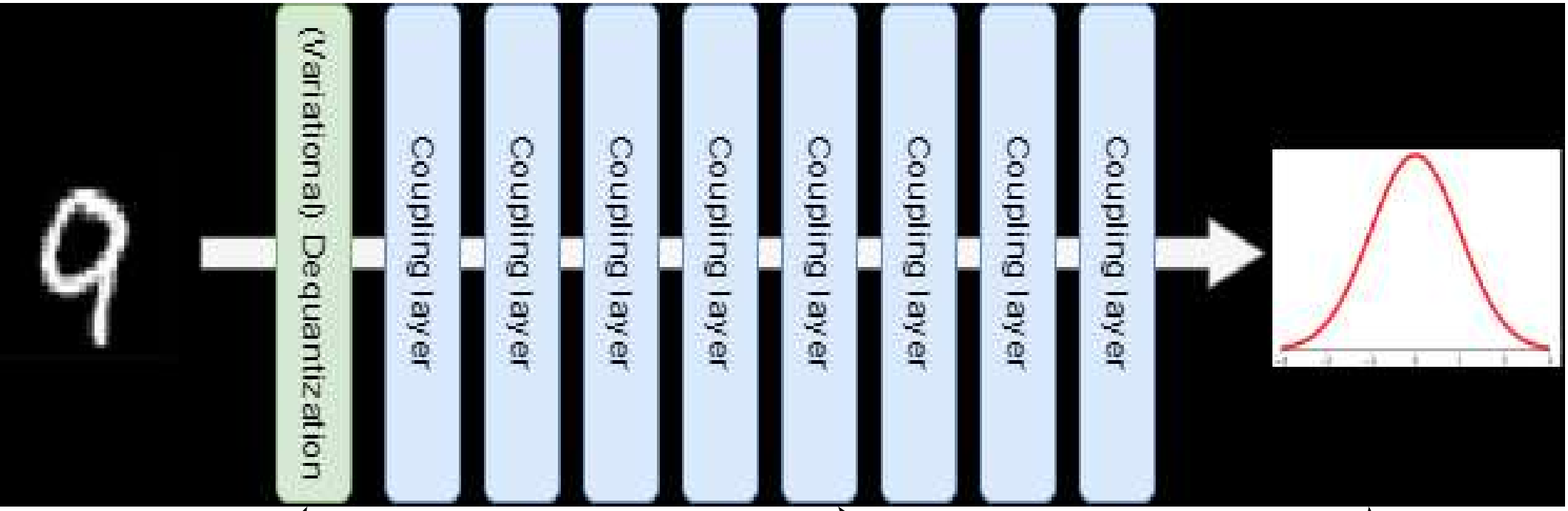
セル16のcreate_simple_flow
これはセル17のtrain_flowの引数となる。
train_flowでは画像が128個のバッチでやってくる。



フラグによって、Variationalかどうかを決める。
VariationalDequantizationはセル10
Dequantizationはセル6

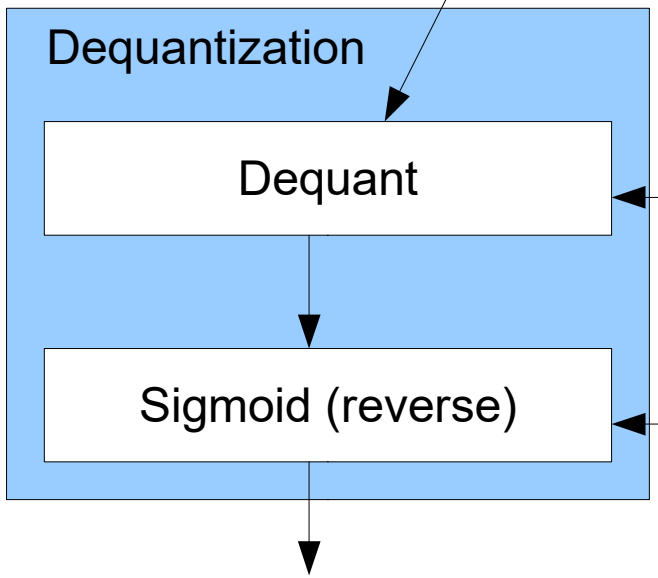
この関数中でfor文で8回実施。
Coupling layerはセル11

最後にImageFlowでまとめる。

このフローでは次元は変わらない。
MNISTの次元はチャンネル1、高さ28、幅28

セル6のDequantization

forward関数の正方向の流れを見る。



(0,1)のノイズを追加し、0から1にする。
ldjは-ln (quants)*zのサイズを足す。(z/quantsが影響)

