Национальный исследовательский университет «МЭИ»

Институт информационных и вычислительных технологий

Кафедра ПМ ИИ

Отчет по курсовой работе

по дисциплине: «Программная инженерия»

Тема курсовой работы

«Разработка веб-серверного приложения для распознавания рукописных цифр»

Группа: А-05-22

Студент:

Павлов Н. М.

Москва 2024

Оглавление

Раздел 1. Введение.

1.1. Постановка задачи.

Данная работа посвящена разработке веб-серверного приложения для решения задачи распознавания рукописных цифр. Архитектура приложения состоит из трех ключевых компонентов: фронтенда, бэкенда и модели нейронной сети, которые работают в тесной интеграции для обеспечения пользовательского опыта.

Фронтенд представляет собой веб-интерфейс, который предоставляет пользователю холст для рисования рукописных цифр. Этот компонент отвечает за сбор пользовательских данных и их отправку на серверную часть через HTTP-запросы.

Бэкенд реализован с использованием фреймворка FastAPI. Основной задачей бэкенда является обработка данных, присланных с фронтенда, передача их в модель для анализа и отправка обратно результатов предсказания.

Модель нейронной сети обучена на наборе данных MNIST. Она принимает на вход одномерное представление изображения, преобразованного из двумерного массива пикселей. Выход модели представляет собой вероятности принадлежности изображения к каждому из возможных классов (цифры от 0 до 9).

1.2. Введение в общую теорию нейронных сетей.

Искусственные Нейронные Сети (ИНС) – это математические модели, созданные по аналогии с биологическими нейронными сетями. ИНС способны моделировать и обрабатывать нелинейные отношения между входными и выходными сигналами. Адаптивное взвешивание сигналов между искусственными нейронами достигается благодаря обучающемуся алгоритму, считывающему наблюдаемые данные и пытающемуся улучшить результаты их обработки. Одной из главных принципов нейронной сети является способность обучаться на наблюдаемых примерах и перестраивать свои нейроны для достижения поставленной цели [2].

Нейронная сеть состоит из 3 уровней. Первым уровнем является входной слой. Он получает информацию из окружающей среды. Следующим уровнем является скрытые слои. В скрытых слоях происходит большая часть обработки информации. Последним уровнем является выходной слой. Выход нейронной сети зависит от веса связей между нейронами в разных слоях. Каждый вес указывает на относительную важность определённого нейрона в разных слоях. Вес нейрона строится от весов предыдущих нейронов [2].

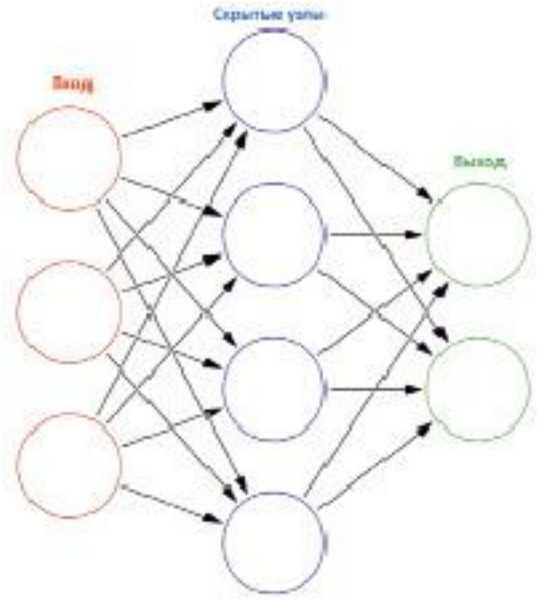


Рисунок 1. Пример нейронной сети [2].

Опишем базовую математическую модель нейронной сети. Каждый нейрон на слое l получается на входе данные от нейронов предыдущего слоя. Пусть a_i^{l-1} – это активация нейрона на i на слое $l-1$, а w_{ji}^l – это вес связи нейрона i на предыдущем слое к нейрону j текущего слоя l. Входные данные на этом нейроне будут вычисляться как взвешенная сумма выходов нейронов предыдущего слоя и добавления смешения b_j^l:

$$a_i^l = \sum_i^{N_{l-1}} \text{a}\, w_{ji}^l a_i^{l-1} + b_j^l$$

Для выходного слоя нейронной сети, нейроны равны признакам. То есть если есть множество входных признаков $x_1, x_2, x_3, \dots, x_n$, то для нейронов первого слоя используется: $a_1^1 = x_1 a_2^1 = x_2 a_3^1 = x_3. \dots, a_n^1 = x_n$ [1].

