Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №5**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Свёрточные автоэнкодеры»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2025

В рамках данной работы необходимо использовать сверточные автоэнкодеры для классификации изображений(batch\_learning, реализовать вручную, но так, чтоб вычисления можно было проводить на gpu. Например на PyTorch).  
Если обучение на всех классах является затруднительным разрешается взять часть из них. Система должна различать не менее 5-х классов.

**Ход работы**

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как Шрифт, текст, белый, Графика

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, снимок экрана, График, линия

Контент, сгенерированный ИИ, может содержать ошибки.from utils.cnn\_autoencoder\_layers import \*  
class ConvAutoencoder:  
 def \_\_init\_\_(self):  
 # Encoder  
 self.enc1 = Conv\_Layer((3, 3, 1, 16), pad=1)  
 self.relu1 = ReLU\_Layer()  
 self.pool1 = MaxPool\_Layer(stride=2, f=2)  
 self.enc2 = Conv\_Layer((3, 3, 16, 32), pad=1)  
 self.relu2 = ReLU\_Layer()  
 self.pool2 = MaxPool\_Layer(stride=2, f=2)  
 # Decoder  
 self.up1 = Upsample\_Layer(scale\_factor=2)  
 self.dec1 = Conv\_Layer((3, 3, 32, 16), pad=1)  
 self.relu3 = ReLU\_Layer()  
 self.up2 = Upsample\_Layer(scale\_factor=2)  
 self.dec2 = Conv\_Layer((3, 3, 16, 1), pad=1)  
 self.sigmoid = Sigmoid\_Layer()  
 def forward\_propagation(self, X, mode="train"):  
 self.input = X.clone()  
 # Encoder  
 x = self.enc1.forward\_propagation(X)  
 x = self.relu1.forward\_propagation(x)  
 x = self.pool1.forward\_propagation(x)   
 x = self.enc2.forward\_propagation(x)  
 x = self.relu2.forward\_propagation(x)  
 x = self.pool2.forward\_propagation(x)  
 # Decoder  
 x = self.up1.forward\_propagation(x)  
 x = self.dec1.forward\_propagation(x)  
 x = self.relu3.forward\_propagation(x)  
 x = self.up2.forward\_propagation(x)  
 x = self.dec2.forward\_propagation(x)  
 self.output = self.sigmoid.forward\_propagation(x)  
 if mode == "train":  
 self.loss = torch.mean((self.input - self.output) \*\* 2)   
 return self.loss.item()  
 return self.output  
 def back\_propagation(self, lr):  
 dout = (self.output - self.input) / self.input.numel()  
 # Decoder  
 dout = self.sigmoid.back\_propagation(dout)  
 dout = self.dec2.back\_propagation(dout, lr)  
 dout = self.up2.back\_propagation(dout)  
 dout = self.relu3.back\_propagation(dout)  
 dout = self.dec1.back\_propagation(dout, lr)  
 dout = self.up1.back\_propagation(dout)  
 # Encoder  
 dout = self.pool2.back\_propagation(dout)  
 dout = self.relu2.back\_propagation(dout)   
 dout = self.enc2.back\_propagation(dout, lr)  
 dout = self.pool1.back\_propagation(dout)  
 dout = self.relu1.back\_propagation(dout)  
 self.enc1.back\_propagation(dout, lr)

**Вывод:** в ходе лабораторной работы я научился реализовывать CNN.