Pytorch网络结构可视化方法汇总（三种实现方法详解）

前言：在深度学习的时候，能够很好的绘制出整个网络的模型图，这是非常方便的，当前流行的tensorflow和pytorch都有很多种方法可以绘制模型结构图，特在此总结如下：

tensorflow的模型结构可视化方法：

（1）使用自带的tensorboard（不直观）

（2）使用netron工具打开（.pd 或者是.meta文件）

（3）第三方库CNNGraph（ https://github.com/huachao1001/CNNGraph）

（4）tensorspace.js （这个比较高级，没用过）

（5）高层API中keras的可视化

pytorch的模型结构可视化方法：

（1）使用tensorboardX（不太直观）

（2）使用graphviz加上torchviz （依赖于graphviz和GitHub第三方库torchviz）

（3）使用微软的tensorwatch （只能在jupyter notebook中使用，**个人最喜欢这种方式**）

（4）使用netron可视化工具（.pt 或者是 .pth 文件）

一、使用TENSORBOARDX

使用tensorboardX必须要安装tensorboard才行，可能会存在一些版本的匹配问题，下面的版本是亲测有效的。

pytorch0.4.1+tensorboard1.7.0+tensorboardX1.4

pytorch1.0.1+tensorboard1.14.0+tensorboardX1.8

上面这两个都是可行的。

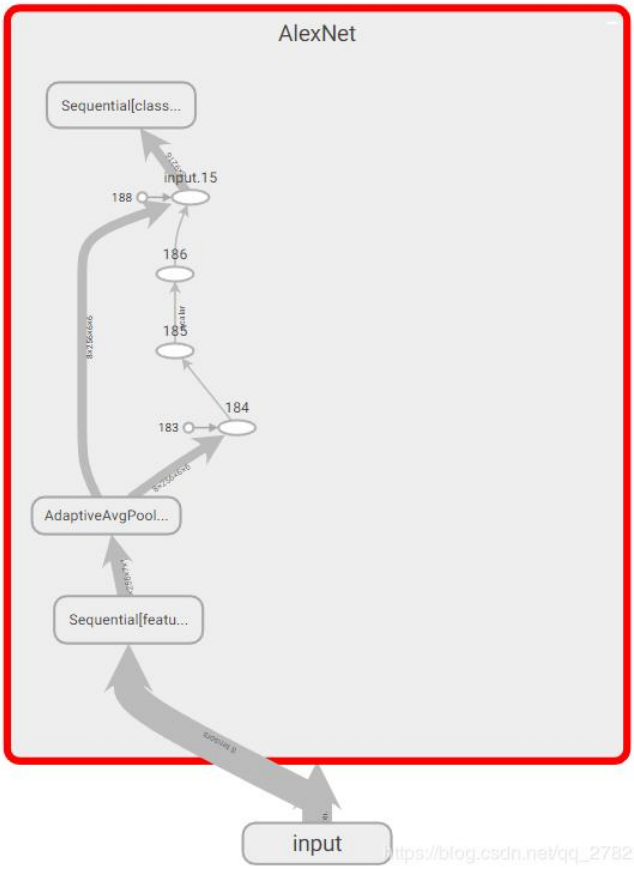
本文以AlexNet为例，鉴于torchvision已经实现了AlexNet模型，就不再自己编写，直接导入即可。

```
import torch
from torchvision.models import AlexNet

from tensorboardX import SummaryWriter


x=torch.rand(8,3,256,512)
model=AlexNet()

with SummaryWriter(comment='AlexNet') as w:
w.add_graph(model, x) # 这其实和tensorflow里面的summarywriter是一样的。
```

上面的代码运行结束后，会在当前目录生成一个叫runs的文件夹，runs文件夹里面会有一个文件夹Jul22_18-03-19_WH-PC19012AlexNet，里面存储了可视化所需要的日志信息。用cmd进入到runs文件夹所在的目录中（路劲中不能有中文），然后cmd中输入：

```
tensorboard --logdir Jul22_18-03-19_WH-PC19012AlexNet
```

