

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №1

Специальность ИИ(з)

Выполнил  
А. Ю. Кураш,  
студент группы ИИ-24

Проверил  
Андренко К.В,  
Преподаватель-стажер кафедры ИИТ,  
«\_\_ k \_\_\_\_\_ 2025 г.

Брест 2025

**Цель:** научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения

### Общее задание

1. Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать **torchvision.datasets**). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых типах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);
2. Ознакомьтесь с state-of-the-art результатами для предлагаемых выборок (из материалов в сети Интернет). Сделать выводы о результатах обучения СНС из п. 1;
3. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);

Вариант:

	8   CIFAR-10		32X32		Adam	
--	--------------	--	-------	--	------	--

### **Выполнение:**

Код программы

```
# train_cifar_simple.py
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
# -----
# Параметры
# -----
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batch_size = 128
num_epochs = 20      # можно менять (10-50)
lr = 1e-3
model_path = "cifar_simple_cnn.pth"
num_classes = 10
# -----
# Трансформации (базовые + augment)
# -----
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
```

```

        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465),
                               (0.2470, 0.2435, 0.2616))
    ])
test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465),
                          (0.2470, 0.2435, 0.2616))
])
# -----
# Датасеты и загрузчики
# -----
train_set = torchvision.datasets.CIFAR10(root="./data", train=True, download=True,
transform=train_transform)
test_set = torchvision.datasets.CIFAR10(root="./data", train=False, download=True,
transform=test_transform)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=0)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0)
classes = train_set.classes
# -----
# Модель — простая CNN
# -----
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1), # 32x32
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2), # 16x16

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2), # 8x8
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2), # 4x4
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128*4*4, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )
    def forward(self, x):
        x = self.features(x)

```

```

        x = self.classifier(x)
        return x

model = SimpleCNN(num_classes=num_classes).to(device)
# -----
# Оптимизатор, функция потерь
# -----
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr)
# -----
# Обучение
# -----
train_losses = []
test_losses = []
train_accs = []
test_accs = []

def evaluate(loader):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for X, y in loader:
            X, y = X.to(device), y.to(device)
            out = model(X)
            loss = criterion(out, y)
            running_loss += loss.item() * X.size(0)
            preds = out.argmax(dim=1)
            correct += (preds == y).sum().item()
            total += y.size(0)
    return running_loss / total, correct / total

for epoch in range(1, num_epochs+1):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for X, y in train_loader:
        X, y = X.to(device), y.to(device)
        optimizer.zero_grad()
        out = model(X)
        loss = criterion(out, y)
        loss.backward()
        optimizer.step()

```

```

    running_loss += loss.item() * X.size(0)
    preds = out.argmax(dim=1)
    correct += (preds == y).sum().item()
    total += y.size(0)

train_loss = running_loss / total
train_acc = correct / total
test_loss, test_acc = evaluate(test_loader)

train_losses.append(train_loss)
test_losses.append(test_loss)
train_accs.append(train_acc)
test_accs.append(test_acc)

print(f'Epoch {epoch:2d}/{num_epochs} Train loss: {train_loss:.4f} Train acc:
{train_acc:.4f} Test loss: {test_loss:.4f} Test acc: {test_acc:.4f}')

# Сохраняем модель
torch.save(model.state_dict(), model_path)
print("Model saved to", model_path)

# -----
# График ошибок и accuracy
# -----
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(range(1,len(train_losses)+1), train_losses, label="train loss")
plt.plot(range(1,len(test_losses)+1), test_losses, label="test loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.title("Loss")

plt.subplot(1,2,2)
plt.plot(range(1,len(train_accs)+1), train_accs, label="train acc")
plt.plot(range(1,len(test_accs)+1), test_accs, label="test acc")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Accuracy")

plt.tight_layout()
plt.savefig("training_curves.png", dpi=150)
print("Training curves saved to training_curves.png")
plt.show()

