**(INetworkDefinition\* network,ITensor\* input, int groups,vector<float> wgt,vector<float> bias,float eps = 1e-5);

需要注意inputC % groups == 0,否则会出错

二十一： padding

ITensor\* **trt\_padding**(INetworkDefinition\* network,ITensor\* input,DimsHW prePad = DimsHW{0,0},DimsHW postPad = DimsHW{0,0});

prePad: 上和左填充

postPad: 下和右填充