```
import torch
import torch.nn as nn

def weights_init(w):
    """
    Initializes the weights of the layer, w.
    """
    classname = w.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(w.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(w.weight.data, 1.0, 0.02)
        nn.init.constant_(w.bias.data, 0)

#Define the generator
class Generator(nn.Module):
    def __init__(self, nz, nc, ngf, ngpu, size = 64):

        super(Generator, self).__init__()
        self.ngpu = ngpu
        if size == 64:
       # 64 x 64
            self.main = nn.Sequential(
                # input is Z, going into a convolution
                nn.ConvTranspose3d(nz, ngf * 8, 4, 1, 0, bias=False),
                nn.BatchNorm3d(ngf * 8),
                nn.ReLU(True),
                # state size. (ngf*8) x 4 x 4
                nn.ConvTranspose3d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
                nn.BatchNorm3d(ngf * 4),
                nn.ReLU(True),
                # state size. (ngf*4) x 8 x 8
                nn.ConvTranspose3d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
                nn.BatchNorm3d(ngf * 2),
                nn.ReLU(True),
                # state size. (ngf*2) x 16 x 16
                nn.ConvTranspose3d(ngf * 2, ngf, 4, 2, 1, bias=False),
                nn.BatchNorm3d(ngf),
                nn.ReLU(True),
                # state size. (ngf) x 32 x 32

                nn.ConvTranspose3d(ngf, nc, 4, 2, 1, bias=False),
                #nn.Tanh()
                nn.Softmax(dim=1)
                # state size. (nc) x 64 x 64
            )
        elif size == 128:
# 128 x 128

            self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose3d(nz, ngf * 2, 4, 1, 0, bias=False),
            nn.BatchNorm3d(ngf * 2),
            nn.ReLU(True),
```

```
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose3d(ngf * 2, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose3d(ngf * 2, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose3d(ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm3d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose3d(ngf, ngf, 4, 2, 1, bias=False),
            #nn.Tanh()
            #nn.Softmax(dim=1)
            nn.BatchNorm3d(ngf),
            nn.ReLU(True),


            nn.ConvTranspose3d(ngf, nc, 4, 2, 1, bias=False),
            #nn.Tanh()
            nn.Softmax(dim=1)
            # state size. (nc) x 64 x 64
        )



        # 32 x 32
        elif size == 32:
            super(Generator, self).__init__()
            self.ngpu = ngpu
            self.main = nn.Sequential(
                # input is Z, going into a convolution
                nn.ConvTranspose3d(nz, ngf * 8, 4, 1, 0, bias=False),
                nn.BatchNorm3d(ngf * 8),
                nn.ReLU(True),
                # state size. (ngf*8) x 4 x 4
                nn.ConvTranspose3d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
                nn.BatchNorm3d(ngf * 4),
                nn.ReLU(True),
                # state size. (ngf*4) x 8 x 8
                nn.ConvTranspose3d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
                nn.BatchNorm3d(ngf * 2),
                nn.ReLU(True),
                # state size. (ngf*2) x 16 x 16
                nn.ConvTranspose3d(ngf * 2, nc, 4, 2, 1, bias=False),
                #nn.BatchNorm3d(ngf),
                #nn.ReLU(True),
                # state size. (ngf) x 32 x 32
            # nn.ConvTranspose3d(ngf, nc, 4, 2, 1, bias=False),
                #nn.Tanh()
                nn.Softmax(dim=1)
                #nn.Sigmoid()
                # state size. (nc) x 64 x 64
```

```
110             )
111
112     def forward(self, input):
113         # breakpoint()
