Batchsize = 64

Learning rate = 0.0005

Epoch = 30

Total time elapsed: 1600.02 seconds

final training accuracy: 0.9218431754173071 final training loss: 0.013143659655670851

final validation accuracy: 0.8386178158297397 final validation loss: 0.0628791180654214

final test accuracy: 0.8343094142750151 final test loss: 0.06433799304068089

class AutoencoderResNet50(nn.Module):

def \_\_init\_\_(self):

super(AutoencoderResNet50, self).\_\_init\_\_()

# Load a pretrained ResNet50 model

resnet50 = models.resnet50(pretrained=True)

self.encoder = nn.Sequential(\*list(resnet50.children())[:-2])

# Adjusted decoder with nn.Upsample

self.decoder = nn.Sequential(

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

nn.Conv2d(2048, 1024, kernel\_size=3, stride=1, padding=1),

nn.LeakyReLU(True),

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

nn.Conv2d(1024, 512, kernel\_size=3, stride=1, padding=1),

nn.LeakyReLU(True),

nn.Upsample(scale\_factor=2, mode='nearest'), # -> [batch\_size, 512, 64, 64]

nn.Conv2d(512, 256, kernel\_size=3, stride=1, padding=1),

nn.LeakyReLU(True),

nn.Upsample(scale\_factor=2, mode='nearest'), # -> [batch\_size, 256, 128, 128]

nn.Conv2d(256, 128, kernel\_size=3, stride=1, padding=1),

nn.LeakyReLU(True),

nn.Upsample(scale\_factor=2, mode='nearest'), # -> [batch\_size, 128, 256, 256]

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

nn.LeakyReLU(True),

nn.Conv2d(64, 3, kernel\_size=3, stride=1, padding=1), # -> [batch\_size, 3, 256, 256]

nn.Sigmoid()

)

def forward(self, x):

x = self.encoder(x)

x = self.decoder(x)

return x