1. Постановка задачи.

Для выполнения этой работы тема выбиралась в свободном формате с условием изучения и открытия для себя понятия нейронных сетей, но так как курсовая работа по предмету “Программная инженерия” требовалось создать какой-то продукт с интерфейсом и возможностью работы с пользователем. Из этого требования был придумана постановка для курсовой работы: “Обучение нейронной сети на наборе MNIST и создание программного обеспечения для взаимодействия с моделью”.

2. Нейронная сеть для распознавания набора MNIST.

2.1. Описание набора MNIST.

Набор данных MNIST (Modified National Institute of Standards and Technology) представляет собой одну из самых стандартных для обучения и тестирования наборов искусственного интеллекта. Данный набор состоит из 70000 изображений рукописных цифр от 0 до 9. Набор разделен на обучающую и тестовую множества, составляющие 60000 и 10000 соответственно. Каждое изображение имеет стандартный вид 28 на 28 пикселей, имеющих черно-белый окрас. В приложение 1 можно просмотреть как выглядят цифры в MNIST наборе.

2.2. Описание модели.

Модель написана на одном из самых популярных фреймворков для работы с нейронными сетями PyTorch на версии 2.5.1.

Каждое изображение 28 на 28 можно представить как нейроны, имеющие свои веса. Для изучения нейронных сетей было принято решение использовать разные слои, чтобы исследовать. Нейронную сеть можно представить как граф, где каждый слой соединен с последующим с вершинами. Этот граф называется граф решений. Модель представляет:

Слой пакетной нормализации: выполняет выравнивание распределения входных данных на каждом слое, нормализуя их к среднему значению и единичной дисперсии. Это ускоряет процесс обучения и обеспечивает стабильность градиентного спуска, так как уменьшает вероятность исчезающего или взрывного градиента. Пакетная нормализация также снижает чувствительность модели к выбору начальных значений весов.

Сверточная сеть: в данной модели применены, чтобы извлечь пространственные признаки из изображений. Несмотря на то, что для набора данных MNIST свёртки не являются строго необходимыми из-за ограниченной сложности изображений, они позволяют модели лучше адаптироваться к потенциальным искажениям данных. Использование фильтров размером 3x3

и активации ReLU обеспечивает эффективное выделение признаков, релевантных для задачи классификации.

Макспулинг: используется для уменьшения размерности выходных данных, выбирая максимальное значение в окне размером 2x2. Это сокращает вычислительные затраты и повышает устойчивость модели, так как сохраняются наиболее значимые признаки. Макспулинг также способствует борьбе с переобучением за счёт обобщения информации.

Полносвязные слои: Модель включает три полносвязных слоя. Первый слой отвечает за преобразование пространственных признаков в одномерный вектор для дальнейшей обработки. Второй слой интерпретирует эти признаки, выявляя сложные нелинейные зависимости. Третий, выходной слой, представляет собой 10 нейронов с активацией Softmax, которые формируют вероятности принадлежности к классам.

Так же были использованы слои активация, обнуляющие отрицательные нейроны и слои случайного обнуления некоторой части нейронов для того, чтобы модель не переобучалась и не сильно была чувствительна.

Для оценки работы модели было использована мера оценки кросс-энтропия. Потери — это величина, характеризующая расхождение между предсказанными моделью значениями и истинными метками. В данной работе для вычисления потерь используется функция кросс-энтропии. Кросс-энтропия измеряет разницу между двумя вероятностными распределениями, одним из которых являются предсказания модели, а другим — истинные метки. Формула для вычисления кросс-энтропии записывается как:

$$H(p, q) = -\sum_{i=0}^{n} p_i \log(q_i)$$

p_i – истинное распределение вероятности.

