Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины «Основы нейронных сетей»

Выполнил:
Гайчук Дарья Дмитриевна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А.-доцент департамента цифровых, роботехнических систем и электроники института перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Исследование рекуррентных и одномерных сверточных нейронных сетей для классификации текстов

Цель: Изучить архитектуры SimpleRNN, LSTM, GRU и Conv1D для обработки текстовых последовательностей

Ссылка на git: https://github.com/Ichizuchi/NN_LR5

1. Подготовка данных

Произведены стандартные этапы предобработки:

— Импорт библиотек

— Загрузка и распаковка архива

— Формирование обучающих и тестовых выборок

— Токенизация и векторизация текстов

— Разделение на фрагменты (скользящее окно: WIN_SIZE, WIN_HOP)

Импорт всех необходимых библиотек для работы с текстом, нейросетями, визуализацией и метриками.

```
# Работа с массивами данных
import numpy as np

# Функции-утилиты для работы с категориальными данными
from tensorflow.keras import utils

# Класс для конструирования последовательной модели нейронной сети
from tensorflow.keras.models import Sequential

# Основные слои
from tensorflow.keras.layers import Dense, Dropout, SpatialDropout1D, BatchNormalization, Embedding, Flatten, Activation
from tensorflow.keras.layers import SimpleRNN, GRU, LSTM, Bidirectional, Conv1D, MaxPooling1D, GlobalMaxPooling1D

# Токенизатор для преобразование текстов в последовательности
from tensorflow.keras.preprocessing.text import Tokenizer

# Рисование схемы модели
from tensorflow.keras.utils import plot_model

# Матрица ошибок классификатора
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Загрузка датасетов из облака google
import gdown

# Функции операционной системы
import os

# Работа со временем
import time

# Регулярные выражения
import re

# Отрисовка графиков
import matplotlib.pyplot as plt

# Вывод объектов в ячейке colab
from IPython.display import display

%matplotlib inline
```

Рисунок 1. Импорт всех библиотек

Импорт всех необходимых библиотек для работы с текстом, нейросетями, визуализацией и метриками. Преобразование текстов в числовые последовательности с ограничением по частоте и длине.

```
# Токенизация и построение частотного словаря по обучающим текстам
with timex():
    # Используется встроенный в Keras токенизатор для разбиения текста и построения частотного словаря
    tokenizer = Tokenizer(num_words=VOCAB_SIZE, filters='!"#$%&()*+,-—./…:;<=>?@[\\]^_`{|}~«»\t\n\xa0\ufeff', lower=True,

    # Использованы параметры:
    # num_words   - объем словаря
    # filters     - убираемые из текста ненужные символы
    # lower       - приведение слов к нижнему регистру
    # split       - разделитель слов
    # char_level  - указание разделять по словам, а не по единичным символам
    # oov_token   - токен для слов, которые не вошли в словарь

    # Построение частотного словаря по обучающим текстам
    tokenizer.fit_on_texts(text_train)

    # Построение словаря в виде пар слово - индекс
    items = list(tokenizer.word_index.items())
```

```
Время обработки: 2.63 с
```

Рисунок 2. Токенизация

Преобразование текстов в числовые последовательности с ограничением по частоте и длине. Универсальная функция для компиляции и обучения модели с возвратом истории обучения.

```
# Функция компиляции и обучения модели нейронной сети
def compile_train_model(model,
                        x_train,
                        y_train,
                        x_val,
                        y_val,
                        optimizer='adam',
                        epochs=50,
                        batch_size=128,
                        figsize=(20, 5)):

    # Компиляция модели
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Вывод сводки
    model.summary()

    # Вывод схемы модели
    display(plot_model(model, dpi=60, show_shapes=True))

    # Обучение модели с заданными параметрами
    history = model.fit(x_train,
                        y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(x_val, y_val))

    # Вывод графиков точности и ошибки
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=figsize)
    fig.suptitle('График процесса обучения модели')
    ax1.plot(history.history['accuracy'],
             label='Доля верных ответов на обучающем наборе')
    ax1.plot(history.history['val_accuracy'],
             label='Доля верных ответов на проверочном наборе')
    ax1.xaxis.get_major_locator().set_params(integer=True)
    ax1.set_xlabel('Эпоха обучения')
    ax1.set_ylabel('Доля верных ответов')
    ax1.legend()
```

