|  |  |
| --- | --- |
| Picture 1 | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  «Санкт-Петербургский государственный морской технический университет» (СПбГМТУ) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет цифровых промышленных технологий

Направление подготовки 09.03.01

"**Интеллектуальные технологии киберфизических систем**"

Лабораторная работа №6

Студент 2 курса группы 20221

Очного отделения

Руденко Вячеслав Сергеевич

Проверил:

Преподаватель CПбГМТУ

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

ОГЛАВЛЕНИЕ

[ОБЩАЯ ЧАСТЬ 3](#_Toc185244746)

1. [1.1 Цель работы 3](#_Toc185244747)

[ХОД РАБОТЫ 4](#_Toc185244748)

1. [Описание структуры программы 4](#_Toc185244749)
2. [Проведение экспериментов 11](#_Toc185244750)

[ЗАКЛЮЧЕНИЕ 14](#_Toc185244751)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 15](#_Toc185244752)

[ПРИЛОЖЕНИЕ А 16](#_Toc185244753)

# ОБЩАЯ ЧАСТЬ

## 1.1 Цель работы

Изучить архитектуру нейронной сети для классификации рукописных цифр, основываясь на датасете MNIST. Реализовать простую полносвязную нейронную сеть, провести эксперименты с гиперпараметрами, применить методы расширения данных (data augmentation) и добиться точности более 95,5% на тестовом наборе данных.

1.2 Формулировка задачи

* Реализовать нейронную сеть для классификации рукописных цифр из датасета MNIST:
* Создать функцию для сдвига изображения на один пиксель в четырёх направлениях (влево, вправо, вверх, вниз). Использовать эту функцию для генерации увеличенного обучающего набора данных, добавив четыре сдвинутые копии каждого изображения.
* Провести три эксперимента:
* Обучить исходную модель (без изменений).
* Обучить улучшенную модель с оптимальными гиперпараметрами.
* Обучить улучшенную модель на расширенном наборе данных.
* Сравнить результаты трёх моделей:
* Привести текстовые результаты (точность и гиперпараметры).
* Построить графики ошибок (обучения и тестирования) для каждой модели.

# ХОД РАБОТЫ

## Описание структуры программы

В начале программы импортируются следующие библиотеки: numpy — используется для работы с массивами и матричными вычислениями. matplotlib.pyplot — применяется для построения графиков. sklearn.datasets.fetch\_openml — загружает датасет MNIST. Импорт показан на рисунке 1.

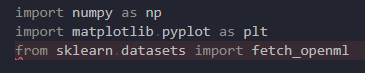


Рисунок 1 – Загрузка данных

Для воспроизводимости результатов устанавливается фиксированный исходный параметр для генерации случайных чисел. Это изображено на рисунке 2



Рисунок 2 – Установка фиксированного параметра

В функции read\_mnist реализована загрузка и первичное разбиение данных на обучающую и тестовую выборки. Используется следующая функция:



Рисунок 3 – Реализация функции read\_mnist

В функции read\_mnist загружаются данные MNIST. mnist["data"] содержит массив изображений, а mnist["target"] содержит метки классов, которые конвертируются в целые числа. Далее данные разделяются на тестовые и на данные для обучения. Тестовые данные содержат 10тыс. изображений, а данные для обучения содержат 60тыс. изображений



Рисунок 4 – Преобразование данных в формат float32:

Данные изображений изначально представлены в виде целых чисел (от 0 до 255). Это изображено на рисунке 4

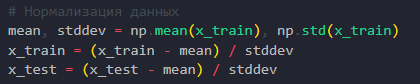


Рисунок 5 – Нормализация данных

Среднее значение (mean) и стандартное отклонение (stddev) вычисляются на основе обучающего набора x\_train.

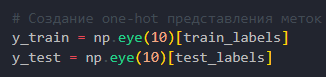


Рисунок 6 – Преобразование меток в формат One-Hot Encoding:

На рисунке 6 изображено, как np.eye(10) создаёт единичную матрицу размером 10×10. Каждая строка этой матрицы представляет метку в формате One-Hot Encoding.