AlexNet的效果图如下所示：



当然这里的节点可以打开进行查看，也可以放大缩小。

二、使用GRAPHVIZ+TORCHVIZ来可视化模型

首先安装这两个依赖包：

```
pip install graphviz # 安装graphviz
pip install git+https://github.com/szagoruyko/pytorchviz # 通过git安装torchviz
```

第一步：加载并运行一个模型

```
import torch
from torchvision.models import AlexNet
from torchviz import make_dot

x=torch.rand(8,3,256,512)
model=AlexNet()
y=model(x)
```

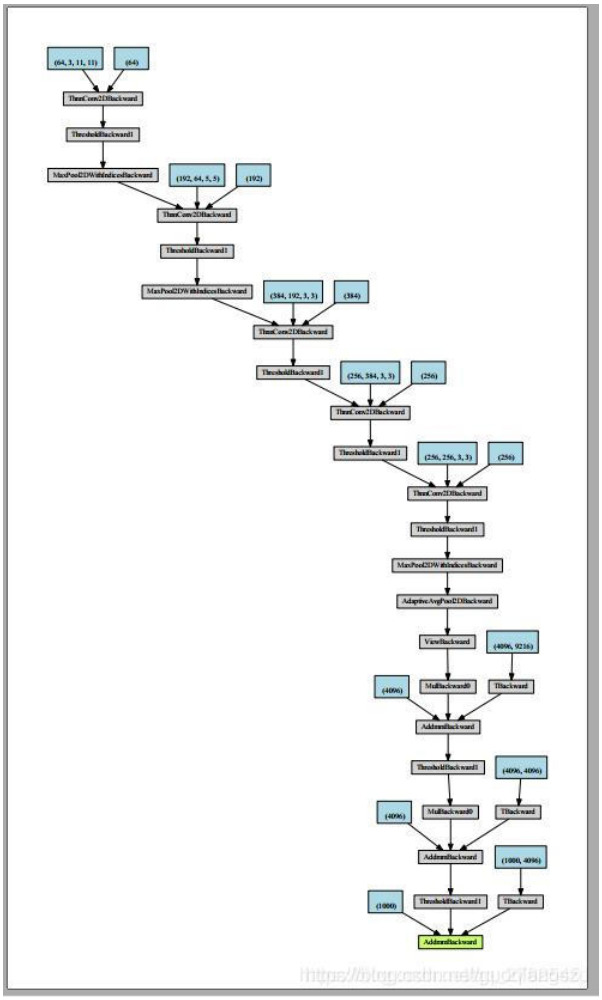
第二步：调用make_dot()函数构造图对象

```
# 这三种方式都可以
g = make_dot(y)
# g=make_dot(y, params=dict(model.named_parameters()))
#g = make_dot(y, params=dict(list(model.named_parameters()) + [('x', x)]))
```

第三步：保存模型，以PDF格式保存

```
# 这两种方法都可以
# g.view() # 会生成一个 Digraph.gv.pdf 的PDF文件
g.render('espnet_model', view=False) # 会自动保存为一个 espnet.pdf，第二个参数为True,则会自动打开该PDF文件，为False则不打开
```

模型的结构如下：



另外，我还可以查询整个模型的参数量信息，代码如下：

```
# 查看模型的参数信息

params = list(model.parameters())
k = 0
for i in params:
    l = 1
    print("该层的结构：" + str(list(i.size())))
    for j in i.size():
        l *= j
    print("该层参数和：" + str(l))
    k = k + l
```

