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-312

Дорогин Н.С.

Работу проверил: старший преподаватель кафедры ПМиК
Дементьева К.И.

Новосибирск 2025 г.

Введение (задание)

Цель: освоить на практике принципы построения, обучения и оценки нейронных сетей с использованием Pytorch.

Основная часть:

Для работы выбран датасет Happy-Sad Peoples

Задание 1:

Загрузим датасет и выведем основную информацию о нём.

```
import torch
from torchvision import datasets, transforms
import matplotlib.pyplot as plt


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

transform = transforms.Compose([
    transforms.Resize((64,64)),
    transforms.Grayscale(),
    transforms.ToTensor(),
   # transforms.Normalize((0.1307,),(0.3081,))
])



happy_sad = datasets.ImageFolder(
    root='/content/dataset/Data',
    transform=transform,
)

test = datasets.ImageFolder(
    root='/content/dataset/test',
    transform=transform,
)
```

```python
happy_sad = datasets.ImageFolder(
    root='/content/dataset/Data',
    transform=transform,
)

test = datasets.ImageFolder(
    root='/content/dataset/test',
    transform=transform,
)


print(f'Классы: {happy_sad.classes}')
print(f'Размер датасета для обучения: {len(happy_sad)}')
print(f'Размер датасета для теста: {len(test)}')
batch_size = 64
hsloader = torch.utils.data.DataLoader(happy_sad, batch_size=batch_size, shuffle=True)
tloader = torch.utils.data.DataLoader(test, batch_size=batch_size, shuffle=False)
```

```python
def show_samples(loader):
  dataiter = iter(loader)
  images, labels = next(dataiter)
  fig, axes = plt.subplots(1, 10, figsize=(12,5))
  for i in range(10):
    ax = axes[i % 10]
    ax.imshow(images[i].squeeze(), cmap='gray')
    ax.set_title(f'Label: {labels[i].item()}')
    ax.axis('off')

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


show_samples(hsloader)
```

```
Устройство: cuda
Классы: ['happy_pics', 'sad_pics']
Размер датасета для обучения: 173
Размер датасета для теста: 46
/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to
  warnings.warn(
```


```
(tensor([[[[0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
           [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
           [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
           ...,
           [0.1451, 0.1216, 0.0980,  ..., 0.1882, 0.0275, 0.0000],
           [0.1137, 0.1059, 0.1098,  ..., 0.2549, 0.1608, 0.0275],
           [0.1098, 0.1176, 0.0980,  ..., 0.2902, 0.2275, 0.1765]]],


         [[[0.9490, 0.9412, 0.9451,  ..., 0.6431, 0.3294, 0.1922],
           [0.9490, 0.9451, 0.9490,  ..., 0.5333, 0.2471, 0.1529],
           [0.9490, 0.9490, 0.9490,  ..., 0.4706, 0.2157, 0.1843],
           ...,
```

Задание 2:

Создаём сверточную нейронную сеть.

```python
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 16 * 16, 128)
        self.fc2 = nn.Linear(128, 2)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)

        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)

        x = x.view(-1, 64 * 16 * 16)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x
```

```python
model = SimpleCNN().to(device)
print("Архитектура 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'Общее количество обучаемых параметров: {total_params:,}')
```

```
Архитектура модели:
SimpleCNN(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=16384, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=2, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
Общее количество обучаемых параметров: 2,116,354
```

Обучаем её:

```python
def train_model(model, train_loader, test_loader, criterion, optimizer, epochs=5):
    model.train()
    train_losses = []
    train_accuracies = []
    test_accuracies = []

    for epoch in range(epochs):
        model.train()
        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 % 100 == 0:
                print(f'Эпоха [{epoch+1}/{epochs}], Батч [{batch_idx}/{total_batches}], '
                      f'Loss: {loss.item():.4f}')

        model.eval()
        test_accuracy = 0.0
        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                test_accuracy += calculate_accuracy(outputs, labels)

        epoch_loss = running_loss / total_batches
        epoch_train_accuracy = running_accuracy / total_batches
        epoch_test_accuracy = test_accuracy / len(test_loader)

        train_losses.append(epoch_loss)
        train_accuracies.append(epoch_train_accuracy)
        test_accuracies.append(epoch_test_accuracy)

        print(f'Эпоха [{epoch+1}/{epochs}] завершена -> '
              f'Средний loss: {epoch_loss:.4f}, '
              f'Точность (train): {epoch_train_accuracy:.4f}, '
              f'Точность (test): {epoch_test_accuracy:.4f}')
        print('-' * 50)

    return train_losses, train_accuracies, test_accuracies
```

