Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Южно-Уральский государственный университет»
(национальный исследовательский университет)
Институт естественных и точных наук
Кафедра прикладной математики и программирования

Отчет по лабораторной работе № 2
по дисциплине «Современные нейросетевые технологии»

Авторы работы
Студент группы ЕТ-122
_____/ Матвеева А.В.
«____» _____2025 г.

Руководитель работы,
_____/ Кичеев Д.М.
«____» _____2025 г.

Челябинск 2025

Задание:

Цель: получить базовые навыки работы с одной из библиотек глубокого обучения (Caffe, Torch, TensorFlow или MXNet на выбор) на примере полностью связанных нейронных сетей.

Задачи:

Выполнение практической работы предполагает решение ***следующих задач***:

1. Выбор библиотеки для выполнения практических работ курса.
2. Установка выбранной библиотеки на кластере.
3. Проверка корректности установки библиотеки. Разработка и запуск тестового примера сети, соответствующей логистической регрессии, для решения задачи классификации рукописных цифр набора данных MNIST (пример разобран в лекционных материалах).
4. Выбор практической задачи компьютерного зрения для выполнения практических работ.
5. Разработка программ/скриптов для подготовки тренировочных и тестовых данных в формате, который обрабатывается выбранной библиотекой.
6. Разработка нескольких архитектур полностью связанных нейронных сетей (варьируются количество слоев и виды функций активации на каждом слое) в формате, который принимается выбранной библиотекой.
7. Обучение разработанных глубоких моделей.
8. Тестирование обученных глубоких моделей.
9. Сделать вывод относительно разработанных архитектур.
10. Подготовка отчета, содержащего минимальный объем информации по каждому этапу выполнения работы.

Выполнение задач:

1. В ходе данного проекта я освоила базовые навыки работы с библиотекой глубокого обучения PyTorch на примере построения и обучения полностью связанных нейронных сетей для задачи бинарной классификации.

Полностью связная нейронная сеть может быть описана последовательностью слоев с весами. Например, для сети с одним скрытым слоем:

$$\mathbf{h} = f(\mathbf{X} \cdot \mathbf{W} + \mathbf{b})$$

$$\mathbf{y}_{\text{pred}} = g(\mathbf{h} \cdot \mathbf{W}_{\text{out}} + b_{\text{out}})$$

где:

- **W** — матрица весов скрытого слоя.
- **b** — вектор смещений скрытого слоя.
- **W**out — вес для выходного слоя.
- *b*out — смещение для выходного слоя.
- *f* — функция активации для скрытого слоя (например, ReLU).
- *g* — функция активации для выходного слоя, используемая для бинарной классификации (обычно это сигмоид):

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Для бинарной классификации используется бинарная кросс-энтропийная функция потерь:

$$L(\mathbf{y}, \mathbf{y}_{\text{pred}}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(y_{\text{pred},i}) + (1 - y_i) \log(1 - y_{\text{pred},i})]$$

- где:
- y — истинная метка для образца i.
- $y$pred — предсказанная вероятность принадлежности к классу 1 для образца i.

Обучение сети заключается в минимизации функции потерь с помощью метода градиентного спуска или его вариантов (например, Adam):

$$\theta \leftarrow \theta - \eta \nabla L(\theta)$$

где θ — параметры модели (веса и смещения), η — скорость обучения, а $\nabla L(\theta)$ — градиент функции потерь по параметрам.

2. Были использованы данные подмножество набора данных Food-101. Боссард, Лукас, Матье Гийомен и Люк Ван Гул. «Food-101 – Mining Discriminative Components with Random Forests». Изображения разделены на две папки: pizza и not_pizza, имеющее равное количество изображений (по 983 шт.). Все изображения были масштабированы таким образом, чтобы максимальная длина

стороны составляла 512 пикселей. Содержание папок соответствует их названию.

Данные разделены:

Количество обучающих изображений: 1376

Количество валидационных изображений: 294

Количество тестовых изображений: 296

3. Я использую метрику качества Accuracy, которая определяется следующим образом:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

где:

- *TP* (True Positives) — количество истинных положительных предсказаний (правильно классифицированные объекты класса 1).
- *TN* (True Negatives) — количество истинных отрицательных предсказаний (правильно классифицированные объекты класса 0).
- *FP* (False Positives) — количество ложных положительных предсказаний (объекты класса 0, ошибочно классифицированные как класс 1).
- *FN* (False Negatives) — количество ложных отрицательных предсказаний (объекты класса 1, ошибочно классифицированные как класс 0).

4. Данные мной были скачаны с сайта kaggle, хранятся локально в двух папках файлами в формате . Jpg.
5. Я использовала создание датасета с помощью встроенной функции torch.utils.data.Dataset,, в нее подавались пути к изображениям и изображения, а на выходе получили Кортеж (features, label): где features — это тензор с признаками, a label — это тензор с меткой класса.