Рисунок 3. Функция для компиляции и обучения модели

Универсальная функция для компиляции и обучения модели с возвратом истории обучения. Функция визуализации точности обучения и валидации.

```
# Вывод графиков точности и ошибки
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=figsize)
fig.suptitle('График процесса обучения модели')
ax1.plot(history.history['accuracy'],
         label='Доля верных ответов на обучающем наборе')
ax1.plot(history.history['val_accuracy'],
         label='Доля верных ответов на проверочном наборе')
ax1.xaxis.get_major_locator().set_params(integer=True)
ax1.set_xlabel('Эпоха обучения')
ax1.set_ylabel('Доля верных ответов')
ax1.legend()

ax2.plot(history.history['loss'],
         label='Ошибка на обучающем наборе')
ax2.plot(history.history['val_loss'],
         label='Ошибка на проверочном наборе')
ax2.xaxis.get_major_locator().set_params(integer=True)
ax2.set_xlabel('Эпоха обучения')
ax2.set_ylabel('Ошибка')
ax2.legend()
plt.show()
```

Рисунок 4. Функция вывода графиков

Функция визуализации точности обучения и валидации. Отображение матрицы ошибок классификатора для оценки качества модели.

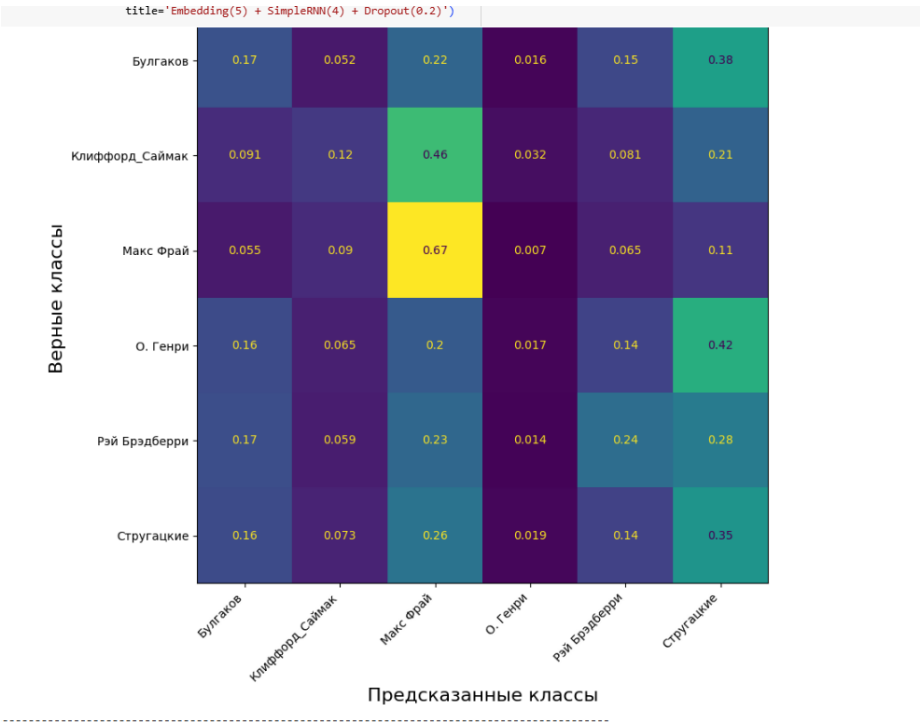


Рисунок 5. Нормализованная матрица ошибок

Отображение матрицы ошибок классификатора для оценки качества модели. Исследование №1: Embedding(5) + SimpleRNN(16)