Одним из способов использования нейронной сети является распознавание картинок. Нейронная сеть интерпретирует изображение как совокупность визуальных данных, где каждый пиксель может быть представлен как элемент входного слоя. На первых этапах обработки сеть выделяет базовые признаки, такие как контуры, границы и текстуры. Эти признаки передаются в последующие слои, где происходит их комбинация и обобщение, что позволяет выделить более сложные структуры, например, формы или объекты. В конечном итоге сеть формирует выход, который представляет собой результат классификации,

сегментации или другой задачи, связанной с изображением [2].

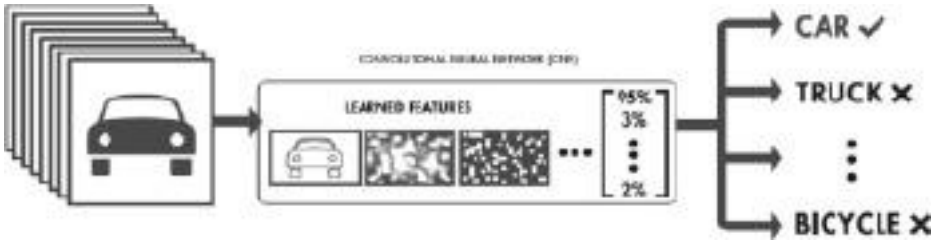


Рисунок 2. Пример работы нейронной сети [2].

Раздел 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.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
       BatchNorm2d-1            [-1, 1, 28, 28]               2
            Conv2d-2           [-1, 16, 26, 26]             160
         MaxPool2d-3           [-1, 16, 13, 13]               0
           Dropout-4           [-1, 16, 13, 13]               0
            Linear-5                  [-1, 512]       1,384,960
            Linear-6                  [-1, 128]          65,664
            Linear-7                   [-1, 64]           8,256
            Linear-8                   [-1, 10]             650
================================================================
Total params: 1,459,692
Trainable params: 1,459,692
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.14
Params size (MB): 5.57
Estimated Total Size (MB): 5.71
----------------------------------------------------------------
```

Рисунок 3. Слои нейронной сети

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.

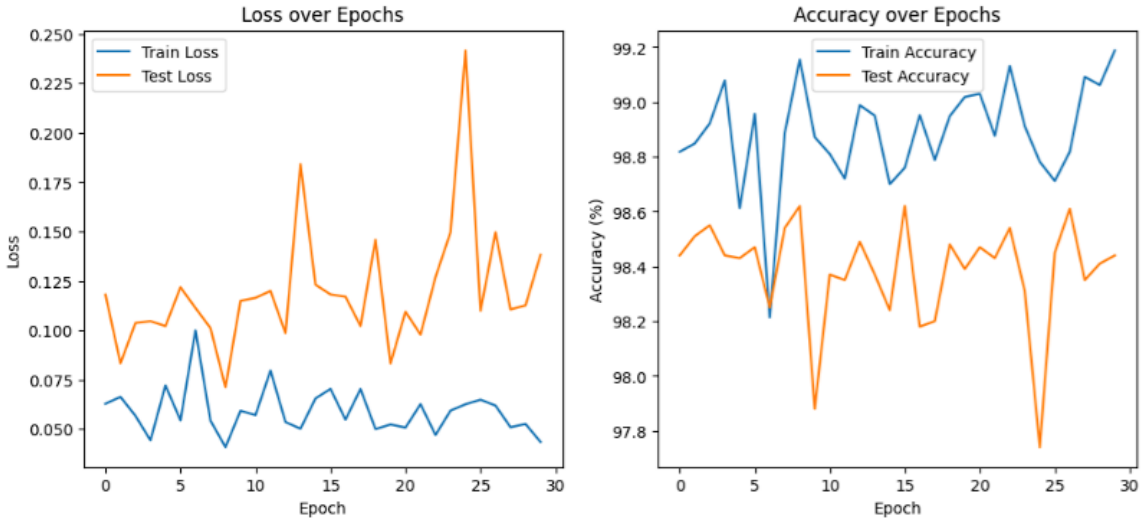


Рисунок 4. График обучения модели 1.

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.

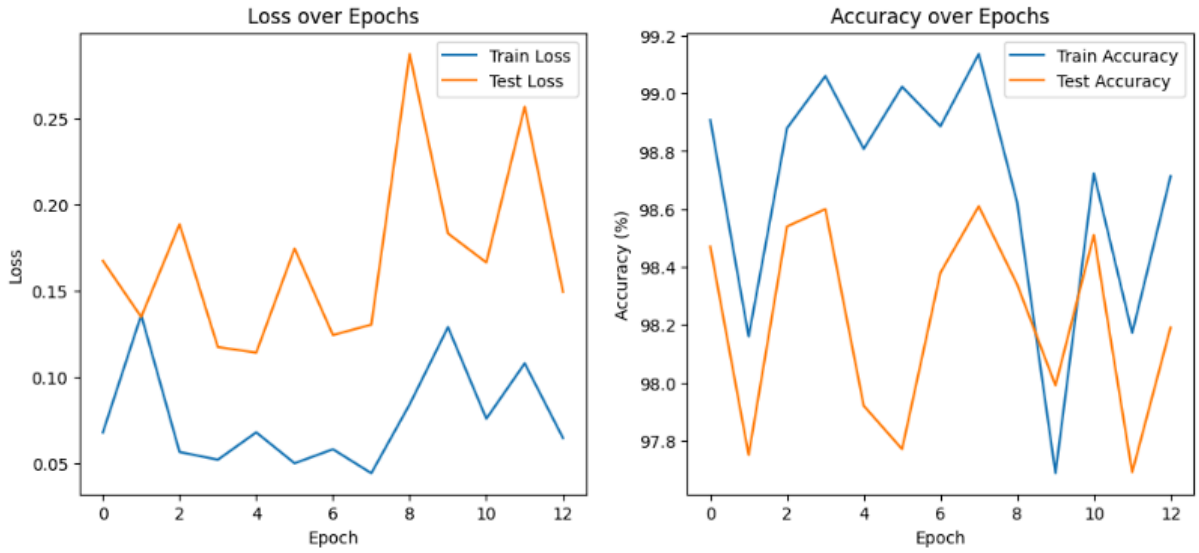


Рисунок 5. График обучения модели 2.

