Министерство цифрового развития

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт

по лабораторной работе № 6 «Нейронные сети с использованием Pytorch для классификации изображений»

Выполнил:

студент группы ИП-213

Дмитриев Антон Александрович

Работу проверил: Преподаватель
Сороковых Дарья Анатольевна

Новосибирск 2025 г.

Введение (задание)

Задание

Задание 1:

1. Загрузите датасет с правильными преобразованиями

2. Выведите информацию о размерах тренировочной и тестовой выборок

3. Визуализируйте 10 случайных изображений с подписями

Задание 2.

Создайте сверточную нейронную сеть для классификации изображений в соответствии с вашим вариантом. Обучите её и выполните предсказание на нескольких изображениях не из набора данных.

Задание 3.

Проанализируйте ошибки на процессе обучения и тестирования модели.

Основная часть

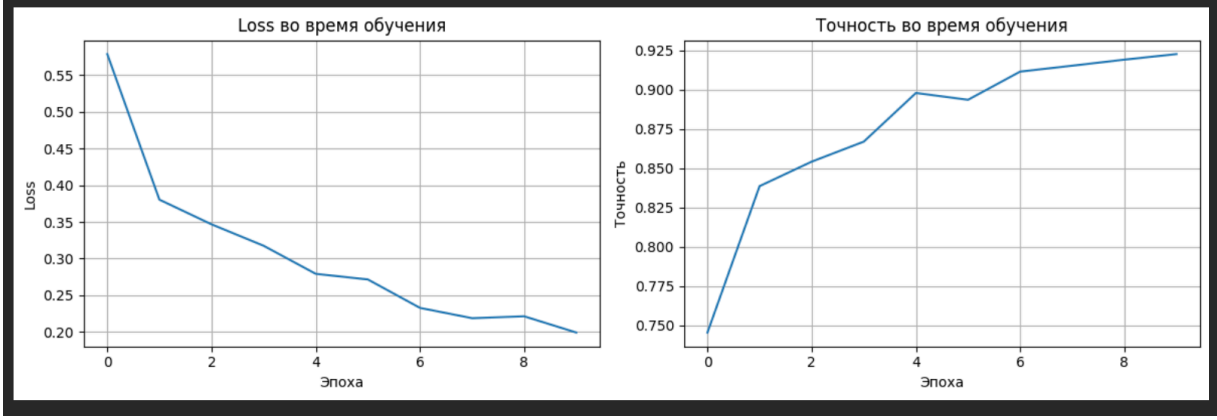
1. Задание 1

2. Задание 2

```
Отчет по классификации:
              precision    recall  f1-score   support

      muffin       0.95      0.86      0.90       544
    chihuahua       0.89      0.96      0.92       640

    accuracy                           0.91      1184
   macro avg       0.92      0.91      0.91      1184
weighted avg       0.92      0.91      0.91      1184


Результаты на тестовой выборке:
Средняя Loss: 0.2048
Точность: 0.9147 (1083/1184)

Визуализация предсказаний на тестовой выборке:
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:627: UserWarning: This DataLoader will create 4 worker processes in total. Our sug
  warnings.warn(
```

3. Задание 3

```
Эпоха [10/10], Батч [0/148], Loss: 0.0953, Accuracy: 1.0000
Эпоха [10/10], Батч [10/148], Loss: 0.1213, Accuracy: 0.9375
Эпоха [10/10], Батч [20/148], Loss: 0.1529, Accuracy: 0.9375
Эпоха [10/10], Батч [30/148], Loss: 0.2833, Accuracy: 0.9062
Эпоха [10/10], Батч [40/148], Loss: 0.1594, Accuracy: 0.8438
Эпоха [10/10], Батч [50/148], Loss: 0.1665, Accuracy: 0.9375
Эпоха [10/10], Батч [60/148], Loss: 0.1574, Accuracy: 0.9062
Эпоха [10/10], Батч [70/148], Loss: 0.2387, Accuracy: 0.9062
Эпоха [10/10], Батч [80/148], Loss: 0.1680, Accuracy: 0.9062
Эпоха [10/10], Батч [90/148], Loss: 0.0804, Accuracy: 1.0000
Эпоха [10/10], Батч [100/148], Loss: 0.2537, Accuracy: 0.9062
Эпоха [10/10], Батч [110/148], Loss: 0.1324, Accuracy: 0.9375
Эпоха [10/10], Батч [120/148], Loss: 0.1915, Accuracy: 0.9688
Эпоха [10/10], Батч [130/148], Loss: 0.3592, Accuracy: 0.7812
Эпоха [10/10], Батч [140/148], Loss: 0.1516, Accuracy: 0.9688
Эпоха [10/10 завершена] -> Средняя Loss: 0.1990, Точность: 0.9226
--------------------------------------------------
```




```
Анализ ошибок...
Количество ошибок: 101

Примеры ошибок классификации:
```

