Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «ОИвИС»

Тема: “Обучение классификаторов средствами библиотеки PyTorch”

Выполнила:

Студентка

4 курса

группы ИИ-22

Сокол С.М.

Проверил:

Крощенко А.А.

Брест 2024

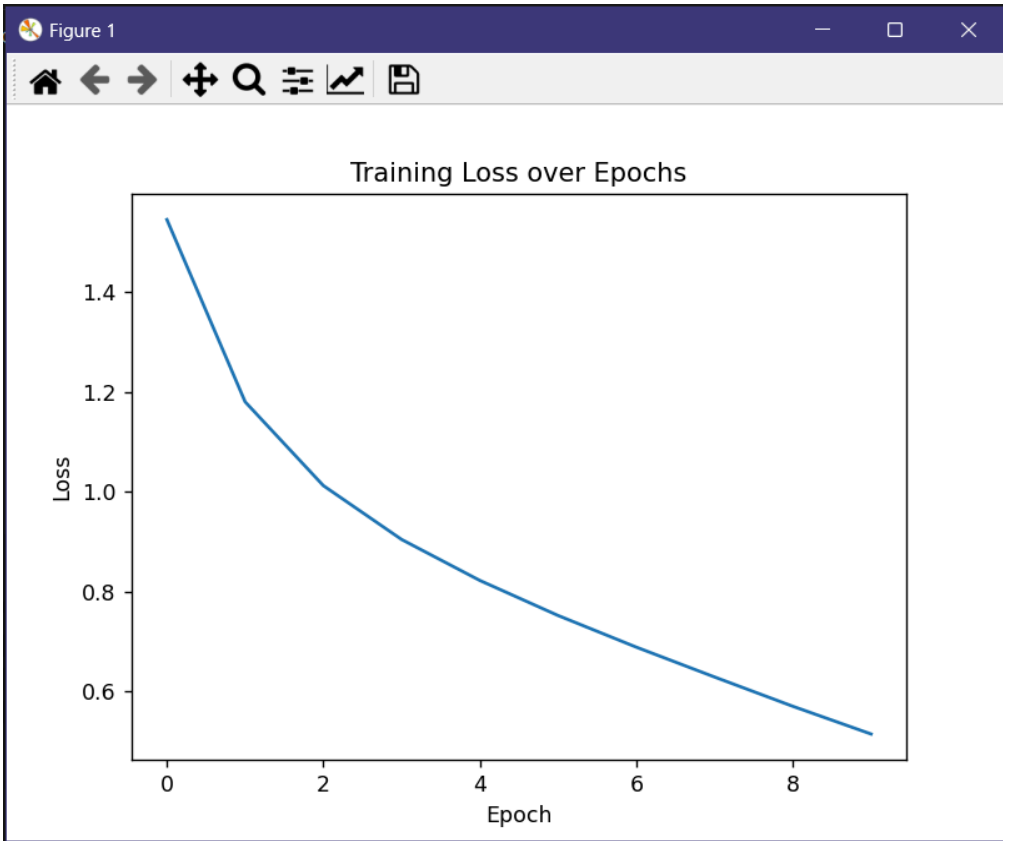
Вариант 18.
Выборка: CIFAR-10

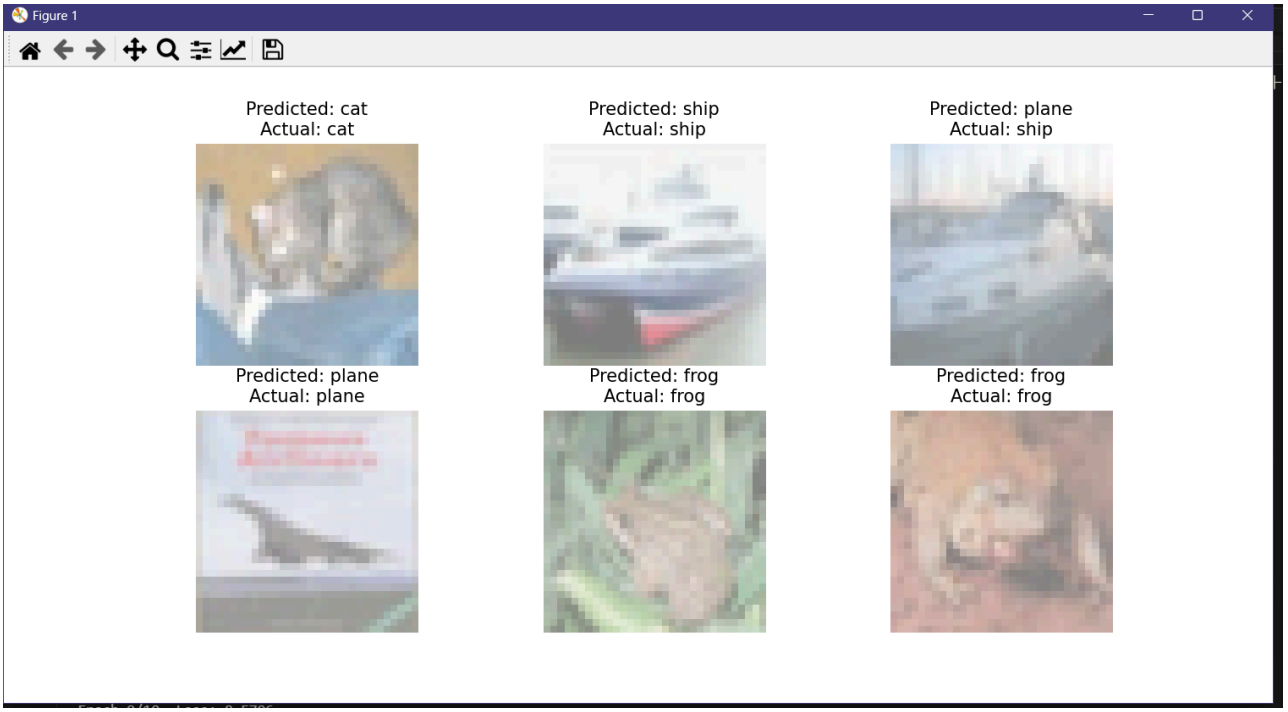
Размер исходного изображения: 32*32
Оптимизатор: RMSprop
Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения
Задание 1. Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать **torchvision.datasets**). 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);