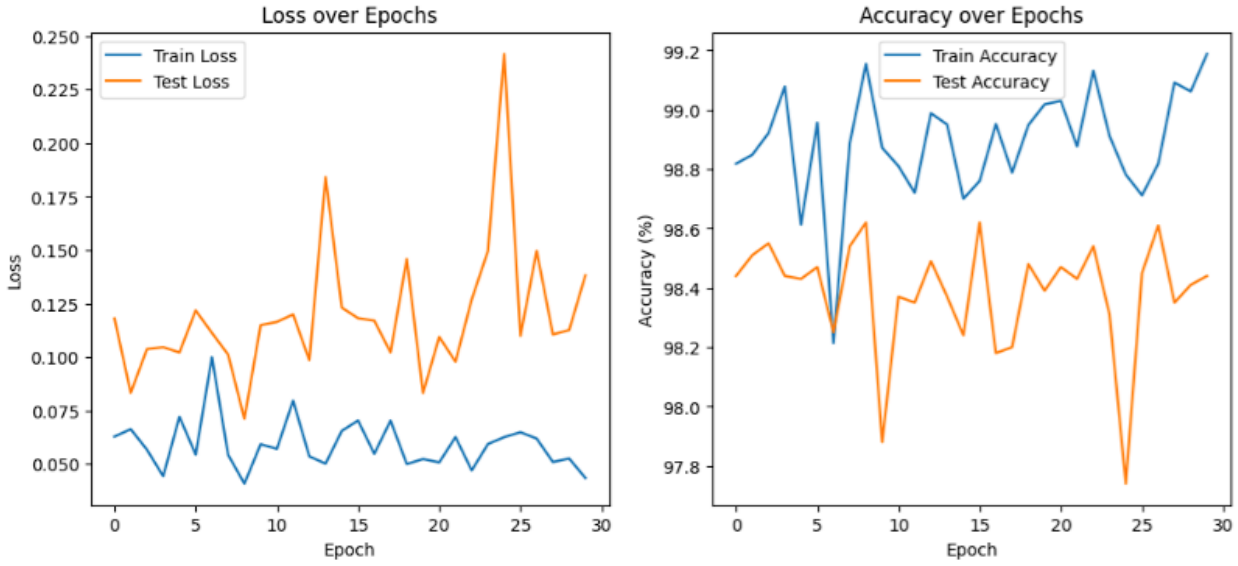
q_i – распределение вероятностей, предсказание моделью.

n – это количество классов.

Надо заметить, что p и q вектора из 10 элементов. Для истинного значение 1 вектор p представляет собой p = {0, 1, 0, 0, 0, 0, 0, 0, 0, 0,}. А q – это вектор вероятностей, полученных предсказанием модели. Создание модели и его код можно посмотреть в приложении 2.

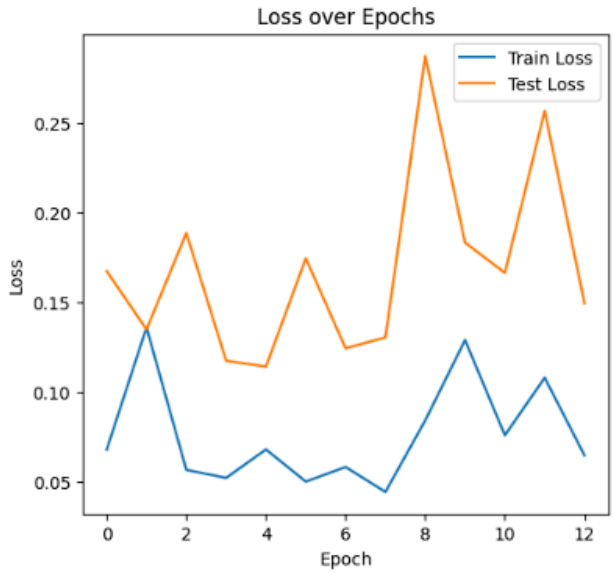
2.3. Обучение модели.

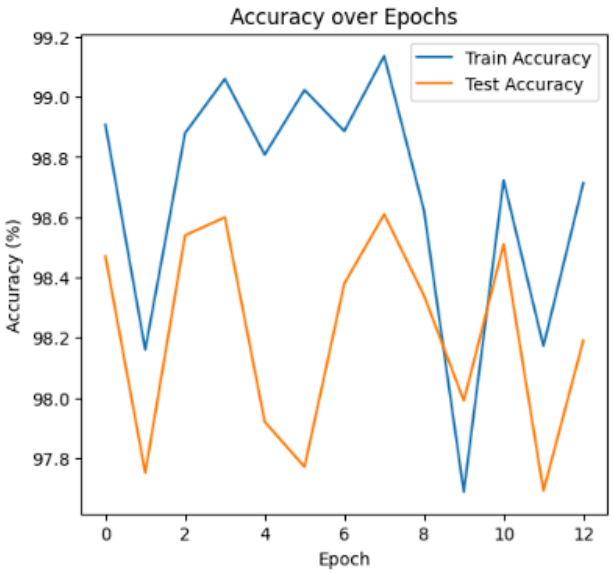
Обучение модели – это пересчет весов, через которые проходят нейроны. А эпоха – это прогонка модели на обучающих наборах и расчет точности и потери модели. Так как предсказание можно рассматривать как функцию от нескольких переменных имеющий на выход меньшего размера для перерасчета весов используется градиентный спуск. Из теории векторного анализа вектор градиентного спуска указывает точку наибольшего возрастания, а отрицание этого вектора на точку наискорейшего спуска. Для начала было выбрано 30 эпох для обучения модели.