Заключение

Ссылка на colab: https://colab.research.google.com/drive/1VbAUkYv293eY-sArbWOQ0fSmX1m7lfC1?usp=sharing

Код программы

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, Dataset
import matplotlib.pyplot as plt
import numpy as np
from google.colab import drive
import pandas as pd
import os
from PIL import Image

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Используемое устройство: {device}")


def calculate_normalization_parameters(dataset_path, batch_size=32,
num_workers=4):
    """Вычисляет среднее и стандартное отклонение для набора данных"""

    # Простой датасет для расчета статистики
    class SimpleDataset(Dataset):
        def __init__(self, data_dir, transform=None):
            self.data_dir = data_dir
            self.transform = transform
            self.images = []
            self.labels = []

            for label, class_name in enumerate(['muffin', 'chihuahua']):
                class_dir = os.path.join(data_dir, class_name)
                if os.path.exists(class_dir):
                    for img_name in os.listdir(class_dir):
                        if img_name.lower().endswith(('.png', '.jpg',
'.jpeg')):
                            img_path = os.path.join(class_dir, img_name)
                            self.images.append(img_path)
                            self.labels.append(label)

        def __len__(self):
            return len(self.images)

        def __getitem__(self, idx):
            img_path = self.images[idx]
            label = self.labels[idx]

            img = Image.open(img_path).convert('RGB')

            if self.transform:
                img = self.transform(img)
```

```
            return img, label

    # Трансформация только в тензор
    temp_transform = transforms.Compose([
        transforms.Resize((224, 224)),  # Изменяем размер для вычисления
статистики
        transforms.ToTensor()
    ])

    dataset = SimpleDataset(dataset_path, transform=temp_transform)

    if len(dataset) == 0:
        raise ValueError(f"В директории {dataset_path} не найдены
изображения")

    dataloader = DataLoader(
        dataset,
        batch_size=batch_size,
        num_workers=num_workers,
        shuffle=False
    )

    mean = torch.zeros(3)
    std = torch.zeros(3)
    nb_samples = 0.0

    for data, _ in dataloader:
        # data shape: [batch_size, 3, height, width]
        batch_samples = data.size(0)
        data = data.view(batch_samples, data.size(1), -1)
        mean += data.mean(2).sum(0)
        std += data.std(2).sum(0)
        nb_samples += batch_samples

    mean /= nb_samples
    std /= nb_samples

    return mean, std


# Монтируем Google Drive
drive.mount('/content/drive')

# Пути к данным
train_path = '/content/drive/MyDrive/archive/train'
test_path = '/content/drive/MyDrive/archive/test'

# Проверяем наличие директорий
print(f"Проверяем наличие train директории: {os.path.exists(train_path)}")
print(f"Проверяем наличие test директории: {os.path.exists(test_path)}")
```

```
if os.path.exists(train_path):
    print("Содержимое train директории:", os.listdir(train_path))

# Вычисляем нормализацию
print("\nВычисляем параметры нормализации...")
mean, std = calculate_normalization_parameters(train_path, batch_size=16)
print(f"Mean: {mean}, Std: {std}")

# Трансформации
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Изменяем размер изображений
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])

# Создаем кастомный датасет
class MuffinChihuahuaDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.images = []
        self.labels = []

        # 0 - muffin, 1 - chihuahua
        for label, class_name in enumerate(['muffin', 'chihuahua']):
            class_dir = os.path.join(data_dir, class_name)
            if os.path.exists(class_dir):
                for img_name in os.listdir(class_dir):
                    if img_name.lower().endswith(('.png', '.jpg',
'.jpeg')):
                        img_path = os.path.join(class_dir, img_name)
                        self.images.append(img_path)
                        self.labels.append(label)

        print(f"Загружено {len(self.images)} изображений из {data_dir}")

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path = self.images[idx]
        label = self.labels[idx]

        img = Image.open(img_path).convert('RGB')

        if self.transform:
            img = self.transform(img)

        return img, label
```

```
# Создаем датасеты
train_dataset = MuffinChihuahuaDataset(train_path, transform=transform)
test_dataset = MuffinChihuahuaDataset(test_path, transform=transform)

batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, num_workers=4)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False, num_workers=4)

print(f"Размер тренировочного датасета: {len(train_dataset)}")
print(f"Размер тестового датасета: {len(test_dataset)}")
print(f"Количество батчей: {len(train_loader)}")
def show_samples(loader, num_samples=10):
    dataiter = iter(loader)
    images, labels = next(dataiter)

