

ГУАП

КАФЕДРА №41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Ассистент

должность, уч. степень, звание

подпись, дата

А.С. Раскопина

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Использование нейронных сетей прямого распространения для
решения задач классификации

по курсу: МАШИННОЕ ОБУЧЕНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

4217

подпись, дата

А.Н. Медянкина

инициалы, фамилия

Санкт-Петербург 2025

1. Цель работы

Изучение основ работы с нейронными сетями прямого распространения (FNN) для классификации данных, обучение модели на подготовленном датасете, анализ и оценка полученных результатов.

2. Задачи

1. Ознакомиться с принципом работы сети прямого распространения (FNN) и её применением в задачах классификации.
2. Подготовить датасет для обучения модели.
3. Реализовать и обучить нейронную сеть прямого распространения (FNN) с использованием выбранного инструмента (PyTorch, TensorFlow или Keras).
4. Провести обучение сети на подготовленных данных.
5. Оценить точность работы модели и проанализировать полученные результаты.
6. Составить отчет, в котором будет описан процесс работы и выводы.

3. Индивидуальный вариант

Вариант 2: «Классификация рукописных цифр (MNIST)»

Шаги работы:

1. Загрузка данных:
 - Датасет MNIST доступен через Keras и на Kaggle.
 - Скачать и загрузить датасет.
2. Предобработка данных:
 - Нормализовать изображения.
 - Разделить данные на обучающую и тестовую выборки.
 - Преобразовать метки классов в формат one-hot encoding ()

3. Обучение модели:

- Создать нейронную сеть для классификации рукописных цифр.

4. Тестирование:

- Протестировать модель на тестовых данных.
- Оценить точность модели на тестовой выборке.

4. Ход работы

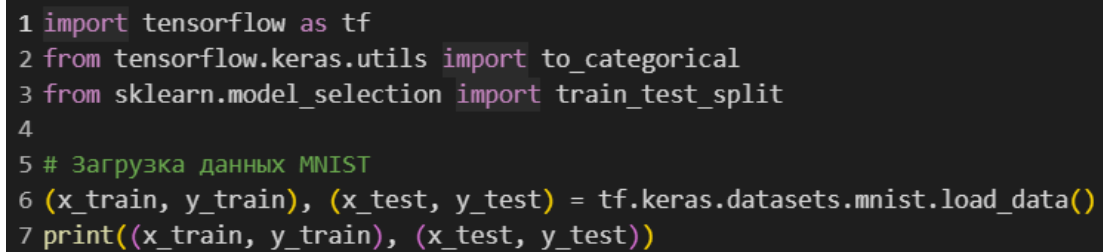
Для начала был загружен датасет MNIST на сайте Kaggle: <https://www.kaggle.com/datasets/hojjatk/mnist-dataset?resource=download> и импортирован в Google Collab.



```
1 import kagglehub
2
3 # Download latest version
4 path = kagglehub.dataset_download("hojjatk/mnist-dataset")
5
6 print("Path to dataset files:", path)
```

Рисунок 1 – Импорт скачанного датасета в Google Collab

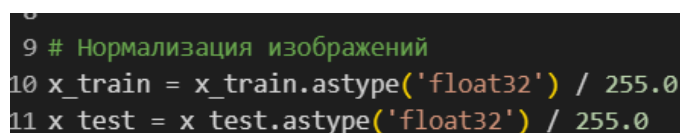
После чего были произведены импорт необходимых для машинного обучения библиотек и загрузка данных MNIST для обучения.



```
1 import tensorflow as tf
2 from tensorflow.keras.utils import to_categorical
3 from sklearn.model_selection import train_test_split
4
5 # Загрузка данных MNIST
6 (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
7 print((x_train, y_train), (x_test, y_test))
```

Рисунок 2 – Импорт библиотек и загрузка MNIST

После чего, с помощью применения `astype('float32')` – преобразует массив в тип данных `float32`, чтобы можно было работать с дробными числами – и деления на `255.0` – делит каждое значение пикселя на `255`, масштабируя его в диапазон `[0, 1]`, производится нормализация изображения.



```
9 # Нормализация изображений
10 x_train = x_train.astype('float32') / 255.0
11 x_test = x_test.astype('float32') / 255.0
```

Рисунок 3 – Нормализация изображений

Затем с помощью встроенной функции «`to_categorical`» были преобразованы метки классов в формат `one-hot encoding` () – процесс преобразования категориальных переменных в форму, подходящую