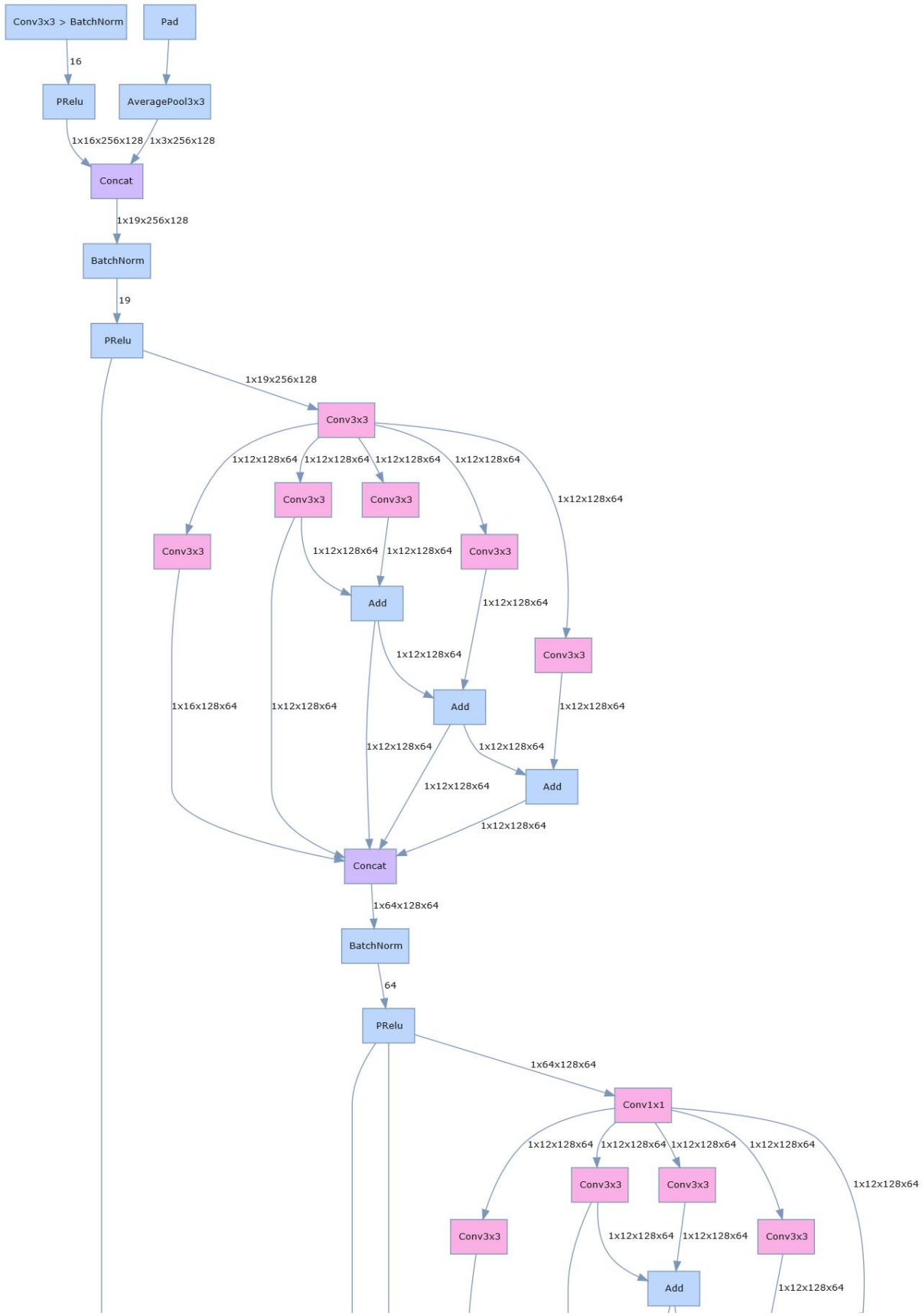
```
print("总参数数量和：" + str(k))
'''
该层的结构：[64, 3, 11, 11]
该层参数和：23232
该层的结构：[64]
该层参数和：64
该层的结构：[192, 64, 5, 5]
该层参数和：307200
该层的结构：[192]
该层参数和：192
该层的结构：[384, 192, 3, 3]
该层参数和：663552
该层的结构：[384]
该层参数和：384
该层的结构：[256, 384, 3, 3]
该层参数和：884736
该层的结构：[256]
该层参数和：256
该层的结构：[256, 256, 3, 3]
该层参数和：589824
该层的结构：[256]
该层参数和：256
该层的结构：[4096, 9216]
该层参数和：37748736
该层的结构：[4096]
该层参数和：4096
该层的结构：[4096, 4096]
该层参数和：16777216
该层的结构：[4096]
该层参数和：4096
该层的结构：[1000, 4096]
该层参数和：4096000
该层的结构：[1000]
该层参数和：1000
总参数数量和：61100840
'''
```