[0,1]をRに単調な関数でマッピングしたい。
そのためにsigmoidの逆関数を利用する。
Sigmoidの逆関数の定義域は(0,1)なので、スケーリングして、
(0,1)とする。
z = z * (1 - self.alpha) + 0.5 * self.alpha
これによるldjの変化は以下。
ldj += np.log(1 - self.alpha) * np.prod(z.shape[1:])

次にシグモイド関数の逆関数を適用する。
ただ、ldjの変化は適用前のzが必要。
ldj += (-torch.log(z) – torch.log(1-z)).sum(dim=[1,2,3])
逆関数は
z = torch.log(z) - torch.log(1-z)

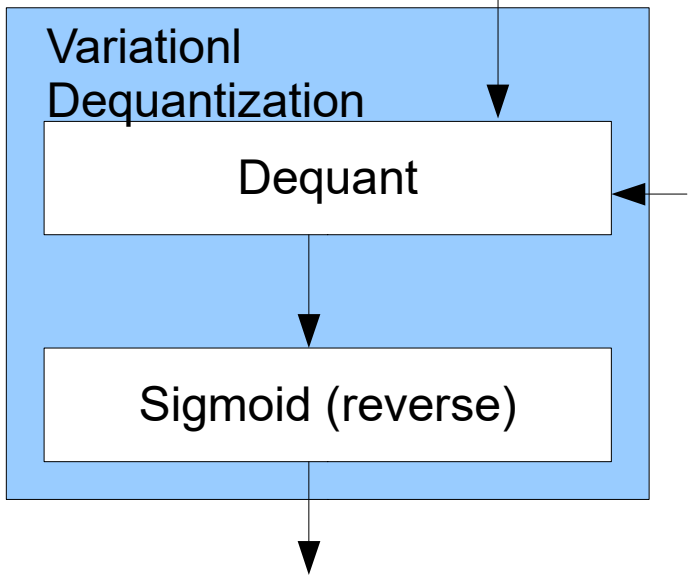
*ここに関してはフローは1つしかできない。以下、pythonの出力。
(0): Dequantization()

セル10のVariationalDequantization

```
vardeq_layers = [CouplingLayer(network=GatedConvNet(c_in=2, c_out=2, c_hidden=16),
        mask=create_checkerboard_mask(h=28, w=28, invert=(i%2==1)),c_in=1) for i in range(4)]
```

forward関数の正方向の流れを見る。
Dequantizationを継承しているので、基本は同じ。

z(バッチ128*チャンネル1*高さ28*幅28)



この子が異なる
ここで、一様分布をflowで変形する。
たちまち、flowはCouplingLayer、4個のみ。
平均と分散を出力する想定?
分散がldjに相当すると考える?
ネットワークに入れるときはもとの画像も含めて入れるので、
チャンネル2として、ldjも出力したいのでチャンネル2
ここをうまく対応するために、GatedConvNetでorig_imgがある

z(バッチ128*チャンネル1*高さ28*幅28)