114         return self.main(input)
115
116 #Define the discriminator
117 class Discriminator(nn.Module):
118     def __init__(self, nz, nc, ndf, ngpu, size = 64):
119         #breakpoint()
120         super(Discriminator, self).__init__()
121         self.ngpu = ngpu
122         if size == 64:
123             self.main = nn.Sequential(
124                 # input is (nc) x 64 x 64
125                 nn.Conv3d(nc, ndf, 4, 2, 1, bias=False),
126                 nn.LeakyReLU(0.2, inplace=True),
127                 # state size. (ndf) x 32 x 32
128                 nn.Conv3d(ndf, ndf * 2, 4, 2, 1, bias=False),
129                 nn.BatchNorm3d(ndf * 2),
130                 nn.LeakyReLU(0.2, inplace=True),
131
132
133                 # state size. (ndf*2) x 16 x 16
134                 nn.Conv3d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
135                 nn.BatchNorm3d(ndf * 4),
136                 nn.LeakyReLU(0.2, inplace=True),
137
138
139
140                 # state size. (ndf*4) x 8 x 8
141                 nn.Conv3d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
142                 nn.BatchNorm3d(ndf * 8),
143                 nn.LeakyReLU(0.2, inplace=True),
144                 # state size. (ndf*8) x 4 x 4
145                 nn.Conv3d(ndf * 8, 1, 4, 1, 0, bias=False),
146                 nn.Sigmoid()
147             )
148         elif size == 128:
149         # 128 x 128
150
151             self.main = nn.Sequential(
152             # input is (nc) x 64 x 64
153             nn.Conv3d(nc, ndf, 4, 2, 1, bias=False),
154             nn.LeakyReLU(0.2, inplace=True),
155             # state size. (ndf) x 32 x 32
156             nn.Conv3d(ndf, ndf , 4, 2, 1, bias=False),
157             nn.BatchNorm3d(ndf),
158             nn.LeakyReLU(0.2, inplace=True),
159
160
161             # state size. (ndf*2) x 16 x 16
162             # nn.Conv3d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
163             # nn.BatchNorm3d(ndf * 4),
164             # nn.LeakyReLU(0.2, inplace=True),
```

```
165
166
167
168             # state size. (ndf*4) x 8 x 8
169             nn.Conv3d(ndf , ndf, 8, 4, 2, bias=False),
170             nn.BatchNorm3d(ndf),
171             nn.LeakyReLU(0.2, inplace=True),
172             # state size. (ndf*8) x 4 x 4
173             nn.Conv3d(ndf, 1, 4, 1, 0, bias=False),
174             nn.BatchNorm3d(1),
175             nn.LeakyReLU(0.2, inplace=True),
176
177
178             nn.Conv3d(1, 1, 5, 1, 0, bias=False),
179             nn.Sigmoid()
180             )
181
182
183         # 32 x 32
184         elif size == 32:
185             self.main = nn.Sequential(
186                 # input is (nc) x 64 x 64
187                 nn.Conv3d(nc, ndf, 4, 2, 1, bias=False),
188                 nn.LeakyReLU(0.2, inplace=True),
189                 # state size. (ndf) x 32 x 32
190                 # nn.Conv3d(ndf, ndf * 2, 4, 2, 1, bias=False),
191                 # nn.BatchNorm3d(ndf * 2),
192                 # nn.LeakyReLU(0.2, inplace=True),
193
194
195                 # state size. (ndf*2) x 16 x 16
196                 nn.Conv3d(ndf, ndf * 4, 4, 2, 1, bias=False),
197                 nn.BatchNorm3d(ndf * 4),
198                 nn.LeakyReLU(0.2, inplace=True),
199
200
201
202                 # state size. (ndf*4) x 8 x 8
203                 nn.Conv3d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
204                 nn.BatchNorm3d(ndf * 8),
205                 nn.LeakyReLU(0.2, inplace=True),
206                 # state size. (ndf*8) x 4 x 4
207                 nn.Conv3d(ndf * 8, 1, 4, 1, 0, bias=False),
208                 nn.Sigmoid()
209             )
210
211     def forward(self, input):
212       #  breakpoint()
213         return self.main(input)
214
```