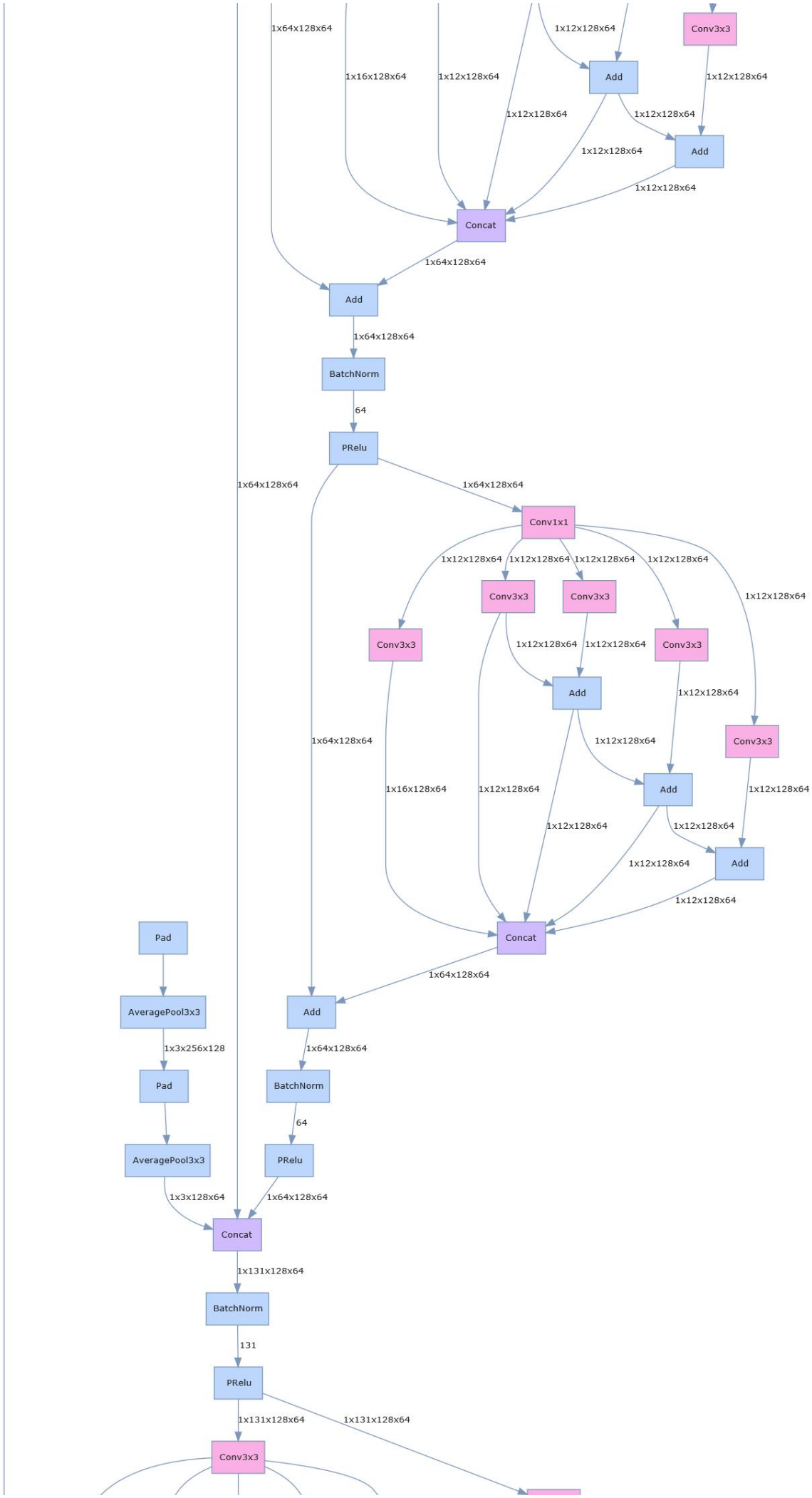
三、通过tensorwatch+jupyter notebook来实现