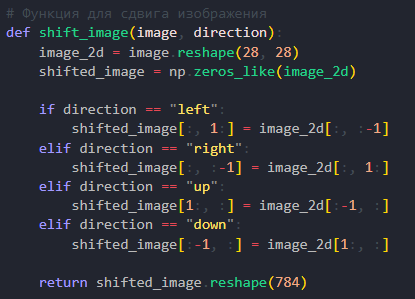


Рисунок 6 – Функция для сдвига изображений

На рисунке 6 показана функция, предназначенная для сдвига изображений. Поскольку изображение изначально представлено в виде одномерного массива длиной 784, его необходимо преобразовать в двумерный массив размером 28×28. Пустой массив shifted\_image используется для хранения сдвинутого изображения. В зависимости от направления сдвига выполняется копирование соответствующих пикселей:

* Влево: Каждая строка копируется со сдвигом вправо, теряя крайний левый пиксель.
* Вправо: Каждая строка копируется со сдвигом влево, теряя крайний правый пиксель.
* Вверх: Каждая строка сдвигается вверх, теряя верхний край.
* Вниз: Каждая строка сдвигается вниз, теряя нижний край.

После выполнения сдвига изображение возвращается в формате одномерного массива для последующего использования в обучающем наборе данных.

Для увеличения объёма обучающего набора реализована функция create\_augmented\_data, которая создаёт 4 дополнительные копии каждого изображения со сдвигами в четырёх направлениях (влево, вправо, вверх и вниз). Её реализация изображена на рисунке 7

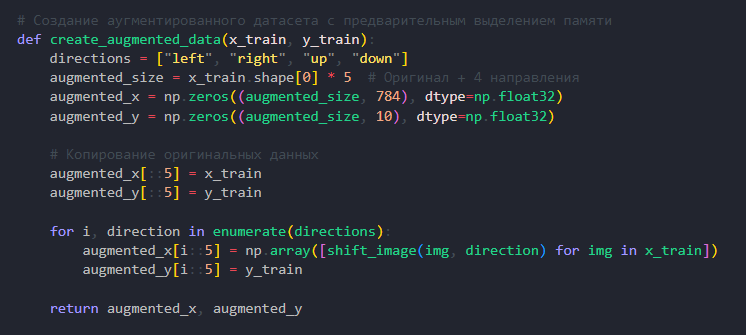


Рисунок 7 – Реализация функции augmented

Размер нового набора данных увеличивается в 5 раз, так как добавляются 4 сдвинутых копии для каждого изображения.

Класс Model представляет собой реализацию нейронной сети с одним скрытым слоем и настраиваемыми гиперпараметрами.

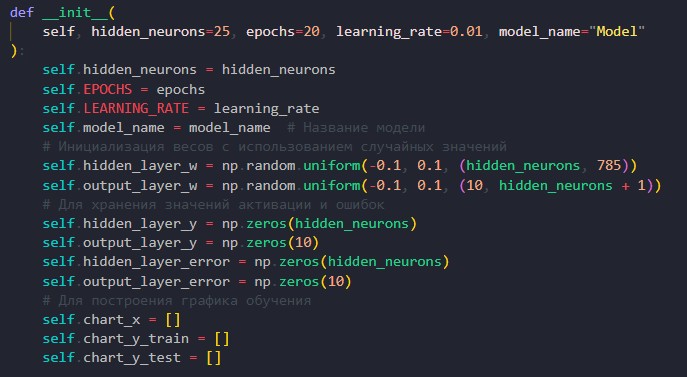


Рисунок 8 – Реализация инициализации класса Model

При создании экземпляра класса задаются следующие параметры:

* hidden\_neurons — количество нейронов в скрытом слое.
* epochs — количество эпох обучения.
* learning\_rate — скорость обучения.
* model\_name — название модели, которое используется для вывода и графиков.

Далее происходит инициализация весов: self.hidden\_layer\_w — веса между входным слоем (784 + 1 нейрон для bias) и скрытым слоем. И self.output\_layer\_w — веса между скрытым слоем (с bias) и выходным слоем (10 нейронов, по одному на каждый класс). Также создаются массивы для хранения значений активации и ошибок скрытого и выходного слоёв, а также переменные для построения графиков обучения.

