Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №4**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Свёрточные сети»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2025

В рамках данной работы необходимо реализовать и обучить свёрточную нейронную сеть для классификации изображений(batch\_learning, реализовать вручную, но так, чтоб вычисления можно было проводить на gpu. Например на PyTorch).  
Если обучение сети на всех классах является затруднительным разрешается взять часть из них. Сеть должна различать не менее 3-х классов.

**Ход работы**

Custom\_сnn.py:

Изображение выглядит как текст, Шрифт, снимок экрана

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

Контент, сгенерированный ИИ, может содержать ошибки.from utils.cnn\_layers import Conv\_Layer, ReLU\_Layer, MaxPool\_Layer, FC\_Layer, FC\_Output\_Layer  
class CustomCNN:  
 def \_\_init\_\_(self):  
 self.C1 = Conv\_Layer((5,5,1,32), pad=2, sigma=0.05, bias\_factor=0.005)  
 self.ReLU1 = ReLU\_Layer()  
 self.S2 = MaxPool\_Layer()  
 self.C3 = Conv\_Layer((5,5,32,64), sigma=0.05, bias\_factor=0.005)  
 self.ReLU2 = ReLU\_Layer()  
 self.S4 = MaxPool\_Layer()  
 self.C5 = Conv\_Layer((5,5,64,128), sigma=0.05, bias\_factor=0.005)  
 self.ReLU3 = ReLU\_Layer()  
 self.F6 = FC\_Layer((128,256), sigma=0.05, bias\_factor=0.005)  
 self.ReLU4 = ReLU\_Layer()  
 self.F7 = FC\_Output\_Layer((256,10), sigma=0.05, bias\_factor=0.005)  
 def forward\_propagation(self, X, labels, mode):  
 x = self.C1.forward\_propagation(X)  
 x = self.ReLU1.forward\_propagation(x)  
 x = self.S2.forward\_propagation(x)  
 x = self.C3.forward\_propagation(x)  
 x = self.ReLU2.forward\_propagation(x)  
 x = self.S4.forward\_propagation(x)  
 x = self.C5.forward\_propagation(x)  
 x = self.ReLU3.forward\_propagation(x)  
 x = x[:,0,0,:]  
 x = self.F6.forward\_propagation(x)  
 x = self.ReLU4.forward\_propagation(x)  
 return self.F7.forward\_propagation(x, labels, mode)  
 def back\_propagation(self, lr):  
 dout = self.F7.back\_propagation(lr)  
 dout = self.ReLU4.back\_propagation(dout)  
 dout = self.F6.back\_propagation(dout, lr)  
 dout = dout.unsqueeze(1).unsqueeze(1)  
 dout = self.ReLU3.back\_propagation(dout)  
 dout = self.C5.back\_propagation(dout, lr)  
 dout = self.S4.back\_propagation(dout)  
 dout = self.ReLU2.back\_propagation(dout)  
 dout = self.C3.back\_propagation(dout, lr)  
 dout = self.S2.back\_propagation(dout)  
 dout = self.ReLU1.back\_propagation(dout)  
 self.C1.back\_propagation(dout, lr)  
 def extract\_model(self):   
 model = CustomCNN()  
 model.C1.weight.data.copy\_(self.C1.weight)  
 model.C1.bias.data.copy\_(self.C1.bias)  
 model.C3.weight.data.copy\_(self.C3.weight)  
 model.C3.bias.data.copy\_(self.C3.bias)  
 model.C5.weight.data.copy\_(self.C5.weight)  
 model.C5.bias.data.copy\_(self.C5.bias)  
 model.F6.weight.data.copy\_(self.F6.weight)  
 model.F6.bias.data.copy\_(self.F6.bias)  
 model.F7.weight.data.copy\_(self.F7.weight)  
 model.F7.bias.data.copy\_(self.F7.bias)  
 return model

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