Визуализация предсказания и их правильности:

```
def visualize_predictions(model, test_loader, num_images=10):
    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)


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


    fig, axes = plt.subplots(2, 5, figsize=(15, 6))

    for i in range(num_images):
        ax = axes[i // 5, i % 5]

        ax.imshow(images[i].squeeze(), cmap='gray')

        color = 'green' if predictions[i] == labels[i] else 'red'

        class_names = test_loader.dataset.classes
        pred_class = class_names[predictions[i].item()]
        true_class = class_names[labels[i].item()]

        ax.set_title(f'Предсказано: {pred_class}\nИстина: {true_class}',
                     color=color, fontweight='bold')
        ax.axis('off')

    plt.tight_layout()
    plt.show()

    return images, predictions, labels
```

Подсчёт предсказаний:

```python
def test_model(model, test_loader, criterion):
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0

    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()

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

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

    return accuracy
```

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

Применяем вышеописанный функционал:

```python
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = 0.001)


test_images, test_predictions, test_labels = visualize_predictions(model, tloader)
test_accurancy = test_model(model, tloader, criterion)
```

Результат предсказаний на тестовой выборке:

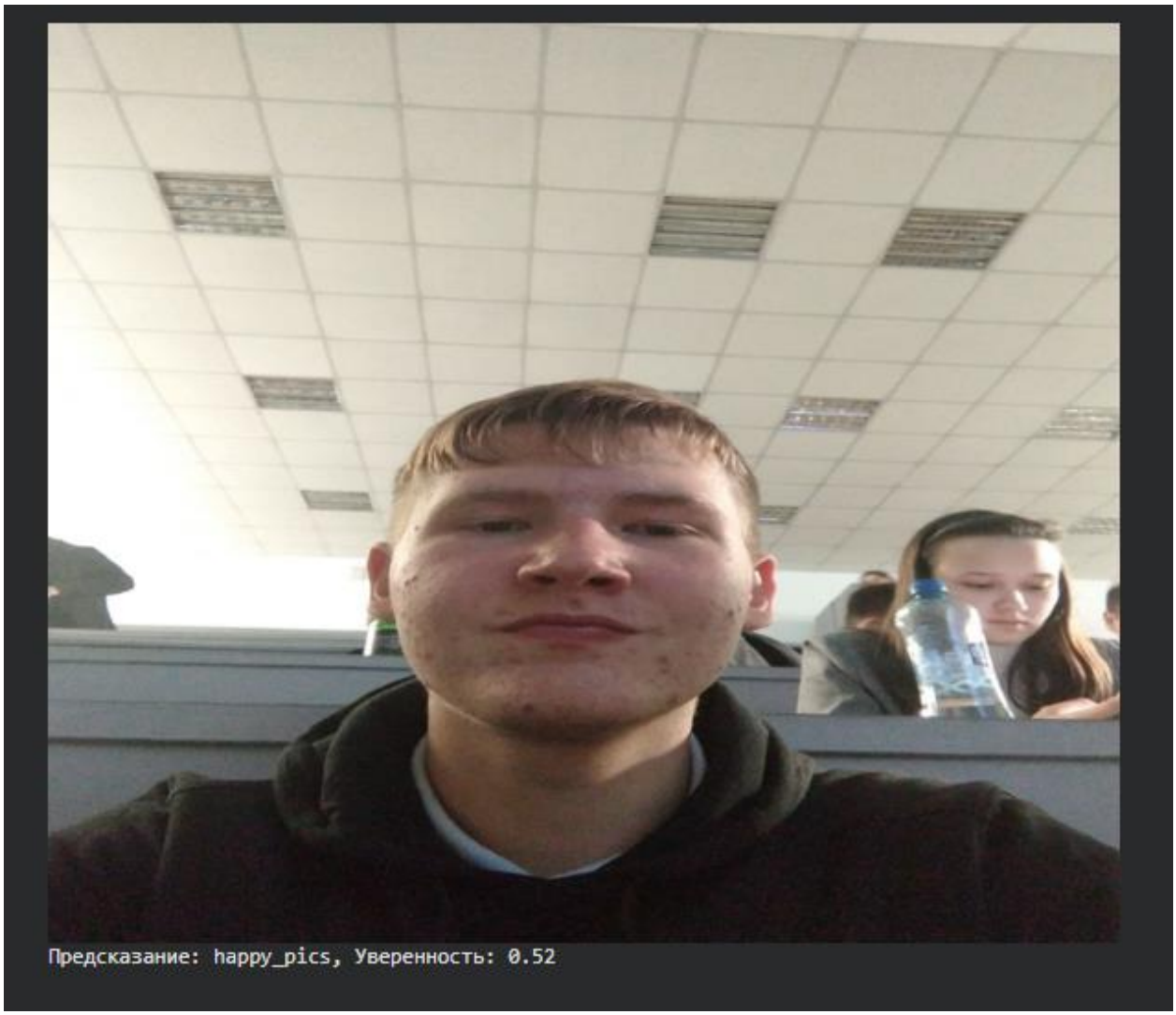
Предсказание изображений, не принадлежащих датасету:

```
print("\n\n\n\n")
from PIL import Image
from IPython.display import display

def predict_single_image(model, image_path):

    model.eval()

    transform = transforms.Compose([
        transforms.Resize((64, 64)),
        transforms.Grayscale(),
        transforms.ToTensor(),
    ])

    image = Image.open(image_path)
    display(image.resize((700, 600)))