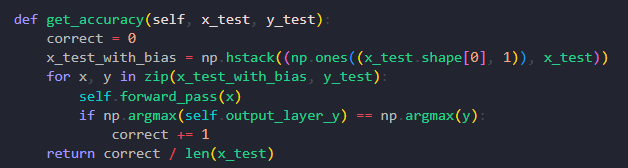


Рисунок 9 – Реализация метода get\_accurancy

Метод, изображенный на рисунке 9, вычисляет точность на тестовом наборе данных. Для этого выполняется прямой проход (forward pass) для каждого примера тестового набора, и предсказанный класс (индекс максимального значения на выходе) сравнивается с истинной меткой. Возвращается доля правильно классифицированных примеров.

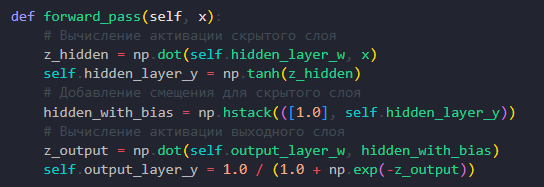


Рисунок 10 – Реализация метода forward\_pass

Метод, изображенный на рисунке 10, реализует прямое распространение данных через сеть. В нём вычисляются активации скрытого слоя с помощью функции активации tanh, добавляется bias для скрытого слоя, а так же вычисляются активации выходного слоя с использованием сигмоидной функции.

Этот метод преобразует входные данные в выходные значения, которые интерпретируются как вероятности принадлежности к различным классам.

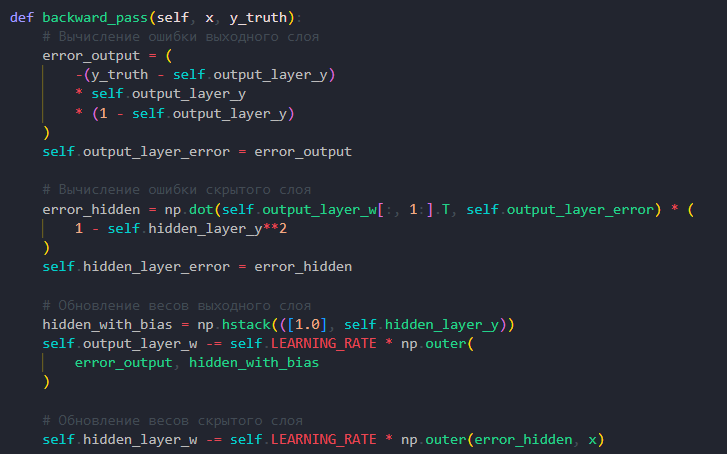


Рисунок 10 – Реализация метода backward\_pass

Метод, изображенный на рисунке 10, выполняет обратное распространение ошибки. В нём вычисляются ошибки выходного слоя на основе разницы между предсказанными и истинными значениями. Эти ошибки распространяются обратно через сеть для вычисления ошибок скрытого слоя. На основе ошибок и градиентов обновляются веса как для скрытого слоя, так и для выходного слоя с использованием градиентного спуска.

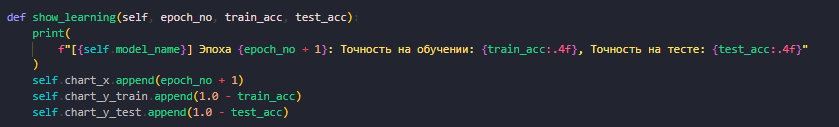


Рисунок 11 – Реализация метода show\_learning

Метод show\_learning, изображенный на рисунке 11, выводит результаты обучения на текущей эпохе, включая точность на обучающем и тестовом наборах данных. Также сохраняет ошибки для построения графиков.

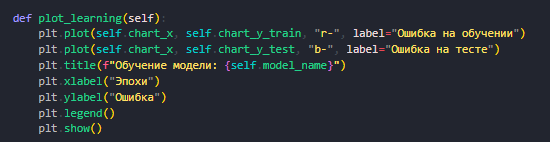


Рисунок 12 – Реализация метода plot\_learning

Метод plot\_learning, изображенный на рисунке 12, строит графики ошибок на обучении и тестировании по всем эпохам. Заголовок графика отображает название модели

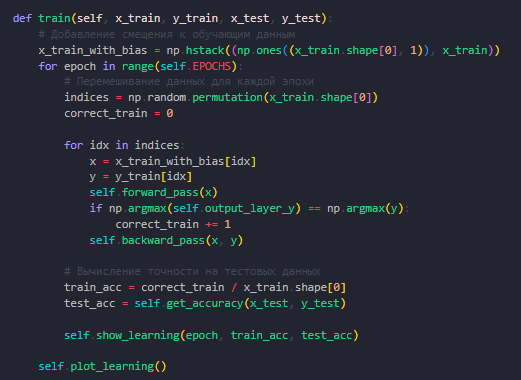


Рисунок 13 – Реализация метода plot\_learning

На рисунке 13 изображен основной метод обучения модели. Он добавляет bias к обучающим данным.

