final training accuracy: 0.8535842662775311 final training loss: 0.029425408544116897

final validation accuracy: 0.7037328556425894 final validation loss: 0.11306891909667424

final test accuracy: 0.7235128320876522 final test loss: 0.09641786345413753

class Autoencoder\_3\_1(nn.Module):

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

super(Autoencoder\_3\_1, self).\_\_init\_\_()

# Encoder

self.enc1 = nn.Sequential(

nn.Conv2d(3, 16, 16, stride=2, padding=1),

nn.ReLU(True),

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

)

self.enc2 = nn.Sequential(

nn.Conv2d(16, 32, 3, stride=2, padding=1),

nn.ReLU(True),

nn.Dropout(dropout\_rate),

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

)

self.enc3 = nn.Sequential(

nn.Conv2d(32, 64, 3, stride=2, padding=1),

nn.ReLU(True),

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

)

self.enc4 = nn.Sequential(

nn.Conv2d(64, 128, 3),

nn.ReLU(True),

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

)

# Decoder

self.dec1 = nn.Sequential(

nn.ConvTranspose2d(128, 128, kernel\_size=4, stride=2, padding = 1),

nn.ConvTranspose2d(128, 64, 3),

nn.LeakyReLU(True),

)

self.dec2 = nn.Sequential(

nn.ConvTranspose2d(64, 64, kernel\_size=4, stride=2, padding = 1),

nn.ConvTranspose2d(64, 32, 3, stride=2, padding=1, output\_padding=0),

nn.LeakyReLU(True),

)

self.dec3 = nn.Sequential(

nn.ConvTranspose2d(32, 32, kernel\_size=4, stride=2, padding = 1),

nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output\_padding=1),

nn.LeakyReLU(True),

nn.Dropout(dropout\_rate),

)

self.dec4 = nn.Sequential(

nn.ConvTranspose2d(16, 16, kernel\_size=4, stride=2, padding = 1),

nn.ConvTranspose2d(16, 3, 14, stride=2, padding=0, output\_padding=0),

nn.LeakyReLU(True),

nn.Sigmoid(),

)

def forward(self, x):

# Encoder

enc1 = self.enc1(x)

enc2 = self.enc2(enc1)

enc3 = self.enc3(enc2)

enc4 = self.enc4(enc3)

# Decoder with skip connections

dec1 = self.dec1(enc4)

# Ensure matching dimensions for skip connection

dec1 = F.interpolate(dec1, size=enc3.size()[2:]) + enc3

dec2 = self.dec2(dec1)

# Ensure matching dimensions for skip connection

dec2 = F.interpolate(dec2, size=enc2.size()[2:]) + enc2

dec3 = self.dec3(dec2)

# Ensure matching dimensions for skip connection

dec3 = F.interpolate(dec3, size=enc1.size()[2:]) + enc1

x = self.dec4(dec3)

return x