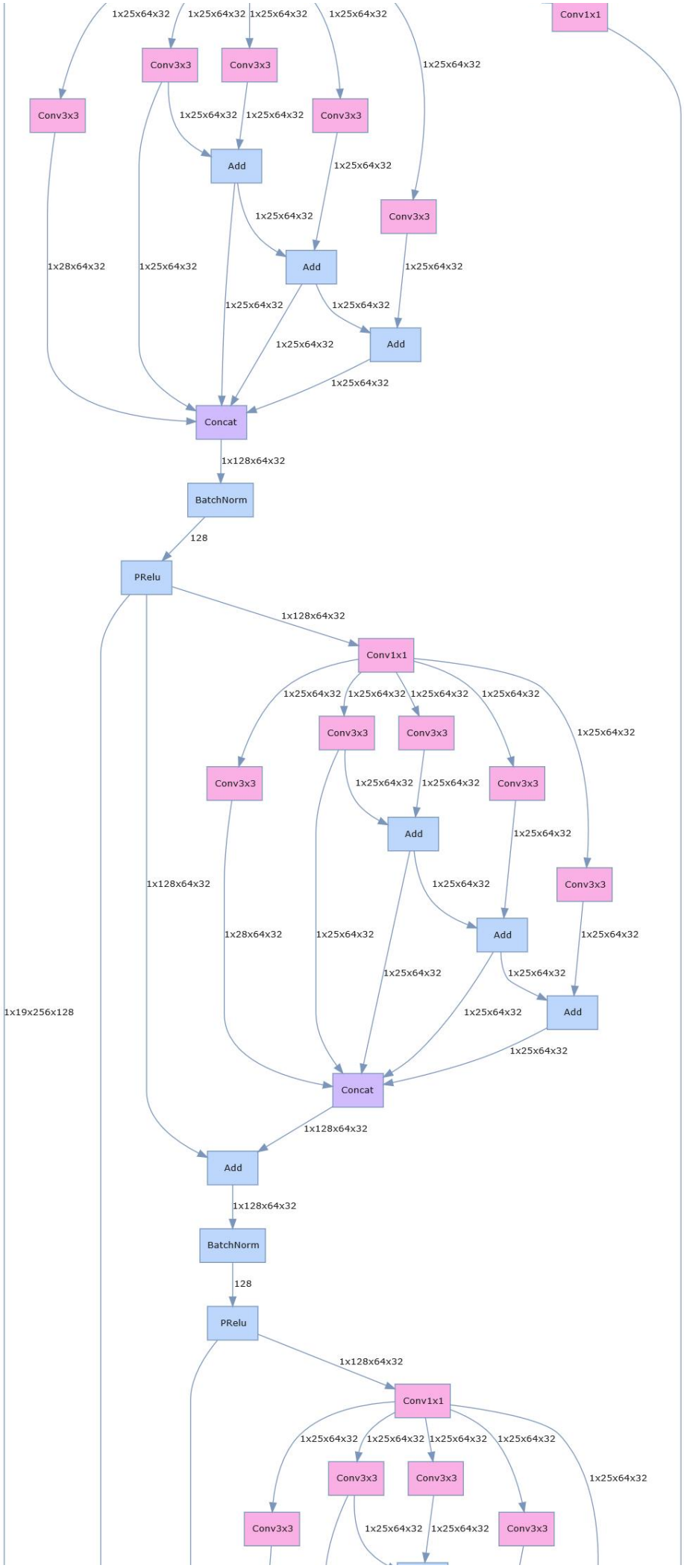
```

import torch
import tensorwatch as tw
```