```
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%
```

Рисунок 6. Потери и точность каждой эпохи.

Можно заметить, что графики стали намного лучше и модель себя ведет одинаково хорошо на тестовых и тренировочных наборах. Можно заметить, что модель является эффективной на MNIST наборе с точностью 98% и с потерями 0.0646. Далее будем использовать эту модель для распознавания рукописных цифр.

Раздел 3. Создание веб-интерфейса для рисования рукописных цифр.

Создан с веб-интерфейс 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.

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.

Отправка на сервер совершается по нажатию кнопки. При нажатии кнопки достается из канваса цвет каждого пикселя и записывается в матрицу. В конечном итоге получится матрица 280 на 280, имеющая значения от 0 до 255, где 0 означает, что пиксель имеет белый цвет, а 255 означает черный. После матрица преобразуется в тип в JSON, где “matrix” ключ, а значение — это матрица 280 на 280. Примерный вид запроса:

```
{
 “matrix” : [[0, 255, 255 … 0], [255, 0, 0 ... 0], …, [0, 0, 0 …, 0]]
}
```

После происходит ожидание ответа от сервера. Сервер также используется json в качестве ответа. В ответе с ключом “predicted_class” содержится цифра, которую предсказала модель и с ключом “probabilities” массив из 10 элементов, которые представляют собой вероятность каждой цифры, которая выдала модель. Примерный вид ответа с сервера:

```
{
 “predicted_class” : “1”
 “probabilities” : [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
}
```

Затем полученные данные достаются с JSON. Предсказанные моделью вероятности записываются в таблица 10 на 2, где первая строка – это цифра, а вторая это вероятность, с которой эта же цифра нарисована на холсте. Получившийся интерфейс выглядит:

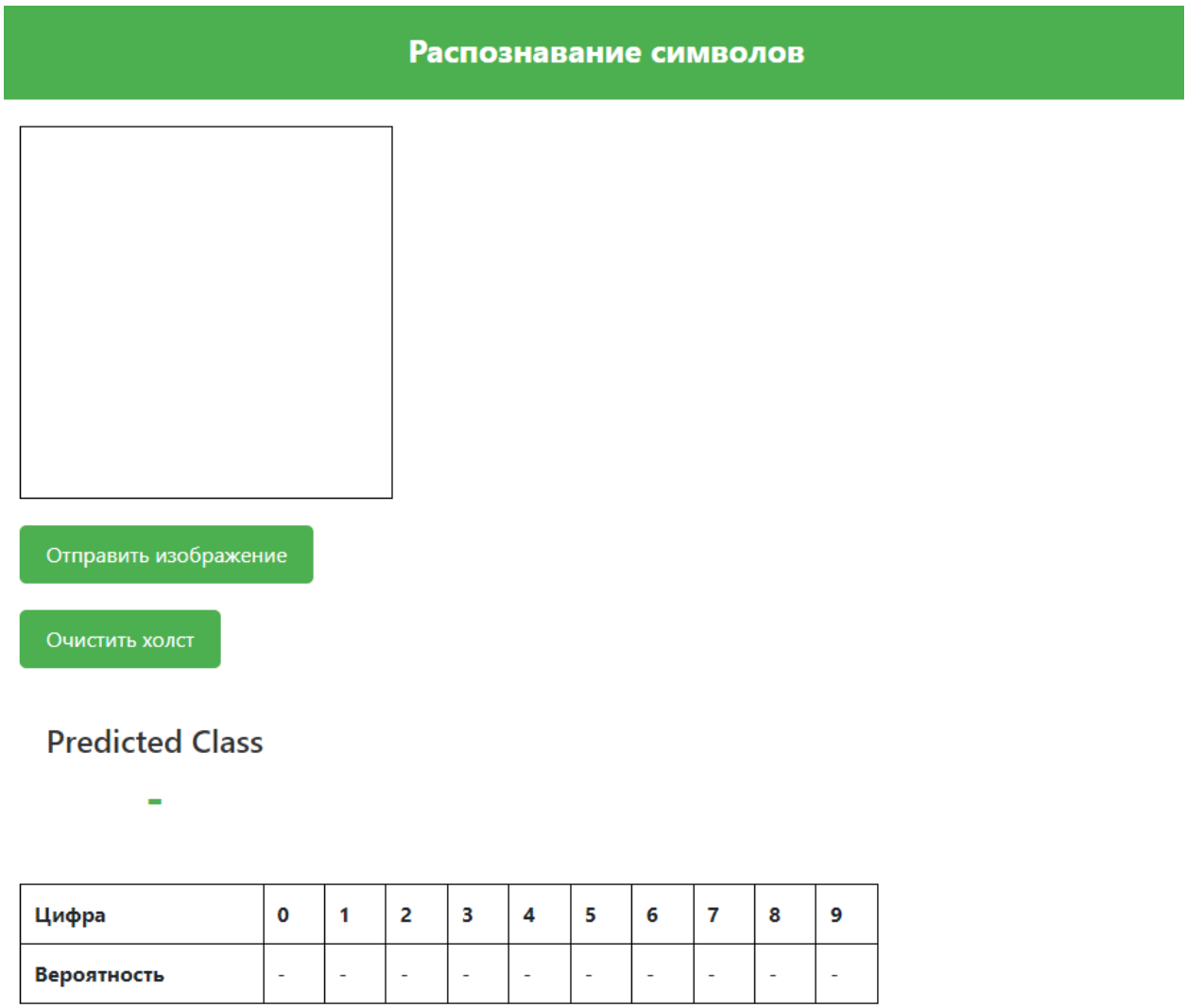
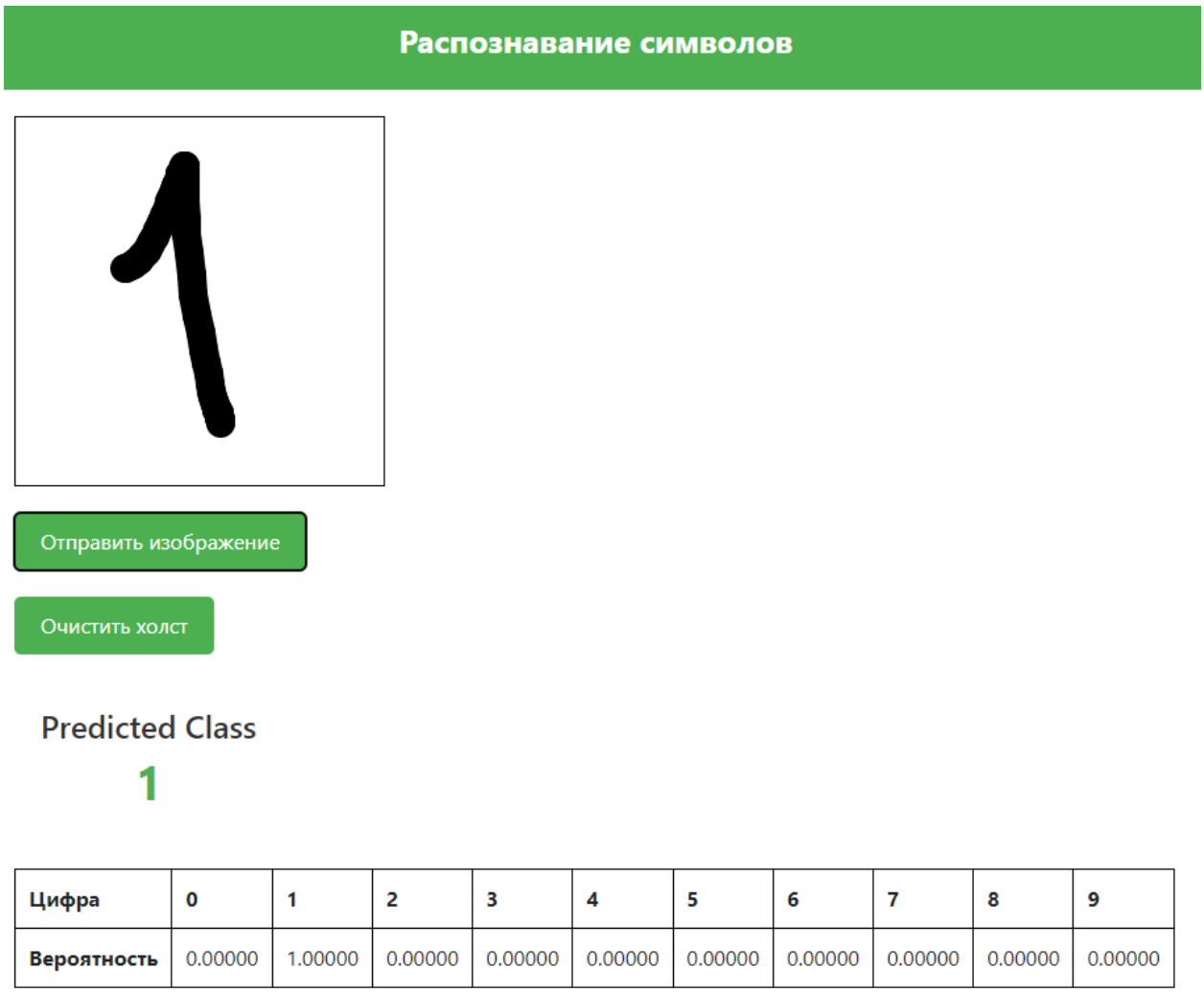


Рисунок 7. Реализованный веб-интерфейс.

После получения с сервера ответа на рисунок интерфейс будет выглядеть как на рисунке 6:



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

Рисунок 8. Реализованный интерфейс с ответом с сервера.

Раздел 4. Создание сервера для обработки изображений.

Серверная часть приложения построена с использованием FastAPI, фреймворка для создания веб-приложений на Python. Он принимает POST-запросы по маршруту /upload-image. Каждый запрос имеет одинаковый вид, описанный *в разделе 3*. В запросе JSON изображение представлено в виде матрицы 280 на 280. Работу бэкенда можно разделить на 3 этапа: подготовка, предсказание и отправка.