алгоритмам машинного обучения, который заключается в создании новых бинарных столбцов для каждой категории в исходных данных. При этом 1 означает, что категория присутствует, а 0 — нет.

```
13 # Преобразование меток классов в формат one-hot encoding (категориальный тип - создает бинарные столбцы для каждой категории.)
14 y_train_one_hot = to_categorical(y_train)
15 y_test_one_hot = to_categorical(y_test)
16
```

Рисунок 4 – Преобразование меток классов в формат one-hot encoding ()

Уже после преобразования данные были разделены на обучающую и тестовую выборки с помощью функции `train_test_split()`, которая разделяет массивы или матрицы на случайные подвыборки для обучающих и тестовых данных соответственно.

```
17 # Разделение данных на обучающую и тестовую выборки
18 # В данном случае, данные уже разделены, но можно дополнительно разделить обучающую выборку
19 x_train_split, x_val_split, y_train_split, y_val_split = train_test_split(
20     x_train, y_train_one_hot, test_size=0.2, random_state=42
21 )
22 print()
23 # Проверка формы данных
24 print(x_train_split.shape, x_val_split.shape, y_train_split.shape, y_val_split.shape, x_test.shape, y_test_one_hot.shape)
```

Рисунок 5 – Разделение на обучающую и тестовую выборки

Далее были добавлены еще библиотеки уже для работы с нейронной сетью для классификации рукописных цифр. Для этого была использована такая модель Keras как `Sequential`.

```
1 import tensorflow as tf
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense, Flatten
4
5
6 # Создание модели с категориальными данными и использованием такой модели Keras как Sequential
7 #тензор - многомерные массивы чисел
8 model = Sequential([
9     Flatten(input_shape=(28, 28)), # Преобразование 2D изображения в 1D вектор
10    Dense(128, activation='relu'), # Полносвязный слой с 128 нейронами и ReLU активацией
11    Dense(64, activation='relu'), # Полносвязный слой с 64 нейронами и ReLU активацией
12    Dense(10, activation='softmax') # Выходной слой с 10 нейронами (по числу классов) и softmax активацией
13 ])
```

Рисунок 6 – Создание модели с категориальными данными

Расскажу немного подробнее обо всех методах, которые были использованы в предыдущем коде, для понимания:

- `Flatten`: Преобразует каждое 28x28 изображение в одномерный вектор.

- Dense: Полносвязные слои, которые обрабатывают входные данные.
- Activation: Функции активации ReLU и softmax используются для введения нелинейности и получения вероятностного распределения на выходе.
- Compile: Указывает, что мы будем использовать оптимизатор Adam, функцию потерь categorical_crossentropy и метрику accuracy.
- Fit: Обучает модель на обучающих данных.
- Evaluate: Оценивает модель на тестовых данных.

После чего модель была скомпилирована и обучена.

```
15 # Компиляция модели
16 model.compile(optimizer='adam',
17               loss='categorical_crossentropy',
18               metrics=['accuracy'])
19
20 # Обучение модели
21 model2 = model.fit(x_train_split, y_train_split, epochs=5, batch_size=32, validation_data=(x_val_split, y_val_split))
22 print(model2)
```

Рисунок 7 – Компиляция и обучение модели

Результат компиляции и обучения модели представлен на рисунке 8.

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/resizing/flatten.py:37: UserWarning: Do not pass an `input_shape` in
super().__init__(**kwargs)
Epoch 1/5
1500/1500 ————— 11s 6ms/step - accuracy: 0.8642 - loss: 0.4692 - val_accuracy: 0.9571 - val_loss: 0.1402
Epoch 2/5
1500/1500 ————— 8s 4ms/step - accuracy: 0.9636 - loss: 0.1188 - val_accuracy: 0.9703 - val_loss: 0.1001
Epoch 3/5
1500/1500 ————— 11s 5ms/step - accuracy: 0.9778 - loss: 0.0725 - val_accuracy: 0.9661 - val_loss: 0.1056
Epoch 4/5
1500/1500 ————— 10s 5ms/step - accuracy: 0.9818 - loss: 0.0557 - val_accuracy: 0.9692 - val_loss: 0.1047
Epoch 5/5
1500/1500 ————— 8s 5ms/step - accuracy: 0.9874 - loss: 0.0386 - val_accuracy: 0.9732 - val_loss: 0.0922
<keras.src.callbacks.history.History object at 0x7cecd011a510>
```

Рисунок 8 – Результат обучения модели

Затем разом была протестирована модель на тестовых данных. После чего была оценена точность модели на тестовых выборках (с помощью разделения тестового набора на два подмножества).