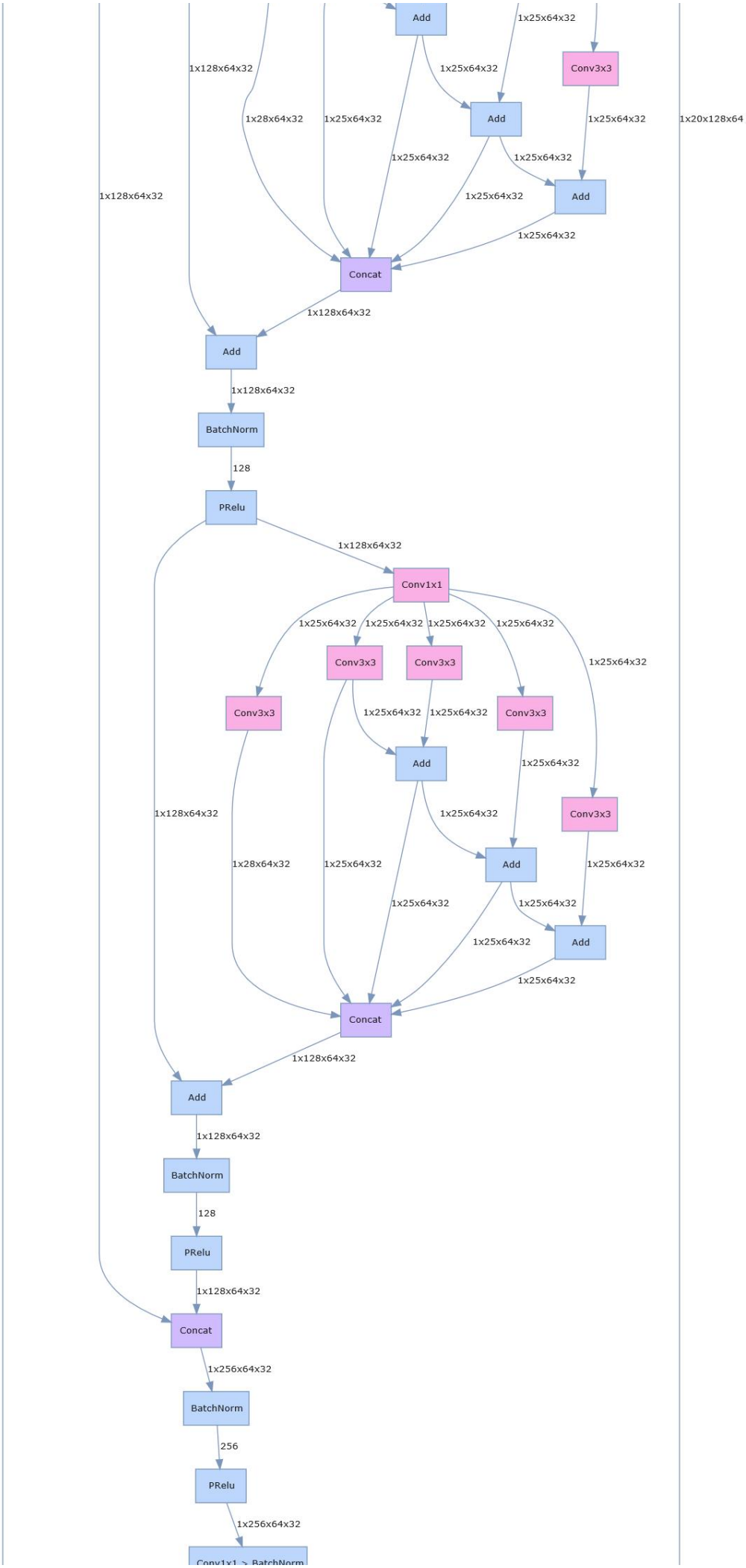
```
5. from lanenet_model.blocks import ESPNet_Encoder # 这是我自己定义的一个网络
6.
7. # 其实就两句话
8. model=ESPNet_Encoder()
9. tw.draw_model(model, [1, 3, 512, 256])
```