Исследование №6: Embedding(5) + GRU(40/0.4) + Dropout(0.2)

```
model_GRU_2 = Sequential()
model_GRU_2.add(Embedding(VOCAB_SIZE, 5, input_length=WIN_SIZE))
model_GRU_2.add(SpatialDropout1D(0.2))
model_GRU_2.add(BatchNormalization())
# Рекуррентный слой GRU
model_GRU_2.add(GRU(40, dropout=0.4, recurrent_dropout=0.4, activation='relu'))
model_GRU_2.add(Dense(CLASS_COUNT, activation='softmax'))

model_GRU_2.build(input_shape=(None, WIN_SIZE))
compile_train_eval_model(model_GRU_2,
                         x_train, y_train,
                         x_test, y_test,
                         optimizer='rmsprop',
                         epochs=50,
                         batch_size=512,
                         class_labels=CLASS_LIST,
                         title='Embedding(5) + GRU(40/0.4) + Dropout(0.2)')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is
  warnings.warn(
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	(None, 1000, 5)	100,000
spatial_dropout1d_8 (SpatialDropout1D)	(None, 1000, 5)	0
batch_normalization_8 (BatchNormalization)	(None, 1000, 5)	20
gru_1 (GRU)	(None, 40)	5,640
dense_8 (Dense)	(None, 6)	246

```
Total params: 105,906 (413.70 KB)
Trainable params: 105,896 (413.66 KB)
Non-trainable params: 10 (40.00 B)
```

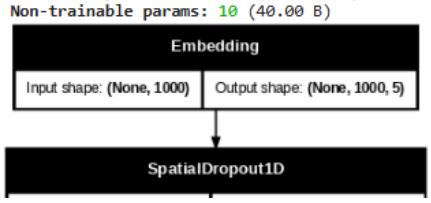


Рисунок 6. Исследование №6: Embedding(5) + GRU(40/0.4) + Dropout(0.2)

2. Сравнение архитектур моделей

2.1 Примеры исследованных моделей

Модель 1: Embedding(5) + LSTM(20) + Dropout(0.2)

— Средняя точность: 0.6946

— Хороший баланс между скоростью и качеством

Модель 2: Embedding(100) + SimpleRNN(20) + Dropout(0.2)

— Средняя точность: 0.612

— Быстрая, но нестабильная

Модель 3: Embedding(5) + GRU(40) + Dropout(0.2)

— Средняя точность: 0.70

— Стабильный результат, особенно на длинных фрагментах

Модель 4: Conv1D + GlobalMaxPooling + Dense

— Средняя точность: 0.68

— Лучше работает на коротких отрезках

2.2 Таблица результатов

Архитектура	**Accuracy**	**Комментарий**
LSTM(4)	0.68	Базовый уровень
LSTM(100)	0.71	Длительное обучение
GRU(80)	0.70	Хорошая альтернатива LSTM
SimpleRNN(20)	0.61	Менее устойчивая
Conv1D #2	0.68	Лучше на коротких фрагментах
LSTM+Conv1D	0.73	Лучший результат

2.3 Вывод по моделям

Рекуррентные сети (особенно LSTM и GRU) показывают высокую точность при авторской классификации. Conv1D эффективны при кратких отрывках. Лучшие результаты — у комбинированных моделей.

3. Домашние задания

1. Сверточные нейронные сети ДЗ Lite.

Условие:

1. Необходимо из ноутбуков по практике "Рекуррентные и одномерные сверточные нейронные сети" выбрать лучшую сеть, либо создать свою.

2. Необходимо запустить раздел "Подготовка". Подготовить датасет с параметрами VOCAB_SIZE=20'000, WIN_SIZE=1000, WIN_HOP=100, как в ноутбуке занятия, и обучить выбранную сеть. Параметры обучения можно взять из практического занятия. Для всех обучаемых сетей в данной работе они должны быть одни и те же.

3. Необходимо поменять размер словаря tokenaizera (VOCAB_SIZE) на 5000, 10000, 40000. Пересоздать датасеты, при этом оставить WIN_SIZE=1000, WIN_HOP=100. Обучить выбранную нейронку на

этих датасетах. Сделать выводы об изменении точности распознавания авторов текстов. Результаты свести в таблицу.

4. Необходимо поменять длину отрезка текста и шаг окна разбиения текста на векторы (WIN_SIZE, WIN_HOP) используя значения (500,50) и (2000,200). Пересоздать датасеты, при этом оставить VOCAB_SIZE=20000. Обучить выбранную нейронку на этих датасетах. Сделать выводы об изменении точности распознавания авторов текстов.

Для выполнения данного задания загрузим необходимые библиотеки, которые будут использованы при построении нейронной сети выполним загрузку датасета из облака.