Цикл происходит по эпохам, в каждой из которых данные перемешиваются для случайной подачи. Затем для каждого примера выполняются:

1. Прямой проход через сеть (forward\_pass).
2. Обратное распространение ошибки (backward\_pass).
3. Подсчёт точности на обучающем и тестовом наборах.
4. Вывод результатов через show\_learning.

После завершения обучения строится график ошибок с помощью метода plot\_learning.

Обучение моделей выполняется с использованием класса Model, который реализует нейронную сеть. В работе используется три различных модели, каждая из которых обучается на собственных данных и с разными настройками гиперпараметров. Это изображено на рисунке 14.

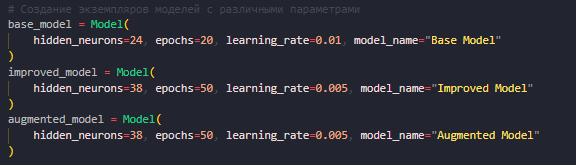


Рисунок 14 – Создание разных видов моделей с разными параметрами

Для обучения моделей и вывода конечной точности реализована функция train\_model, которая изображена на рисунке 15.

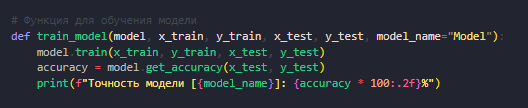


Рисунок 14 – Реализация функции train\_model

На рисунке 15 изображено то, как используется функция train\_model для моделей.

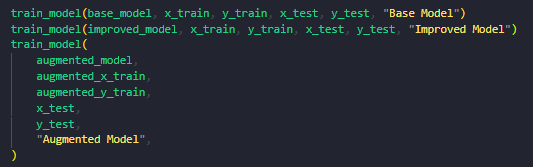


Рисунок 15– Использование функции train\_model

## Проведение экспериментов

Для анализа были обучены три модели с различными параметрами и наборами данных. Их основные отличия заключаются в архитектуре, гиперпараметрах и размере обучающего набора.

Base Model (базовая модель):

* Использует 24 нейрона в скрытом слое.
* Обучалась на оригинальном наборе данных (60,000 изображений).
* Количество эпох: 20.
* Скорость обучения: 0.01.

Эта модель представляет собой минимальную реализацию нейронной сети без дополнительных улучшений.

Improved Model (улучшенная модель):

* Использует 38 нейронов в скрытом слое (увеличено по сравнению с базовой моделью).
* Обучалась также на оригинальном наборе данных.
* Количество эпох: 50 (в 2,5 раза больше, чем в базовой модели).
* Скорость обучения снижена до 0.005 для более стабильного обучения.

Эта модель была создана для изучения влияния гиперпараметров на качество классификации.

Augmented Model (модель с расширенным набором данных):

* Имеет то же количество нейронов в скрытыъ слоях и гиперпараметры, что и Improved Model (38 нейронов, 50 эпох, скорость обучения 0.005).
* Основное отличие — обучение на расширенном наборе данных, который в 5 раз больше оригинального (включает сдвинутые копии изображений).

Эта модель позволяет оценить, как увеличение данных с помощью аугментации влияет на качество классификации.

Для анализа были обучены три модели, и результаты их обучения представлены на графиках, изображенных на рисунке 16, и в виде итоговых точностей, изображенных на рисунке 17