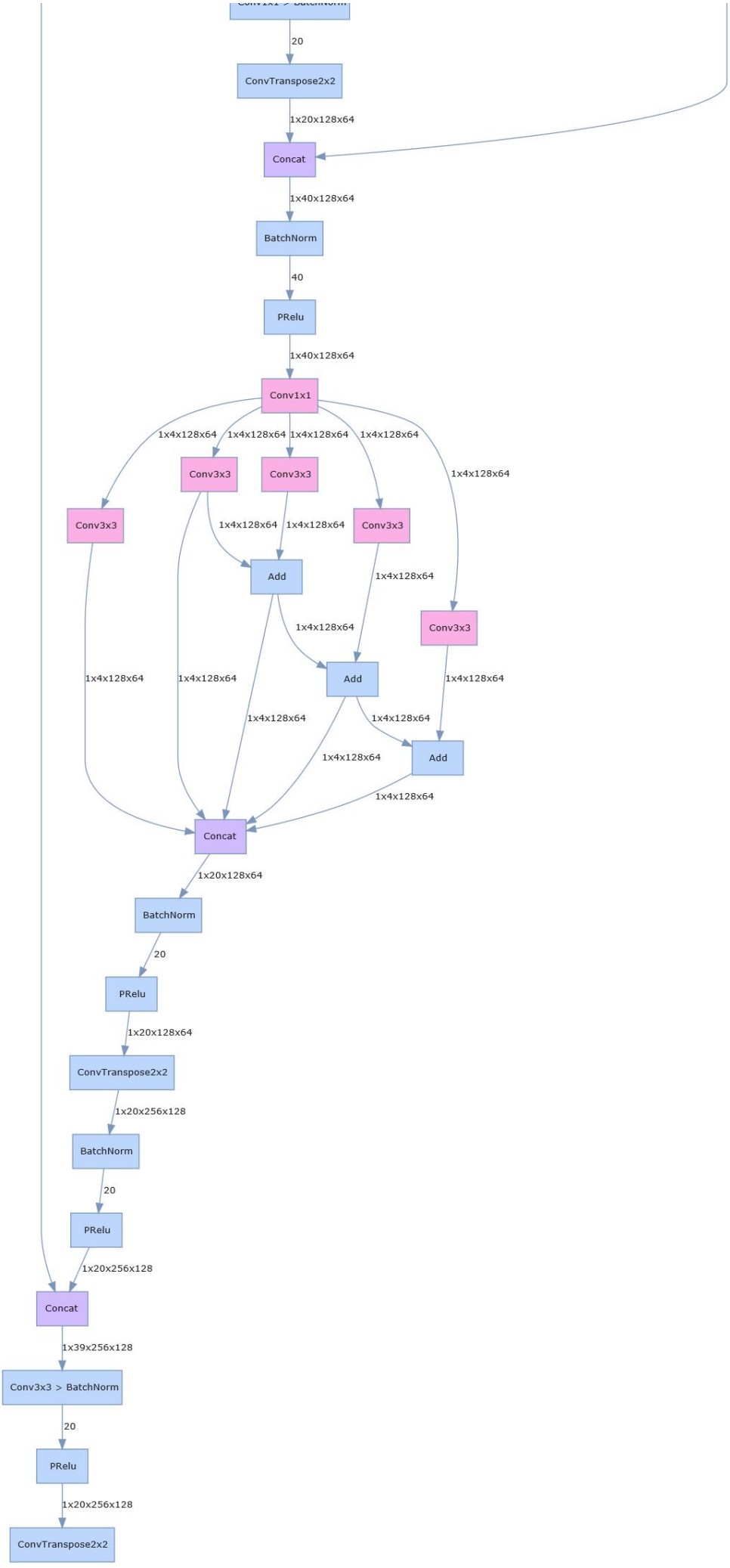
网络结构如下:




1x64x128x64
1x12x128x64
Conv3x3
1x16x128x64
1x12x128x64
Add
1x12x128x64
1x12x128x64
1x12x128x64
Add
1x12x128x64
Concat
1x64x128x64
Add
1x64x128x64
BatchNorm
64
PRelu
1x64x128x64
1x64x128x64
Conv1x1
1x12x128x64
1x12x128x64
1x12x128x64
1x12x128x64
1x12x128x64
Conv3x3
Conv3x3
Conv3x3
1x12x128x64
1x12x128x64
Conv3x3
Add
1x12x128x64
1x64x128x64
1x12x128x64
Conv3x3
1x16x128x64
1x12x128x64
Add
1x12x128x64
1x12x128x64
1x12x128x64
Add
1x12x128x64
Concat
Pad
1x64x128x64
AveragePool3x3
Add
1x3x256x128
1x64x128x64
Pad
BatchNorm
64
AveragePool3x3
PRelu
1x3x128x64
1x64x128x64
Concat
1x131x128x64
BatchNorm
131
PRelu
1x131x128x64
1x131x128x64
Conv3x3


1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
Conv1x1
Conv3x3
Conv3x3
1x25x64x32
Conv3x3
1x25x64x32
1x25x64x32
Conv3x3
Add
1x25x64x32
1x25x64x32
Conv3x3
Add
1x28x64x32
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
Add
1x25x64x32
Concat
1x128x64x32
BatchNorm
128
PRelu
1x128x64x32
Conv1x1
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
Conv3x3
Conv3x3
Conv3x3
1x25x64x32
1x25x64x32
Conv3x3
Add
1x25x64x32
1x128x64x32
Conv3x3
1x28x64x32
1x25x64x32
Add
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
Add
1x19x256x128
1x25x64x32
Concat
1x128x64x32
Add
1x128x64x32
BatchNorm
128
PRelu
1x128x64x32
Conv1x1
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
Conv3x3
Conv3x3
Conv3x3
1x25x64x32
1x25x64x32
Conv3x3


Add
1x25x64x32
Conv3x3
1x128x64x32
1x25x64x32
Add
1x28x64x32
1x25x64x32
1x25x64x32
1x25x64x32
1x20x128x64
1x128x64x32
1x25x64x32
Add
1x25x64x32
Concat
1x128x64x32
Add
1x128x64x32
BatchNorm
128
PRelu
1x128x64x32
Conv1x1
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
1x25x64x32
Conv3x3
Conv3x3
Conv3x3
1x25x64x32
1x25x64x32
Conv3x3
Add
1x25x64x32
1x128x64x32
1x25x64x32
Conv3x3
1x28x64x32
1x25x64x32
Add
1x25x64x32
1x25x64x32
1x25x64x32
Add
1x25x64x32
Concat
1x128x64x32
Add
1x128x64x32
BatchNorm
128
PRelu
1x128x64x32
Concat
1x256x64x32
BatchNorm
256
PRelu
1x256x64x32
Conv1x1 > BatchNorm


20
ConvTranspose2x2
1x20x128x64
Concat
1x40x128x64
BatchNorm
40
PRelu
1x40x128x64
Conv1x1
1x4x128x64
1x4x128x64
1x4x128x64
1x4x128x64
1x4x128x64
Conv3x3
Conv3x3
Conv3x3
Conv3x3
1x4x128x64
1x4x128x64
Add
1x4x128x64
1x4x128x64
Conv3x3
1x4x128x64
1x4x128x64
Add
1x4x128x64
1x4x128x64
1x4x128x64
Add
1x4x128x64
Concat
1x20x128x64
BatchNorm
20
PRelu
1x20x128x64
ConvTranspose2x2
1x20x256x128
BatchNorm
20
PRelu
1x20x256x128
Concat
1x39x256x128
Conv3x3 > BatchNorm
20
PRelu
1x20x256x128
ConvTranspose2x2