    # Берем только нужное количество изображений
    images = images[:num_samples]
    labels = labels[:num_samples]

    # Денормализуем изображения для отображения
    images_denorm = images.clone()
    for i in range(images.size(1)):
        images_denorm[:, i, :, :] = images_denorm[:, i, :, :] * std[i] +
mean[i]

    # Определяем количество строк и столбцов для отображения
    n_cols = 5  # 5 изображений в ряд
    n_rows = (num_samples + n_cols - 1) // n_cols  # Вычисляем нужное
количество строк

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))

    # Делаем axes двумерным массивом, даже если только одна строка
    if n_rows == 1:
        axes = axes.reshape(1, -1)

    for i in range(num_samples):
        row = i // n_cols
        col = i % n_cols
        ax = axes[row, col]

        # Преобразуем из [C, H, W] в [H, W, C] для отображения
        img = images_denorm[i].permute(1, 2, 0).numpy()
        img = np.clip(img, 0, 1)  # Ограничиваем значения
        ax.imshow(img)
        ax.set_title(f'Label: {"muffin" if labels[i]==0 else
"chihuahua"}')
        ax.axis('off')
```

```
    # Скрываем пустые оси
    for i in range(num_samples, n_rows * n_cols):
        row = i // n_cols
        col = i % n_cols
        axes[row, col].axis('off')

    plt.tight_layout()
    plt.show()
    return images, labels

print("\nПримеры изображений из датасета (10 штук):")
sample_images, sample_labels = show_samples(train_loader, num_samples=10)
print(f"Форма батча изображений: {sample_images.shape}")  # [10, 3, 224,
224]

# Модель для бинарной классификации
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32,
kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128,
kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        # После трех пулов: 224 -> 112 -> 56 -> 28
        self.fc1 = nn.Linear(128 * 28 * 28, 256)
        self.fc2 = nn.Linear(256, 64)
        self.fc3 = nn.Linear(64, 2)  # 2 класса
        self.dropout = nn.Dropout(0.5)
        self.batch_norm1 = nn.BatchNorm2d(32)
        self.batch_norm2 = nn.BatchNorm2d(64)
        self.batch_norm3 = nn.BatchNorm2d(128)

    def forward(self, x):
        x = self.pool(F.relu(self.batch_norm1(self.conv1(x))))
        x = self.pool(F.relu(self.batch_norm2(self.conv2(x))))
        x = self.pool(F.relu(self.batch_norm3(self.conv3(x))))

        x = x.view(-1, 128 * 28 * 28)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)

        return x
```

```
model = SimpleCNN().to(device)
print("\nАрхитектура модели:")
print(model)

def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

total_params = count_parameters(model)
print(f"\nОбщее количество обучаемых параметров: {total_params:,}")

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

def calculate_accuracy(outputs, labels):
    _, predicted = torch.max(outputs.data, 1)
    correct = (predicted == labels).sum().item()
    total = labels.size(0)
    return correct / total

def train_model(model, train_loader, criterion, optimizer, epochs=10):
    model.train()
    train_losses = []
    train_accuracies = []

    for epoch in range(epochs):
        running_loss = 0.0
        running_accuracy = 0.0
        total_batches = len(train_loader)

        for batch_idx, (images, labels) in enumerate(train_loader):
            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()
            running_accuracy += calculate_accuracy(outputs, labels)

            if batch_idx % 10 == 0:
                print(f'Эпоха [{epoch+1}/{epochs}], Батч
[{batch_idx}/{total_batches}], '
                      f'Loss: {loss.item():.4f}, Accuracy:
{calculate_accuracy(outputs, labels):.4f}')

        epoch_loss = running_loss / total_batches
        epoch_accuracy = running_accuracy / total_batches
```

```
        train_losses.append(epoch_loss)
        train_accuracies.append(epoch_accuracy)

        print(f'Эпоха [{epoch+1}/{epochs} завершена] -> '
              f'Средняя Loss: {epoch_loss:.4f}, '
              f'Точность: {epoch_accuracy:.4f}')
        print('-' * 50)

    return train_losses, train_accuracies

print("\nНачинаем обучение...")
train_losses, train_accuracies = train_model(model, train_loader,
criterion, optimizer, epochs=10)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(train_losses)
plt.title('Loss во время обучения')
plt.xlabel('Эпоха')
plt.ylabel('Loss')
plt.grid(True)
plt.subplot(1, 2, 2)
plt.plot(train_accuracies)
plt.title('Точность во время обучения')
plt.xlabel('Эпоха')
plt.ylabel('Точность')
plt.grid(True)

plt.tight_layout()
plt.show()

def test_model(model, test_loader):
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    all_labels = []
    all_predictions = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)

            test_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
```

```
            correct += (predicted == labels).sum().item()

            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    accuracy = correct / total
    avg_loss = test_loss / len(test_loader)