После обучения можно заметить, что модель довольно точна, но можно заметить, что после 10-12 эпохи модель начала переобучатся и стала не сильно уверенной. Это можно увидеть из-за того, что график потерь на тестовых наборах начал расти, а на тренировочных наборах модель стала очень точной. Хотя из теории машинного обучения не рекомендуется на основании графиков потерь тестового и тренировочного набора делать выводы по следующим эпохам, так как образуется неявная связь между тестовым и тренировочным набором. По рекомендациям нужно делать кросс-валидацию, чтобы существовал всегда набор, который не зависит от тренировочного набора. Но так модель имеет очень большую точность и маленькие

потери, уменьшим количество эпох до 13.






```
Epoch [1/13], Train Loss: 0.0678, Train Accuracy: 98.91%, Test Loss: 0.1673, Test Accuracy: 98.47%
Epoch [2/13], Train Loss: 0.1359, Train Accuracy: 98.16%, Test Loss: 0.1348, Test Accuracy: 97.75%
Epoch [3/13], Train Loss: 0.0566, Train Accuracy: 98.88%, Test Loss: 0.1885, Test Accuracy: 98.54%
Epoch [4/13], Train Loss: 0.0521, Train Accuracy: 99.06%, Test Loss: 0.1174, Test Accuracy: 98.60%
Epoch [5/13], Train Loss: 0.0679, Train Accuracy: 98.81%, Test Loss: 0.1141, Test Accuracy: 97.92%
Epoch [6/13], Train Loss: 0.0500, Train Accuracy: 99.02%, Test Loss: 0.1744, Test Accuracy: 97.77%
Epoch [7/13], Train Loss: 0.0581, Train Accuracy: 98.89%, Test Loss: 0.1243, Test Accuracy: 98.38%
Epoch [8/13], Train Loss: 0.0443, Train Accuracy: 99.14%, Test Loss: 0.1304, Test Accuracy: 98.61%
Epoch [9/13], Train Loss: 0.0842, Train Accuracy: 98.62%, Test Loss: 0.2872, Test Accuracy: 98.34%
Epoch [10/13], Train Loss: 0.1289, Train Accuracy: 97.69%, Test Loss: 0.1832, Test Accuracy: 97.99%
Epoch [11/13], Train Loss: 0.0758, Train Accuracy: 98.72%, Test Loss: 0.1663, Test Accuracy: 98.51%
Epoch [12/13], Train Loss: 0.1079, Train Accuracy: 98.17%, Test Loss: 0.2566, Test Accuracy: 97.69%
Epoch [13/13], Train Loss: 0.0646, Train Accuracy: 98.71%, Test Loss: 0.1493, Test Accuracy: 98.19%
```

Можно заметить, что графики стали намного лучше. В будущем эту модель будем использовать для работы. Код можно обучения можно увидеть в приложении 3.

3. Создание сайта для рисования рукописных цифр.

Создан сайт для рисования рукописных цифр с использованием HTML, CSS и JavaScript. На странице размещен элемент холст для рисования изображений, кнопки для отправки изображения на сервер и очистки холста, а также таблица для отображения предсказанных результатов. При рисовании на холсте пользователи могут свободно вводить цифры. После завершения рисования, изображение отправляется на сервер с помощью кнопки "Отправить изображение". Сервер обрабатывает изображение, распознает цифру и возвращает прогноз вместе с вероятностями для каждой из цифр (от 0 до 9), которые отображаются в таблице на странице. В таблице отображаются как вероятность для каждой цифры, так и предсказанный класс.