   После этого я использовала загрузчик данных DataLoder. В него подается полученный на предыдущем этапе датасет, а в итоге получаем пакеты данных (batch), состоящие из кортежей (features, labels) для каждого образца в пакете.
6. Мной была разработана функция для обучения нейронной сети:

```python
def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs):
    train_losses = []
    val_losses = []
    val_accuracies = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs.squeeze(), labels.float())
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        epoch_loss = running_loss / len(train_loader)
        train_losses.append(epoch_loss)
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}')

        val_loss, accuracy = validate_model(model, val_loader, criterion)
        val_losses.append(val_loss)
        val_accuracies.append(accuracy)

    return train_losses, val_losses, val_accuracies
```

В приведенной функции происходит процесс обучения модели на тренировочном наборе данных с последующей валидацией на валидационном наборе.

1) Инициализируем списки для хранения потерь и метрики качества.
2) Переводим модель в режим обучения, создаем переменную для накопления потерь в текущей эпохе
3) Проходим циклом по каждому батчу. В цикле обнуляем градиенты, получаем предсказания, вычисляем потери, вычисляем градиенты на основе потерь, обновление параметров модели на основе градиентов, производим накапливание потерь.
4) Вычисляем средние потери за эпоху и сохраняет их.
5) Тестирует модель на валидационных данных и сохраняет результат.
6) Пункты 2-5 делаются по количеству эпох.

7. Мной были разработаны три нейронные сети.

- Простая модель. Один скрытый линейный слой, функция активации скрытого слоя - RELU.

```python
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(512 * 512 * 3, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 1)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return torch.sigmoid(x)
```

Визуальная схема:

Входное изображение (512 x 512 x 3)

|

Преобразование (Flatten) -> Вектор длины 786432

|

Полносвязный слой (fc1): 786432 -> 128

|

ReLU (активация)

|

Полносвязный слой (fc2): 128 -> 1

|

Sigmoid (активация)

|

Выходное значение (0 или 1)

- Средняя модель. Четыре скрытых слоя, функция активации скрытого слоя - тангенс.

```python
class ComplexNN(nn.Module):
    def __init__(self):
        super(ComplexNN, self).__init__()
        self.fc1 = nn.Linear(512 * 512 * 3, 512)
        self.tanh = nn.Tanh()
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 1)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.tanh(x)
        x = self.fc2(x)
        x = self.tanh(x)
        x = self.fc3(x)
        x = self.fc4(x)
        return torch.sigmoid(x)
```

Визуальная схема

Входное изображение (512 x 512 x 3)

|

Преобразование (Flatten) -> Вектор длины 786432

|

Полносвязный слой (fc1): 786432 -> 512

|

Tanh (активация)

|

Полносвязный слой (fc2): 512 -> 256

|

Tanh (активация)

|

Полносвязный слой (fc3): 256 -> 128

|

Полносвязный слой (fc4): 128 -> 1

|

Sigmoid (активация)

|

Выходное значение (0 или 1)

- Сложная модель. Четыре скрытых слоя, используем нормализацию, обнуление нейронов и LeakyReLU в качестве активации.

```python
class ReducedNN(nn.Module):
    def __init__(self):
        super(ReducedNN, self).__init__()
        self.fc1 = nn.Linear(512 * 512 * 3, 512)
        self.bn1 = nn.BatchNorm1d(512)
        self.fc2 = nn.Linear(512, 256)
        self.bn2 = nn.BatchNorm1d(256)
        self.fc3 = nn.Linear(256, 128)
        self.bn3 = nn.BatchNorm1d(128)
        self.fc4 = nn.Linear(128, 1)
        self.dropout = nn.Dropout(p=0.5)
        self.relu = nn.LeakyReLU(negative_slope=0.01)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.bn2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        x = self.bn3(x)
        x = self.relu(x)
        x = self.fc4(x)
        return torch.sigmoid(x)
```

Входное изображение (512 x 512 x 3)

|

Преобразование (Flatten) -> Вектор длины 786432

|

Полносвязный слой (fc1): 786432 -> 512

|

Tanh (активация)

|

Полносвязный слой (fc2): 512 -> 256

|

Tanh (активация)

|

Полносвязный слой (fc3): 256 -> 128

|

Полносвязный слой (fc4): 128 -> 1

|

Sigmoid (активация)

|

Выходное значение (0 или 1)

8. Результаты:

Модель	Время обучения, сек.	Количество скрытых слоев	Функция активации	Accuracy
SimpleNN()	4359	1	ReLU	62.50%
ReducedNN()	17436	4	Tanh	65.54%
ComplexNN(	19180	4	LeakyReLU	70.27%