    # Вычисляем точность по классам
    from sklearn.metrics import classification_report
    print("\nОтчет по классификации:")
    print(classification_report(all_labels, all_predictions,
                                target_names=['muffin', 'chihuahua']))

    print(f'\nРезультаты на тестовой выборке:')
    print(f'Средняя Loss: {avg_loss:.4f}')
    print(f'Точность: {accuracy:.4f} ({correct}/{total})')

    return accuracy

print("\nТестирование модели...")
test_accuracy = test_model(model, test_loader)

def visualize_predictions(model, test_loader, num_images=5):
    model.eval()
    dataiter = iter(test_loader)
    images, labels = next(dataiter)

    images, labels = images[:num_images].to(device),
labels[:num_images].to(device)

    with torch.no_grad():
        outputs = model(images)
        _, predictions = torch.max(outputs, 1)
        probabilities = F.softmax(outputs, dim=1)

    images = images.cpu()
    predictions = predictions.cpu()
    labels = labels.cpu()
    probabilities = probabilities.cpu()

    fig, axes = plt.subplots(1, min(num_images, len(images)), figsize=(15,
6))
    class_names = ['muffin', 'chihuahua']

    for i in range(min(num_images, len(images))):
        ax = axes[i % min(num_images, len(images))]

        # Денормализуем изображение
        img = images[i].clone()
        for j in range(img.size(0)):
```

```
            img[j] = img[j] * std[j] + mean[j]
        img = img.permute(1, 2, 0).numpy()
        img = np.clip(img, 0, 1)

        ax.imshow(img)

        true_label = class_names[labels[i].item()]
        pred_label = class_names[predictions[i].item()]
        prob = probabilities[i][predictions[i]].item() * 100

        color = 'green' if predictions[i] == labels[i] else 'red'
        ax.set_title(f'Истина: {true_label}\nПредсказано:
{pred_label}\nУверенность: {prob:.1f}%',
                     color=color, fontweight='bold')
        ax.axis('off')

    plt.tight_layout()
    plt.show()

    return images, predictions, labels

print("\nВизуализация предсказаний на тестовой выборке:")
test_images, test_predictions, test_labels = visualize_predictions(model,
test_loader)

def analyze_errors(model, test_loader):
    model.eval()
    errors = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predictions = torch.max(outputs, 1)

            wrong_indices = (predictions !=
labels).nonzero(as_tuple=True)[0]

            for idx in wrong_indices:
                errors.append({
                    'image': images[idx].cpu(),
                    'true_label': labels[idx].cpu().item(),
                    'predicted_label': predictions[idx].cpu().item(),
                    'probabilities': F.softmax(outputs[idx],
dim=0).cpu().numpy()
                })

    return errors

print("\nАнализ ошибок...")
errors = analyze_errors(model, test_loader)
```

```
print(f"Количество ошибок: {len(errors)}")

if len(errors) > 0:
    print("\nПримеры ошибок классификации:")
    fig, axes = plt.subplots(1, min(5, len(errors)), figsize=(15, 6))
    class_names = ['muffin', 'chihuahua']

    for i in range(min(5, len(errors))):
        ax = axes[i % min(5, len(errors))]
        error = errors[i]

        # Денормализуем изображение
        img = error['image'].clone()
        for j in range(img.size(0)):
            img[j] = img[j] * std[j] + mean[j]
        img = img.permute(1, 2, 0).numpy()
        img = np.clip(img, 0, 1)

        ax.imshow(img)
        ax.set_title(f'Истина:
{class_names[error["true_label"]]}\nПредсказано:
{class_names[error["predicted_label"]]}',
                     color='red', fontweight='bold')
        ax.axis('off')
    plt.tight_layout()
    plt.show()

def save_model(model, filepath):
    torch.save({
        'model_state_dict': model.state_dict(),
        'model_architecture': model.__class__.__name__,
        'mean': mean,
        'std': std
    }, filepath)
    print(f"Модель сохранена в {filepath}")

save_model(model,
'/content/drive/MyDrive/simple_cnn_muffin_chihuahua.pth')

def load_model(filepath, model_class, *args, **kwargs):
    checkpoint = torch.load(filepath, map_location=device)
    model = model_class(*args, **kwargs).to(device)
    model.load_state_dict(checkpoint['model_state_dict'])
    model.eval()
    print(f"Модель загружена из {filepath}")
    return model, checkpoint.get('mean', None), checkpoint.get('std',
None)

# Проверяем загрузку модели
try:
```

```
    loaded_model, loaded_mean, loaded_std =
load_model('/content/drive/MyDrive/simple_cnn_muffin_chihuahua.pth',
SimpleCNN)
    print("Модель успешно загружена!")
except Exception as e:
    print(f"Ошибка при загрузке модели: {e}")
```