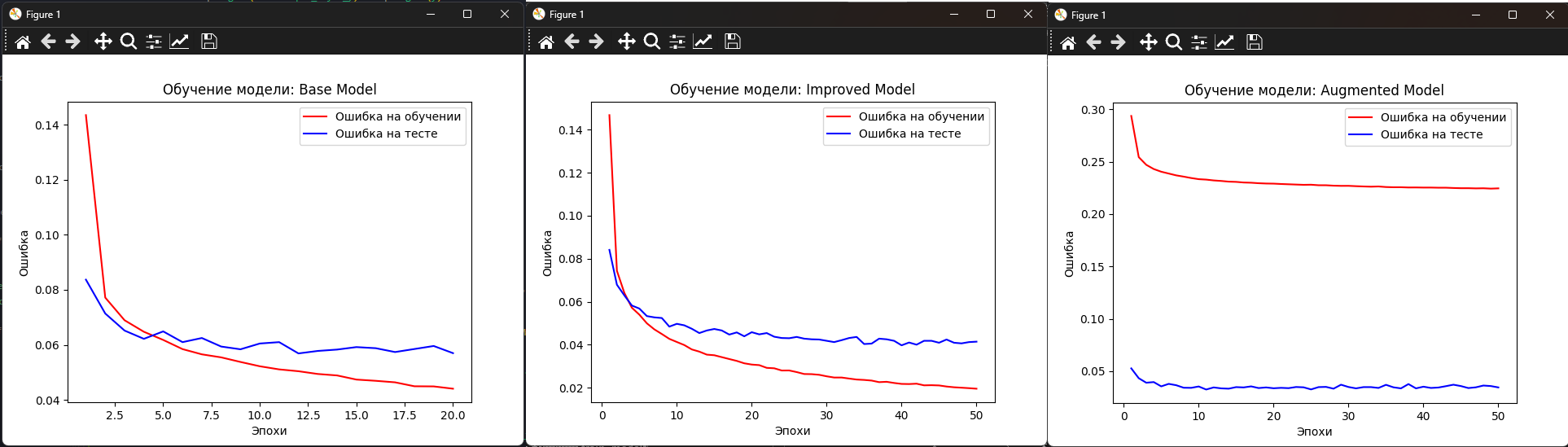


Рисунок 16– Результаты обучения моделей на графике



Рисунок 17– Результаты обучения моделей в консоли

Точность базовой модели: 94.3%. Наблюдая за обучением этой модели, можно сказать, что ошибки в процессе обучения постепенно снижаются, но модели не хватает количества эпох и количества скрытых нейронов.

Точность же улучшенной модели 95.86%, а значит, что порог в 95.5% пройден. Увеличение количество скрытых слоев с 24 до 38, увеличение количества эпох с 20 до 50 и уменьшение шага обучения повысили показатели точности на 1.56%. График показывает, что ошибки на тесте стабилизируются.

Точность модели с расширенным набором данных (Augmented Model) – 96.55%. Прирост по сравнению с улучшенной моделью – 0,86%. Это означает, что использование расширенного набора данных значительно улучшило результат. Судя по графику, ошибки на тестовом наборе данных стабилизировались на низком уровне, что говорит о устойчивости модели к небольшим сдвигам во входных данных

Выводы по экспериментам:

* Базовая модель показала недостаточную точность (94.30%) из-за ограниченного количества скрытых нейронов и эпох.
* Улучшенная модель превысила целевой показатель 95.5%, благодаря увеличению числа нейронов и более длительному обучению.
* Модель с расширенными данными показала наилучший результат (96.55%), что подтверждает эффективность аугментации данных для улучшения обобщающей способности нейронной сети.

Таким образом, цель работы была достигнута: точность модели на тестовом наборе данных превысила 95.5%.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была реализована нейронная сеть с одним скрытым слоем для задачи классификации рукописных цифр из датасета MNIST. В процессе работы были обучены три модели:

* Base Model — базовая версия сети с минимальными настройками гиперпараметров.
* Improved Model — улучшенная версия с увеличенным числом нейронов и количеством эпох.
* Augmented Model — улучшенная модель, обученная на расширенном наборе данных.

Результаты эксперимента показали, что увеличение количества: эпох, скрытых слоев и данных, а так же уменьшения learning\_rate ведет к увеличению точности модели.

Таким образом, цель работы была полностью достигнута. Реализованная модель с расширенным количеством данных продемонстрировала высокую точность классификации рукописных цифр, а применение методов расширения данных и оптимизация гиперпараметров позволили значительно улучшить результаты.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.0.97-2016. Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Организационно-распорядительная документация. Требования к оформлению документов: утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 14.05.2018 N 244-ст: Дата введения 2018-07-01. - URL: <https://docs.cntd.ru/document/1200159234> (дата обращения: 02.10.2024). - Текст: электронный.