Для рисования на странице был использован объект <canvas>, размером 280 на 280 пикселей. Рисование происходило с помощью отслеживания нажатий и зажатий левой кнопки мыши. При удержании кнопки мыши, пользователь мог рисовать на канвасе, и процесс рисования был реализован с использованием линий шириной 22 пикселя. Эти линии рисуются путем

последовательного движения мыши, где каждый новый отрезок соединяет предыдущую точку с текущей.

Для отправки изображения на сервер используется функция, которая извлекает данные о содержимом канваса через его контекст. Изображение представляется в виде массива данных, из которого каждый 4-й элемент отвечает за значение прозрачности (альфа-канал) пикселей. С помощью этого массива происходит перебор каждого элемента, и значения прозрачности используются для формирования двумерной матрицы, представляющей изображение в черно-белом формате. Эта матрица затем отправляется на сервер в формате JSON для дальнейшей обработки и распознавания.

4. Создание сервера для рисования.

Серверная часть приложения построена с использованием FastAPI, фреймворка для создания веб-приложений на Python. Он принимает POST-запросы по маршруту /upload-image, содержащие изображение в виде матрицы, которое было отправлено с клиентской стороны. Изображение, полученное с сайта, имеет размер 280 на 280 пикселей, и для его обработки выполняется предварительное сжатие: каждый блок размером 10 на 10 пикселей усредняется с коэффициентом затемнения, что помогает уменьшить размер данных и сохранить основные черты изображения. После этого данные передаются в заранее загруженную модель нейронной сети, которая делает предсказание, выводя не только сам класс (цифру), но и вектор вероятностей для каждого возможного класса.

Модель использует функцию softmax для нормализации выходных данных в вероятностное распределение, что позволяет точно определить наибольшую вероятность для конкретного класса. Сервер отправляет этот результат обратно на клиент в виде JSON-ответа, который включает предсказанный класс и вероятности для всех классов. Такой подход позволяет эффективно и быстро обрабатывать изображения, полученные с веб-интерфейса, и предоставлять пользователю точное предсказание.

Цифра	0	1	2	3	4	5	6	7	8	9
Вероятность	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

5. Подведение итогов и вывод.

5.1. Оценка результатов.

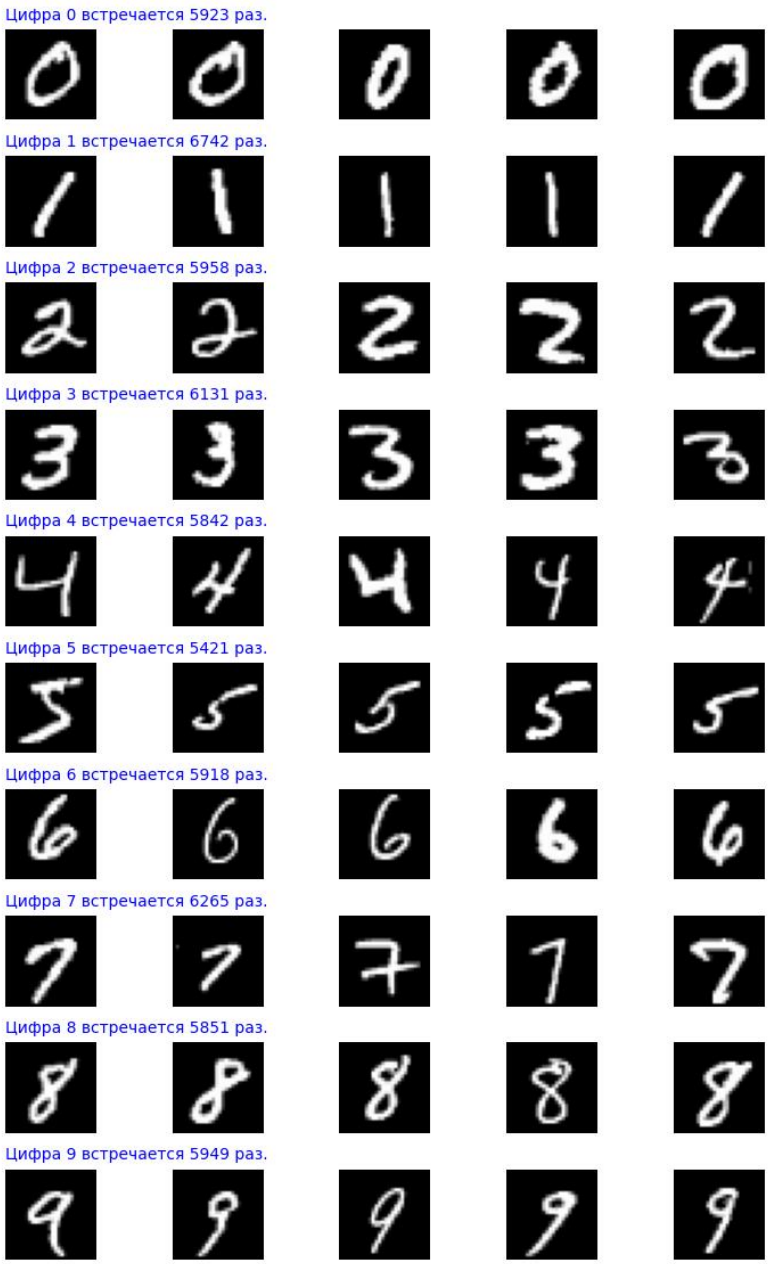
В целом, результаты работы приложения являются вполне удовлетворительными. Модель справляется с распознаванием рукописных цифр, а сервер успешно обрабатывает изображения и возвращает предсказания. Однако стоит отметить, что датасет MNIST, на основе которого была обучена модель, не предназначен для распознавания цифр, написанных на краях или смещённых от центра. В таких случаях точность модели может существенно снизиться, и результаты могут быть не такими точными, как при стандартных примерах, где цифры написаны в центре изображения. В остальном, модель демонстрирует хорошие результаты и адекватно справляется с большинством изображений, выполненных в стандартных условиях.