Первым этапом является подготовка исходного изображения. Так как MNIST набор представляет собой изображение 28 на 28, а полученное изображение имеет размер 280 на 280, находится среднее значение каждого 10 на 10 элементов матрицы и записывается в новую матрицу. Таким образом получается изображение, подходящее для работы модели.

Вторым этапом является предсказание моделью нейронной сети изображения. Получаем массив из 10 элементов, где под номером i находится вероятность, с которой изображено i на холсте. Из этого массива выбирается максимально вероятный индекс и сохраняется.

Последним третьим этапом является отправка полученных данных. Полученные данные переводятся в JSON формат, где массив вероятностей записывается с ключом “probabilities”, а наивероятное число записывается с ключом “predicted_class”. Этот JSON отправляется ответом на фронтенд ответом на изначальный запрос.

Раздел 5. Подведение итогов и вывод.

5.1. Оценка результатов.

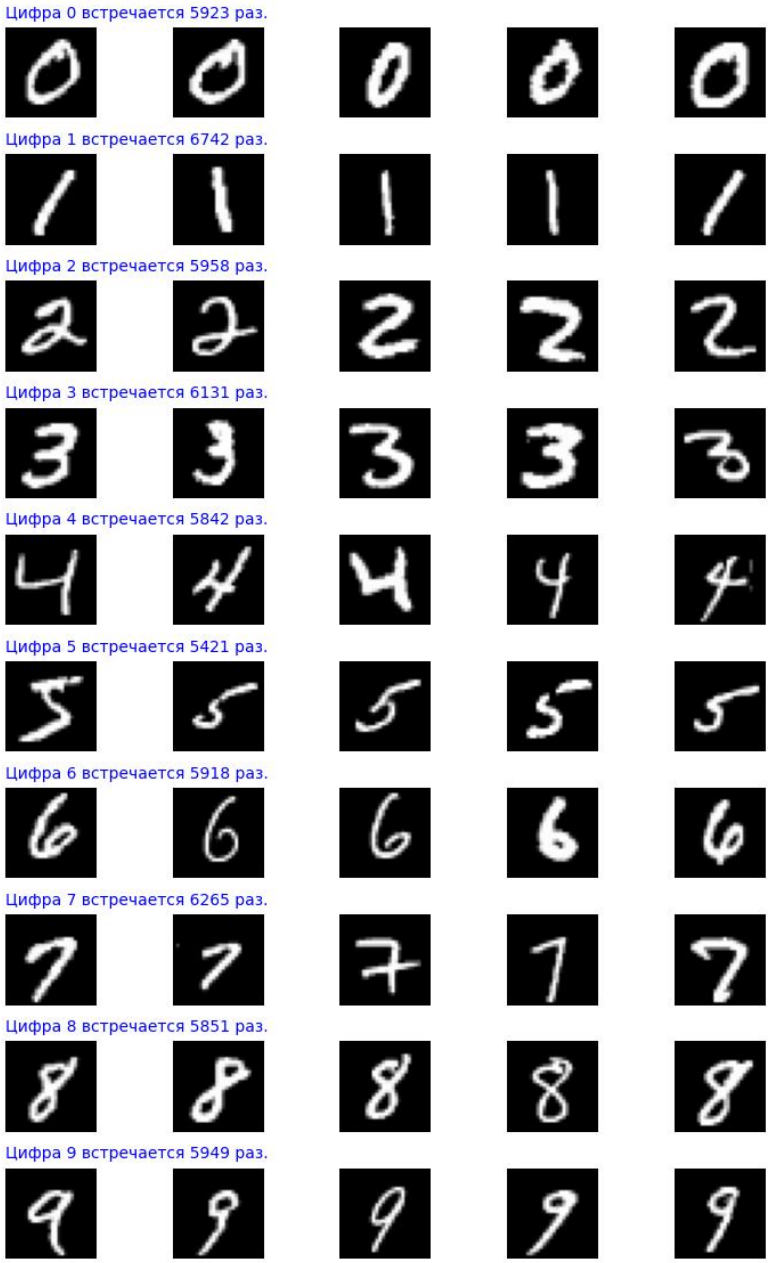
Поскольку предложенная в работе модель достигает высокой степени точности (более 98%) на обучающей выборке, а реализованное веб-серверное приложение предусматривает возможности для распознавания рукописных цифр, можно сделать вывод об эффективности разработанного приложения. Однако стоит отметить, что 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. Все коды можно посмотреть по ссылке https://github.com/panikkuo/character-recognition/

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.

Приложение.

Приложение 1.



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
```

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()
```

Список использованной литературы.

[1] Ян Гудфеллоу, Иошуа Бенджио, Аарон Курвилль”Глубокое обучение”. – пер. с англ. А.А.Слинкина – 2-е издание М.: ДМК Пресс, 2018.

[2] Нейронные сети: общие технологические характеристики – Журнал научное обозрение 22.03.2019 [Электронный ресурс]. – Режим доступа: ссылка https://science-engineering.ru/ru/article/view?id=1236