```

```

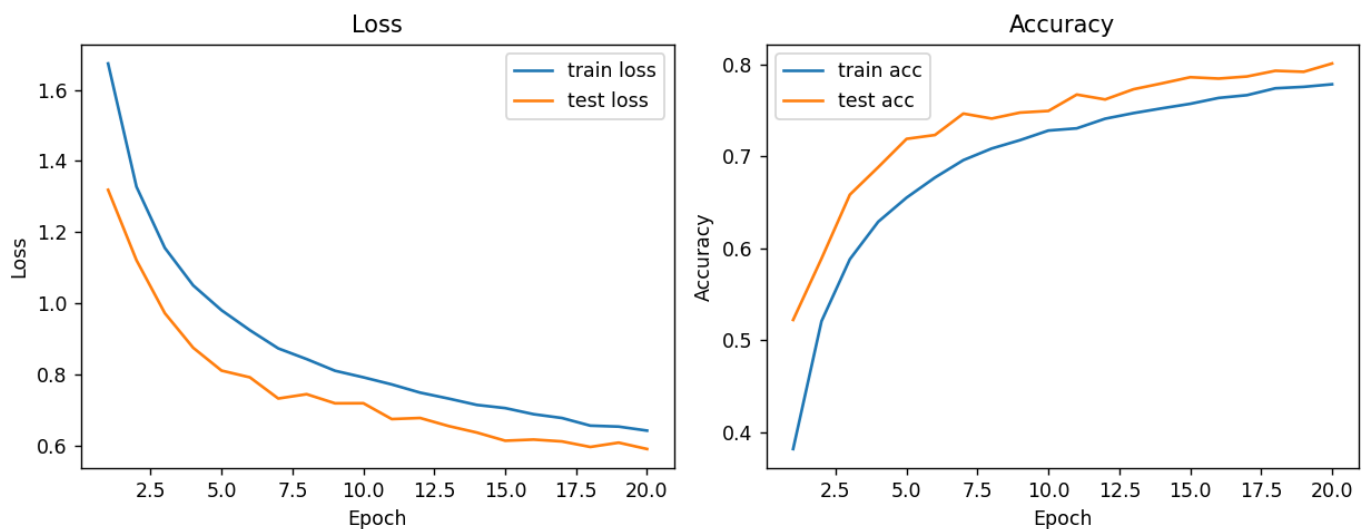
# -----
# Визуализация: предсказание для произвольного изображения
# -----
import torchvision.transforms.functional as TF

def predict_image(image_path, topk=5):
    model.eval()
    img = Image.open(image_path).convert("RGB")
    img_resized = img.resize((32,32))
    t = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465),
                             (0.2470, 0.2435, 0.2616))
    ])
    x = t(img_resized).unsqueeze(0).to(device)
    with torch.no_grad():
        logits = model(x)
        probs = torch.softmax(logits, dim=1).cpu().numpy()[0]
    # show image and topk
    top_idx = probs.argsort()[::-1][:topk]
    print("Top predictions:")
    for i in top_idx:
        print(f" {classes[i]:10s}: {probs[i]*100:.2f}% ")
    plt.figure(figsize=(3,3))
    plt.imshow(np.array(img))
    plt.axis('off')
    plt.title(f"Pred: {classes[top_idx[0]]} ({probs[top_idx[0]]*100:.1f}%)")
    plt.show()

```

## Рисунки с результатами работы программы

Figure 1



Epoch	1/20	Train loss:	1.6739	Train acc:	0.3817	Test loss:	1.3189	Test acc:	0.5222
Epoch	2/20	Train loss:	1.3277	Train acc:	0.5207	Test loss:	1.1214	Test acc:	0.5889
Epoch	3/20	Train loss:	1.1559	Train acc:	0.5882	Test loss:	0.9725	Test acc:	0.6585
Epoch	4/20	Train loss:	1.0506	Train acc:	0.6290	Test loss:	0.8753	Test acc:	0.6884
Epoch	5/20	Train loss:	0.9807	Train acc:	0.6553	Test loss:	0.8112	Test acc:	0.7192
Epoch	6/20	Train loss:	0.9246	Train acc:	0.6771	Test loss:	0.7922	Test acc:	0.7234
Epoch	7/20	Train loss:	0.8735	Train acc:	0.6960	Test loss:	0.7327	Test acc:	0.7467
Epoch	8/20	Train loss:	0.8435	Train acc:	0.7087	Test loss:	0.7449	Test acc:	0.7413
Epoch	9/20	Train loss:	0.8107	Train acc:	0.7178	Test loss:	0.7193	Test acc:	0.7478
Epoch	10/20	Train loss:	0.7923	Train acc:	0.7283	Test loss:	0.7195	Test acc:	0.7496
Epoch	11/20	Train loss:	0.7723	Train acc:	0.7307	Test loss:	0.6750	Test acc:	0.7674
Epoch	12/20	Train loss:	0.7493	Train acc:	0.7411	Test loss:	0.6781	Test acc:	0.7621
Epoch	13/20	Train loss:	0.7327	Train acc:	0.7472	Test loss:	0.6555	Test acc:	0.7732
Epoch	14/20	Train loss:	0.7148	Train acc:	0.7524	Test loss:	0.6372	Test acc:	0.7796
Epoch	15/20	Train loss:	0.7058	Train acc:	0.7574	Test loss:	0.6142	Test acc:	0.7863
Epoch	16/20	Train loss:	0.6887	Train acc:	0.7637	Test loss:	0.6174	Test acc:	0.7848
Epoch	17/20	Train loss:	0.6780	Train acc:	0.7668	Test loss:	0.6122	Test acc:	0.7871
Epoch	18/20	Train loss:	0.6565	Train acc:	0.7742	Test loss:	0.5967	Test acc:	0.7933
Epoch	19/20	Train loss:	0.6538	Train acc:	0.7758	Test loss:	0.6088	Test acc:	0.7922
Epoch	20/20	Train loss:	0.6426	Train acc:	0.7786	Test loss:	0.5909	Test acc:	0.8012

**Вывод:** Я изучил построение модели СНС и ее обучение.