# ПРИЛОЖЕНИЕ А

Листинг кода

Листинг 1 - index.py

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

import multiprocessing

# Установка начального зерна для воспроизводимости

np.random.seed(7)

# Загрузка датасета MNISTп

def read\_mnist():

    mnist = fetch\_openml("mnist\_784", version=1, as\_frame=False)

    images, labels = mnist["data"], mnist["target"].astype(int)

    # Разделение на обучающую и тестовую выборки

    train\_images, train\_labels = images[:60000], labels[:60000]

    test\_images, test\_labels = images[60000:], labels[60000:]

    return train\_images, train\_labels, test\_images, test\_labels

# Подготовка данных

train\_images, train\_labels, test\_images, test\_labels = read\_mnist()

x\_train = train\_images.astype(np.float32)

x\_test = test\_images.astype(np.float32)

mean, stddev = np.mean(x\_train), np.std(x\_train)

x\_train = (x\_train - mean) / stddev

x\_test = (x\_test - mean) / stddev

# Создание one-hot представления меток

y\_train = np.eye(10)[train\_labels]

y\_test = np.eye(10)[test\_labels]

# Функция для сдвига изображения

def shift\_image(image, direction):

    image\_2d = image.reshape(28, 28)

    shifted\_image = np.zeros\_like(image\_2d)

    if direction == "left":

        shifted\_image[:, 1:] = image\_2d[:, :-1]

    elif direction == "right":

        shifted\_image[:, :-1] = image\_2d[:, 1:]

    elif direction == "up":

        shifted\_image[1:, :] = image\_2d[:-1, :]

    elif direction == "down":

        shifted\_image[:-1, :] = image\_2d[1:, :]

    return shifted\_image.reshape(784)

# Создание аугментированного датасета с предварительным выделением памяти

def create\_augmented\_data(x\_train, y\_train):

    directions = ["left", "right", "up", "down"]

    augmented\_size = x\_train.shape[0] \* 5  # Оригинал + 4 направления

    augmented\_x = np.zeros((augmented\_size, 784), dtype=np.float32)

    augmented\_y = np.zeros((augmented\_size, 10), dtype=np.float32)

    # Копирование оригинальных данных

    augmented\_x[::5] = x\_train

    augmented\_y[::5] = y\_train

    for i, direction in enumerate(directions):

        augmented\_x[i::5] = np.array([shift\_image(img, direction) for img in x\_train])

        augmented\_y[i::5] = y\_train

    return augmented\_x, augmented\_y

augmented\_x\_train, augmented\_y\_train = create\_augmented\_data(x\_train, y\_train)

# Определение класса модели

class Model:

    def \_\_init\_\_(

        self, hidden\_neurons=25, epochs=20, learning\_rate=0.01, model\_name="Model"

    ):

        self.hidden\_neurons = hidden\_neurons

        self.EPOCHS = epochs

        self.LEARNING\_RATE = learning\_rate

        self.model\_name = model\_name  # Название модели

        # Инициализация весов с использованием случайных значений

        self.hidden\_layer\_w = np.random.uniform(-0.1, 0.1, (hidden\_neurons, 785))

        self.output\_layer\_w = np.random.uniform(-0.1, 0.1, (10, hidden\_neurons + 1))

        # Для хранения значений активации и ошибок

        self.hidden\_layer\_y = np.zeros(hidden\_neurons)

        self.output\_layer\_y = np.zeros(10)

        self.hidden\_layer\_error = np.zeros(hidden\_neurons)

        self.output\_layer\_error = np.zeros(10)

        # Для построения графика обучения

        self.chart\_x = []

        self.chart\_y\_train = []

        self.chart\_y\_test = []

    def get\_accuracy(self, x\_test, y\_test):

        correct = 0

        x\_test\_with\_bias = np.hstack((np.ones((x\_test.shape[0], 1)), x\_test))

        for x, y in zip(x\_test\_with\_bias, y\_test):

            self.forward\_pass(x)

            if np.argmax(self.output\_layer\_y) == np.argmax(y):

                correct += 1

        return correct / len(x\_test)

    def forward\_pass(self, x):

        # Вычисление активации скрытого слоя

        z\_hidden = np.dot(self.hidden\_layer\_w, x)

        self.hidden\_layer\_y = np.tanh(z\_hidden)

        # Добавление смещения для скрытого слоя

        hidden\_with\_bias = np.hstack(([1.0], self.hidden\_layer\_y))