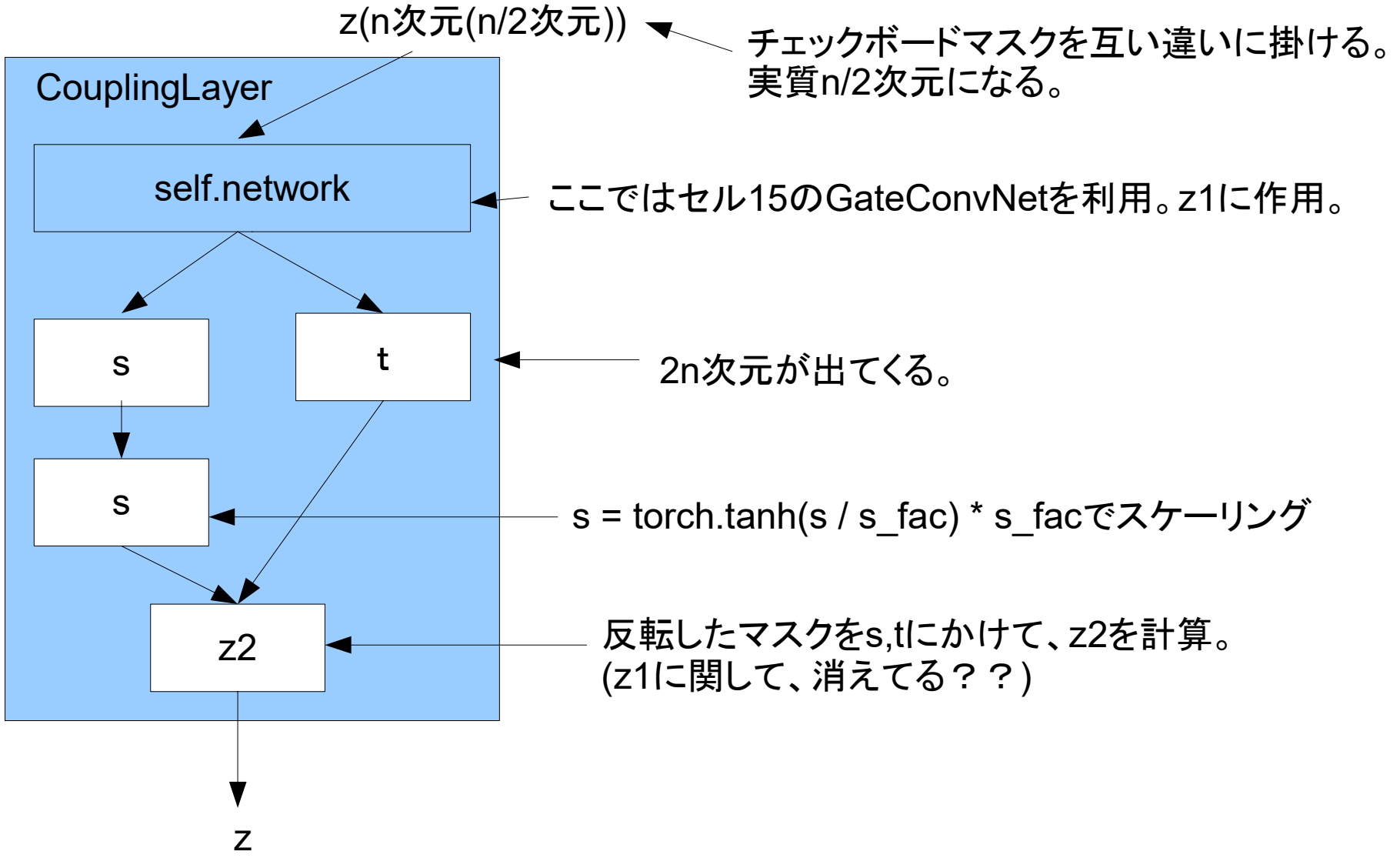
VariationalDequantizationの構造に関する、pythonの出力。(一部省略)

```
(0): VariationalDequantization(
    (flows): ModuleList(
      (0): CouplingLayer(
        (network): GatedConvNet(
          (nn): Sequential(
            (0): Conv2d(2, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): GatedConv(
              (net): Sequential(
                (0): ConcatELU()
                (1): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (2): ConcatELU()
                (3): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1))
              )
            )
            (2): LayerNormChannels()
            (3): GatedConv(
            )
            (4): LayerNormChannels()
            (5): GatedConv(
            )
            (6): LayerNormChannels()
            (7): ConcatELU()
            (8): Conv2d(32, 2, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          )
        )
      )
      (1): CouplingLayer(
      )
      (2): CouplingLayer(
      )
      (3): CouplingLayer(
      )
    )
  )
```

セル11のCouplingLayer

```
flow_layers += [CouplingLayer(network=GatedConvNet(c_in=1, c_hidden=32),
                              mask=create_checkerboard_mask(h=28, w=28, invert=(i%2==1)),
                              c_in=1)]
```

