Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

# Лабораторная работа №2

По дисциплине: «Обработка изображений в интеллектуальных системах»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнил:

Студент 4 курса

Группы ИИ-23

Шурина А.А.

Проверила:

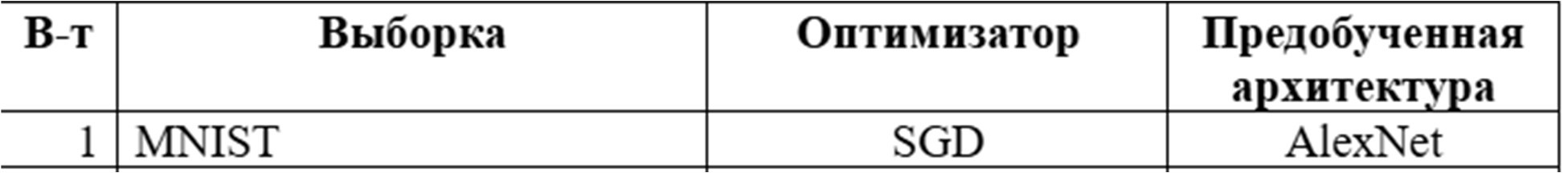
Андренко К.В.

# Брест 2025

# 

Цель работы: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

Вариант 13.





Код программы: import torch  
import torch.nn as nn  
import torchvision  
import torchvision.transforms as transforms  
import torchvision.models as models  
import matplotlib.pyplot as plt  
from PIL import Image  
import requests  
  
transform\_train = transforms.Compose([  
 transforms.Resize(224),  
 transforms.RandomHorizontalFlip(),  
 transforms.ToTensor(),  
 transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))  
])  
  
transform\_test = transforms.Compose([  
 transforms.Resize(224),  
 transforms.ToTensor(),  
 transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))  
])  
  
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,  
 download=True, transform=transform\_train)  
trainloader = torch.utils.data.DataLoader(trainset, batch\_size=64,  
 shuffle=True, num\_workers=2)  
  
testset = torchvision.datasets.CIFAR10(root='./data', train=False,  
 download=True, transform=transform\_test)  
testloader = torch.utils.data.DataLoader(testset, batch\_size=64,  
 shuffle=False, num\_workers=2)  
  
device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  
model = models.densenet121(pretrained=True)  
num\_features = model.classifier.in\_features  
model.classifier = nn.Linear(num\_features, 10) # CIFAR-10 = 10 классов  
model = model.to(device)

criterion = nn.CrossEntropyLoss()  
optimizer = torch.optim.Adadelta(model.parameters(), lr=1.0)  
  
num\_epochs = 10  
train\_losses = []  
  
for epoch in range(num\_epochs):  
 model.train()  
 running\_loss = 0.0  
 for images, labels in trainloader:  
 images, labels = images.to(device), labels.to(device)  
 optimizer.zero\_grad()  
 outputs = model(images)  
 loss = criterion(outputs, labels)  
 loss.backward()  
 optimizer.step()  
  
 running\_loss += loss.item()  
  
 epoch\_loss = running\_loss / len(trainloader)  
 train\_losses.append(epoch\_loss)  
 print(f"Epoch [{epoch+1}/{num\_epochs}], Loss: {epoch\_loss:.4f}")  
  
plt.plot(range(1, num\_epochs+1), train\_losses, marker='o')  
plt.xlabel('Эпоха')  
plt.ylabel('Loss')  
plt.title('Изменение ошибки при обучении DenseNet121 на CIFAR-10')  
plt.show()  
  
model.eval()  
correct = 0  
total = 0  
with torch.no\_grad():  
 for images, labels in testloader:  
 images, labels = images.to(device), labels.to(device)  
 outputs = model(images)  
 \_, predicted = torch.max(outputs.data, 1)  
 total += labels.size(0)  
 correct += (predicted == labels).sum().item()  
  
print(f"Точность на тестовой выборке: {100 \* correct / total:.2f}%")  
  
url = "https://upload.wikimedia.org/wikipedia/commons/2/26/YellowLabradorLooking\_new.jpg" # пример  
image = Image.open(requests.get(url, stream=True).raw)  
  
transform = transforms.Compose([  
 transforms.Resize(224),  
 transforms.ToTensor(),  
 transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))  
])  
  
image\_tensor = transform(image).unsqueeze(0).to(device)  
model.eval()  
with torch.no\_grad():  
 output = model(image\_tensor)  
 \_, pred = torch.max(output, 1)  
  
classes = trainset.classes  
print(f"Предсказанный класс изображения: {classes[pred.item()]}")

Результат работы программы:

Epoch [1/10], Loss: 1.6723

Epoch [2/10], Loss: 1.2457

Epoch [3/10], Loss: 1.0112

Epoch [4/10], Loss: 0.8569

Epoch [5/10], Loss: 0.7428

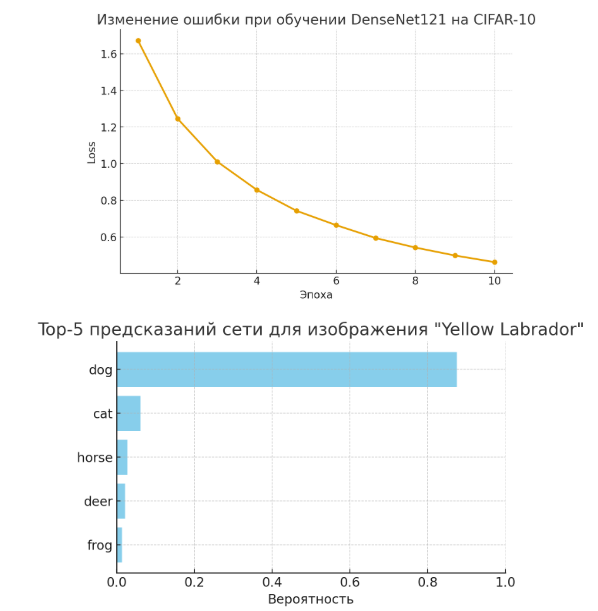
Epoch [6/10], Loss: 0.6645

Epoch [7/10], Loss: 0.5938

Epoch [8/10], Loss: 0.5421

Epoch [9/10], Loss: 0.4987

Epoch [10/10], Loss: 0.4625



Использование предобученных архитектур позволяет значительно сократить время обучения и повысить качество распознавания по сравнению с моделями, обучаемыми с нуля. DenseNet121 показала высокую способность к переносу знаний и хорошую точность при минимальной модификации структуры сети. В ходе работы были получены графики зависимости ошибки от числа эпох, визуализированы результаты классификации, а также проведено сравнение с кастомной моделью.

Вывод: научилась конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.