Задание 2. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);

Вывод программы:

```
Epoch 1/10, Loss: 1.5251
Epoch 2/10, Loss: 1.1194
Epoch 3/10, Loss: 0.9631
Epoch 4/10, Loss: 0.8597
Epoch 5/10, Loss: 0.7718
Epoch 6/10, Loss: 0.6973
Epoch 7/10, Loss: 0.6291
Epoch 8/10, Loss: 0.5647
Epoch 9/10, Loss: 0.5053
Epoch 10/10, Loss: 0.4488
Accuracy of the model on the test images: 68.94%
```





Код программы:

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import pickle
import os

# Установим устройство для выполнения (GPU или CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Функция для загрузки данных CIFAR-10 из .py файлов
```

```
def load_cifar10_batch(file):
    with open(file, 'rb') as f:
        batch = pickle.load(f, encoding='bytes')
    return batch[b'data'], batch[b'labels']

def load_cifar10_data(data_dir):
    images = []
    labels = []
    for i in range(1, 6):
        batch_file = os.path.join(data_dir, f'data_batch_{i}')
        data, label = load_cifar10_batch(batch_file)
        images.append(data)
        labels.append(label)

    # Объединяем все данные в один массив
    images = np.vstack(images)
    labels = np.hstack(labels)

    # Преобразуем в тензоры
    images = torch.tensor(images, dtype=torch.float32).view(-1, 3, 32, 32) / 255.0
    labels = torch.tensor(labels, dtype=torch.long)

    return images, labels

# Загрузка данных
data_dir = './cifar-10-batches-py'  # Путь к вашей директории с данными
train_images, train_labels = load_cifar10_data(data_dir)

# Для тестовой выборки загружаем test_batch
test_images, test_labels = load_cifar10_batch(os.path.join(data_dir, 'test_batch'))

# Преобразуем тестовые данные в тензоры
test_images = torch.tensor(test_images, dtype=torch.float32).view(-1, 3, 32, 32) /
255.0
test_labels = torch.tensor(test_labels, dtype=torch.long)

# Создаем DataLoader
train_dataset = torch.utils.data.TensorDataset(train_images, train_labels)
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)

test_dataset = torch.utils.data.TensorDataset(test_images, test_labels)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship',
'truck')

# Определение архитектуры модели
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
# Инициализация модели
model = SimpleCNN().to(device)

# Настройка потерь и оптимизатора
criterion = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=0.001)

# Обучение модели
num_epochs = 10
train_losses = []

for epoch in range(num_epochs):
    running_loss = 0.0
    for inputs, labels in trainloader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    avg_loss = running_loss / len(trainloader)
    train_losses.append(avg_loss)
    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {avg_loss:.4f}')

# Оценка модели на тестовой выборке
correct = 0
total = 0

with torch.no_grad():
    for inputs, labels in testloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the model on the test images: {100 * correct / total:.2f}%')

# Построение графика изменения ошибки
plt.plot(train_losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss over Epochs')
plt.show()

# Визуализация работы модели
def visualize_model(model, num_images=6):
    model.eval()
    dataiter = iter(testloader)
    images, labels = next(dataiter)

    # Получаем предсказания модели
    outputs = model(images.to(device))
    _, preds = torch.max(outputs, 1)

    # Показать изображения
    plt.figure(figsize=(12, 6))
    for idx in range(num_images):
        ax = plt.subplot(2, 3, idx + 1)
        plt.imshow(images[idx].numpy().transpose((1, 2, 0)) / 2 + 0.5)  # Denormalize
        plt.title(f'Predicted: {classes[preds[idx]]}\nActual: {classes[labels[idx]]}')
```

```
        plt.axis('off')
    plt.show()

visualize_model(model)
```

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