    image_tensor = transform(image).unsqueeze(0)
    image_tensor = image_tensor.to(device)

    with torch.no_grad():
        output = model(image_tensor)
        probabilities = torch.softmax(output, dim=1)
        predicted_class = torch.argmax(output, 1).item()
        confidence = probabilities[0][predicted_class].item()

    class_names = ['happy_pics', 'sad_pics']
    predicted_label = class_names[predicted_class]

    return predicted_label, confidence
```

```
shrek = "/content/dataset/custom_pics/shrek.jpg"
harold = "/content/dataset/custom_pics/harold.webp"
me = "/content/dataset/custom_pics/me.jpg"

s_prediction, s_confidence = predict_single_image(model, shrek)
print(f"Предсказание: {s_prediction}, Уверенность: {s_confidence:.2f}\n\n")

h_prediction, h_confidence = predict_single_image(model, harold)
print(f"Предсказание: {h_prediction}, Уверенность: {h_confidence:.2f}\n\n")

m_prediction, m_confidence = predict_single_image(model, me)
print(f"Предсказание: {m_prediction}, Уверенность: {m_confidence:.2f}\n\n")
```

Результат предсказаний изображений, не принадлежащих датасету:


Предсказание: happy_pics, Уверенность: 0.52



Задание 3:

На этапе обучения и тестирования возникли следующие проблемы:

1. Все изображения на тестовой выборке, кроме одного принадлежат 0 классу, что затрудняет оценку точности работы нейросети.
2. С каждым прогоном программы предсказания нейросети меняются разительно. Она может на первом прогоне предсказать все happy и будет почти права (потому что на тестовой выборке почти все и есть happy), а на следующем прогоне может предсказать все sad и попадёт только в одно изображение, то самое. На некоторых прогонах соотношение может быть более нейтральным, но точность при этом ниже 50%. Что забавно, не датасетные изображения нейросеть с точностью 10/42 на тестовой выборке предсказала лучше всего.

Отчёт:

Ссылка на GoogleCollab: https://colab.research.google.com/drive/1InYZSFYOxjGShodyJXLZvcIH-hd156SS#scrollTo=gqiWNBjtP8A1