5.2. Будущие возможные улучшения.

Первым пунктом является увеличение возможностей расписывания рукописных символов. Вторым улучшением можно отметить улучшения фронтендерского составляющего интерфейса и поднятия сервера на фреймворках как react.

6. Приложение

6.1. Приложение 1.



6.2 Приложение 2.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.ep = 0
        self.bn = nn.BatchNorm2d(1)
        self.conv = nn.Conv2d(1, 16, kernel_size=3)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.drop = nn.Dropout(0.3)
```

```
        self.fc0 = nn.Linear(2704, 512)
        self.fc1 = nn.Linear(512, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)
        nn.init.kaiming_normal_(self.conv.weight, mode='fan_out', nonlinearity='relu')
        nn.init.xavier_normal_(self.fc0.weight)
        nn.init.xavier_normal_(self.fc1.weight)
    def forward(self, x):
        x = self.bn(x)
        x = self.conv(x)
        x = self.pool(x)
        x = self.drop(x)
        x = F.relu(x)
        x = x.view(-1, 2704)
        x = self.fc0(x)
        x = F.relu(x)
        x = self.fc1(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        out = x
        return out
```

6.3 Приложение 3.

```
def train(net, epochs, train_loader, test_loader):
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(net.parameters(), lr=0.01)

    train_losses = []
    test_losses = []
    train_accuracies = []
    test_accuracies = []

    for ep in range(epochs):
        net.train()
        train_loss = 0
        correct_train = 0
```

```
total_train = 0
for batch_n, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    output = net(data)
    loss = criterion(output, target)
    train_loss += loss.item() * data.size(0)
    loss.backward()
    optimizer.step()

    _, predicted = output.max(1)
    total_train += target.size(0)
    correct_train += predicted.eq(target).sum().item()

net.eval()
test_loss = 0
correct_test = 0
total_test = 0
with torch.no_grad():
    for data, target in test_loader:
        output = net(data)
        loss = criterion(output, target)
        test_loss += loss.item() * data.size(0)

        _, predicted = output.max(1)
        total_test += target.size(0)
        correct_test += predicted.eq(target).sum().item()

train_loss /= len(train_loader.dataset)
test_loss /= len(test_loader.dataset)
train_accuracy = 100. * correct_train / total_train
test_accuracy = 100. * correct_test / total_test

train_losses.append(train_loss)
test_losses.append(test_loss)
train_accuracies.append(train_accuracy)
test_accuracies.append(test_accuracy)
```

```
    print(f'Epoch [{ep+1}/{epochs}], '
          f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%, '
          f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(range(epochs), train_losses, label='Train Loss')
plt.plot(range(epochs), test_losses, label='Test Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(epochs), train_accuracies, label='Train Accuracy')
plt.plot(range(epochs), test_accuracies, label='Test Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.legend()

plt.show()
```