```
1 test_loss, test_acc = model.evaluate(x_test, y_test_one_hot)
2 print(f'Точность модели на тестовых данных: {test_acc:.4f}')
```

313/313 ————— 1s 2ms/step - accuracy: 0.9661 - loss: 0.1081
Точность модели на тестовых данных: 0.9716

```
1 from sklearn.model_selection import train_test_split
2
3 # Разделение тестового набора на два подмножества
4 x_test_subset1, x_test_subset2, y_test_subset1, y_test_subset2 = train_test_split(
5     x_test, y_test_one_hot, test_size=0.5, random_state=42
6 )
7
8 # Оценка модели на первом подмножестве
9 test_loss1, test_acc1 = model.evaluate(x_test_subset1, y_test_subset1)
10 print(f'Точность модели на первом тестовом подмножестве: {test_acc1:.4f}')
```

157/157 ————— 0s 2ms/step - accuracy: 0.9698 - loss: 0.1032
Точность модели на первом тестовом подмножестве: 0.9722
0.08278897404670715 0.9721999764442444

```
11 print(test_loss1, test_acc1)
12
13 # Оценка модели на втором подмножестве
14 test_loss2, test_acc2 = model.evaluate(x_test_subset2, y_test_subset2)
15 print(f'Точность модели на втором тестовом подмножестве: {test_acc2:.4f}')
```

157/157 ————— 0s 3ms/step - accuracy: 0.9696 - loss: 0.0970
Точность модели на втором тестовом подмножестве: 0.9710

Рисунок 8 – Оценка качества работы модели на выборочных тестовых данных

На основе предоставленных результатов, можно сделать несколько выводов о производительности обученной модели:

1. Точность:

- Точность на первом тестовом подмножестве: 0.9722 (или 97.22%)
- Точность на втором тестовом подмножестве: 0.9710 (или 97.10%)

Обе точности очень близки к 97%, что свидетельствует о том, что модель хорошо обобщает данные и может правильно классифицировать изображения, которых она не видела во время обучения.

2. Потери:

- Потери на первом тестовом подмножестве: 0.1032
- Потери на втором тестовом подмножестве: 0.0970

Низкие значения потерь указывают на то, что модель хорошо справляется с предсказаниями, и разница между предсказанными и истинными значениями невелика.

3. Стабильность модели:

Так как точность и потери на двух разных тестовых подмножествах очень близки, можно сделать вывод, что модель стабильно работает на разных данных. Это хороший признак того, что модель не переобучилась на обучающих данных и способна обобщать.

5. Вывод

В ходе выполнения данной лабораторной работы были изучены основы работы с нейронными сетями прямого распространения (FNN) для классификации данных, обучена модель на подготовленном датасете, проведены анализ и оценка полученных результатов.

В процессе работы были выполнены следующие задачи:

1. Загрузка и подготовка данных: Датасет MNIST был успешно загружен и подготовлен для обучения модели. Это включало нормализацию изображений и преобразование меток классов в формат one-hot encoding.

2. Создание и обучение модели: Была создана нейронная сеть с использованием библиотеки Keras. Модель включала в себя слои Flatten и Dense с функциями активации ReLU и softmax. Модель была обучена на обучающих данных и протестирована на тестовых данных.

3. Оценка производительности: Модель показала высокую точность на тестовых данных, что свидетельствует о её способности правильно классифицировать изображения, которых она не видела во время обучения. Низкие значения потерь указывают на то, что модель хорошо справляется с предсказаниями.

3. Анализ стабильности: Модель продемонстрировала стабильную работу на разных тестовых подмножествах, что говорит о её способности обобщать данные и отсутствии переобучения.

Таким образом, выполненная работа показала эффективность использования нейронных сетей прямого распространения для задач классификации, а также подтвердила важность правильной подготовки данных и выбора архитектуры модели для достижения высокой точности и стабильности.

6. Список использованных источников.

1. Kaggle. MNIST dataset [Электронный ресурс] / Hojjat K. — 2023.
— Режим доступа: <https://www.kaggle.com/datasets/hojjatk/mnist-dataset?resource=download> (дата обращения: 18.02.2025).
2. Программирование на Python. Последовательная модель Keras [Электронный ресурс]. — Режим доступа: https://proproprogs.ru/neural_network/keras-posledovatel'naya-model-sequential (дата обращения: 18.02.2025).
3. Keras. Guide to Build a Sequential Model [Электронный ресурс]. — Режим доступа: https://keras.io/guides/sequential_model/ (дата обращения: 18.02.2025).

7. Приложение А

Ссылка на Google Collab:

<https://colab.research.google.com/drive/1sM81q9aHEVDHG5BN0CtBgQSs541ZWx1q?usp=sharing>