import torch.nn as nn

import torch

import torch.nn as nn

import torch.nn.functional as F

class Autoencoder\_3\_0(nn.Module):

def \_\_init\_\_(self, dropout\_rate=0.2):

super(Autoencoder\_3\_0, self).\_\_init\_\_()

# Encoder

self.enc = nn.Sequential(

nn.Conv2d(3, 32, 4, stride=2, padding=1),

nn.BatchNorm2d(32),

nn.ReLU(True),

nn.MaxPool2d(2, stride=2, padding=0),

nn.Conv2d(32, 32, 4, stride=1, padding=1),

nn.BatchNorm2d(32),

nn.ReLU(True),

nn.Conv2d(32, 128, 4, stride=2, padding=1),

nn.BatchNorm2d(128),

nn.ReLU(True),

nn.MaxPool2d(2, stride=2, padding=0),

nn.Dropout(dropout\_rate),

nn.Conv2d(128, 64, 4, stride=1, padding=1),

nn.BatchNorm2d(64),

nn.ReLU(True),

nn.Conv2d(64, 128, 4, stride=2, padding=2),

nn.BatchNorm2d(128),

nn.ReLU(True),

)

self.dec = nn.Sequential(

nn.Upsample(scale\_factor=2, mode='nearest'), # -> [batch\_size, 2048, 16, 16]

nn.Conv2d(128, 64, kernel\_size=3, stride=1, padding=1),

nn.LeakyReLU(True),

nn.Upsample(scale\_factor=2, mode='nearest'), # -> [batch\_size, 1024, 32, 32]

nn.Conv2d(64, 32, kernel\_size=3, stride=1, padding=1),

nn.LeakyReLU(True),

nn.Dropout(dropout\_rate),

nn.Upsample(scale\_factor=2, mode='nearest'),

nn.Conv2d(32, 16, kernel\_size=3, stride=1, padding=1), # -> [batch\_size, 512, 64, 64]

nn.LeakyReLU(True),

nn.Upsample(scale\_factor=2, mode='nearest'),

nn.Conv2d(16, 8, kernel\_size=3, stride=1, padding=1),

nn.LeakyReLU(True),

nn.Upsample(scale\_factor=2, mode='nearest'),

nn.Conv2d(8, 3, kernel\_size=3, stride=1, padding=1), # -> [batch\_size, 3, 256, 256]

nn.LeakyReLU(True),

nn.Sigmoid()

)

def forward(self, x):

x = self.enc(x)

x = self.dec(x)

return x