        # Вычисление активации выходного слоя

        z\_output = np.dot(self.output\_layer\_w, hidden\_with\_bias)

        self.output\_layer\_y = 1.0 / (1.0 + np.exp(-z\_output))

    def backward\_pass(self, x, y\_truth):

        # Вычисление ошибки выходного слоя

        error\_output = (

            -(y\_truth - self.output\_layer\_y)

            \* self.output\_layer\_y

            \* (1 - self.output\_layer\_y)

        )

        self.output\_layer\_error = error\_output

        # Вычисление ошибки скрытого слоя

        error\_hidden = np.dot(self.output\_layer\_w[:, 1:].T, self.output\_layer\_error) \* (

            1 - self.hidden\_layer\_y\*\*2

        )

        self.hidden\_layer\_error = error\_hidden

        # Обновление весов выходного слоя

        hidden\_with\_bias = np.hstack(([1.0], self.hidden\_layer\_y))

        self.output\_layer\_w -= self.LEARNING\_RATE \* np.outer(

            error\_output, hidden\_with\_bias

        )

        # Обновление весов скрытого слоя

        self.hidden\_layer\_w -= self.LEARNING\_RATE \* np.outer(error\_hidden, x)

    def show\_learning(self, epoch\_no, train\_acc, test\_acc):

        print(

            f"[{self.model\_name}] Эпоха {epoch\_no + 1}: Точность на обучении: {train\_acc:.4f}, Точность на тесте: {test\_acc:.4f}"

        )

        self.chart\_x.append(epoch\_no + 1)

        self.chart\_y\_train.append(1.0 - train\_acc)

        self.chart\_y\_test.append(1.0 - test\_acc)

    def plot\_learning(self):

        plt.plot(self.chart\_x, self.chart\_y\_train, "r-", label="Ошибка на обучении")

        plt.plot(self.chart\_x, self.chart\_y\_test, "b-", label="Ошибка на тесте")

        plt.title(f"Обучение модели: {self.model\_name}")

        plt.xlabel("Эпохи")

        plt.ylabel("Ошибка")

        plt.legend()

        plt.show()

    def train(self, x\_train, y\_train, x\_test, y\_test):

        # Добавление смещения к обучающим данным

        x\_train\_with\_bias = np.hstack((np.ones((x\_train.shape[0], 1)), x\_train))

        for epoch in range(self.EPOCHS):

            # Перемешивание данных для каждой эпохи

            indices = np.random.permutation(x\_train.shape[0])

            correct\_train = 0

            for idx in indices:

                x = x\_train\_with\_bias[idx]

                y = y\_train[idx]

                self.forward\_pass(x)

                if np.argmax(self.output\_layer\_y) == np.argmax(y):

                    correct\_train += 1

                self.backward\_pass(x, y)

            # Вычисление точности на тестовых данных

            train\_acc = correct\_train / x\_train.shape[0]

            test\_acc = self.get\_accuracy(x\_test, y\_test)

            self.show\_learning(epoch, train\_acc, test\_acc)

        self.plot\_learning()

# Функция для обучения модели

def train\_model(model, x\_train, y\_train, x\_test, y\_test, model\_name="Model"):

    model.train(x\_train, y\_train, x\_test, y\_test)

    accuracy = model.get\_accuracy(x\_test, y\_test)

    print(f"Точность модели [{model\_name}]: {accuracy \* 100:.2f}%")

if \_\_name\_\_ == "\_\_main\_\_":

    # Создание экземпляров моделей с различными параметрами

    base\_model = Model(

        hidden\_neurons=24, epochs=20, learning\_rate=0.01, model\_name="Base Model"

    )

    improved\_model = Model(

        hidden\_neurons=38, epochs=50, learning\_rate=0.005, model\_name="Improved Model"

    )

    augmented\_model = Model(

        hidden\_neurons=38, epochs=50, learning\_rate=0.005, model\_name="Augmented Model"

    )

    train\_model(base\_model, x\_train, y\_train, x\_test, y\_test, "Base Model")

    train\_model(improved\_model, x\_train, y\_train, x\_test, y\_test, "Improved Model")

    train\_model(

        augmented\_model,

        augmented\_x\_train,

        augmented\_y\_train,

        x\_test,

        y\_test,

        "Augmented Model",

    )