forward関数の正方向の流れを見る。

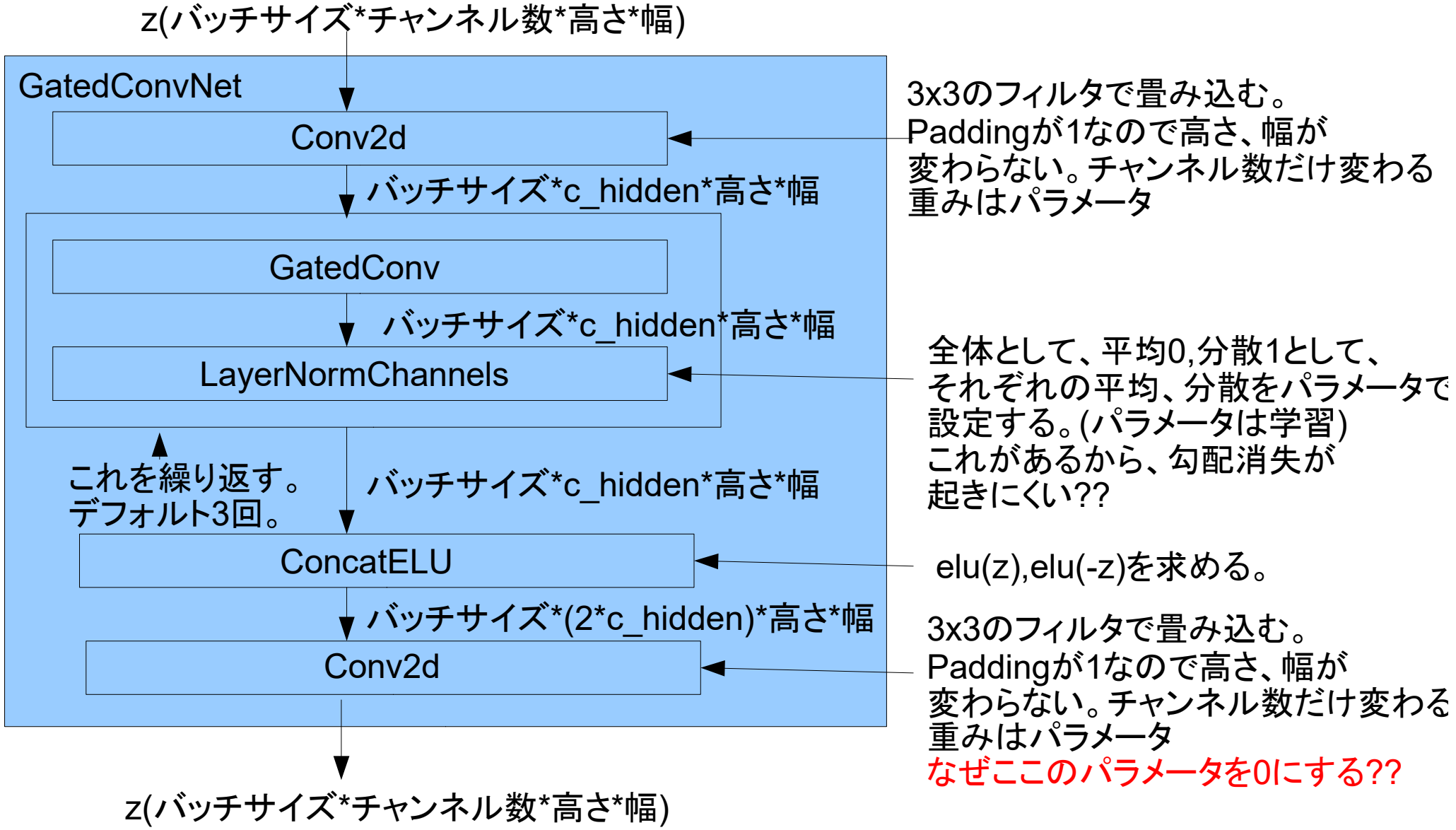


CouplingLayerの構造に関する、pythonの出力。

```
(1): CouplingLayer(
    (network): GatedConvNet(
      (nn): Sequential(
        (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): GatedConv(
          (net): Sequential(
            (0): ConcatELU()
            (1): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (2): ConcatELU()
            (3): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
          )
        )
        (2): LayerNormChannels()
        (3): GatedConv(
          (net): Sequential(
            (0): ConcatELU()
            (1): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (2): ConcatELU()
            (3): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
          )
        )
        (4): LayerNormChannels()
        (5): GatedConv(
          (net): Sequential(
            (0): ConcatELU()
            (1): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (2): ConcatELU()
            (3): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
          )
        )
        (6): LayerNormChannels()
        (7): ConcatELU()
        (8): Conv2d(64, 2, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
  )
```

セル15のGatedConvNet (各種ネットワークがこれになっている。)

forward関数を見れば良いが実際はコンストラクタで、ネットワークを作っている。



セル15のGatedConv

forward関数を見る。