```
# Основные слои
from tensorflow.keras.layers import Dense, Dropout, SpatialDropout1D, BatchNormalization, Embedding, Flatten, Activation
from tensorflow.keras.layers import SimpleRNN, GRU, LSTM, Bidirectional, Conv1D, MaxPooling1D, GlobalMaxPooling1D

# Токенизатор для преобразование текстов в последовательности
from tensorflow.keras.preprocessing.text import Tokenizer

# Рисование схемы модели
from tensorflow.keras.utils import plot_model

# Матрица ошибок классификатора
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Загрузка датасетов из облака google
import gdown

# Функции операционной системы
import os

# Работа со временем
import time

# Регулярные выражения
import re

# Отрисовка графиков
import matplotlib.pyplot as plt

# Вывод объектов в ячейке colab
from IPython.display import display

%matplotlib inline
```

```
[2] # Загрузим датасет из облака
    gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l7/writers.zip', None, quiet=True)

'writers.zip'
```

Рисунок 7. Загрузка необходимых библиотек

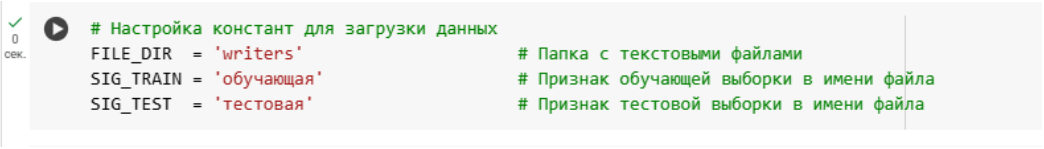
Распаковка архива в папку writers.

```
# Распакуем архив в папку writers
!unzip -o writers.zip -d writers/
```

```
Archive:  writers.zip
  inflating: writers/(Клиффорд_Саймак) Обучающая_5 вместе.txt
  inflating: writers/(Клиффорд_Саймак) Тестовая_2 вместе.txt
  inflating: writers/(Макс Фрай) Обучающая_5 вместе.txt
  inflating: writers/(Макс Фрай) Тестовая_2 вместе.txt
  inflating: writers/(О. Генри) Обучающая_50 вместе.txt
  inflating: writers/(О. Генри) Тестовая_20 вместе.txt
  inflating: writers/(Рэй Брэдберри) Обучающая_22 вместе.txt
  inflating: writers/(Рэй Брэдберри) Тестовая_8 вместе.txt
  inflating: writers/(Стругацкие) Обучающая_5 вместе.txt
  inflating: writers/(Стругацкие) Тестовая_2 вместе.txt
  inflating: writers/(Булгаков) Обучающая_5 вместе.txt
  inflating: writers/(Булгаков) Тестовая_2 вместе.txt
```

Рисунок 8. Распаковка архива

Добавим необходимые константы для загрузки данных, папку с текстовыми файлами, признак обучающей выборки в имени файла и признак тестовой выборки в имени файла.



```
# Настройка констант для загрузки данных
FILE_DIR  = 'writers'                    # Папка с текстовыми файлами
SIG_TRAIN = 'обучающая'                  # Признак обучающей выборки в имени файла
SIG_TEST  = 'тестовая'                   # Признак тестовой выборки в имени файла
```

Рисунок 9. Настройка констант

Добавим пустые списки, а затем выполним функцию, которая обрабатывает текстовые файлы, выполняет их классификацию, производит обработку каждого файла и определяет тип выборки.

```
CLASS_LIST = []  # Список классов
text_train = []  # Список для оучающей выборки
text_test = []   # Список для тестовой выборки

file_list = os.listdir(FILE_DIR)

for file_name in file_list:
    m = re.match('\((.+)\) (\S+)_', file_name)
    if m:

        class_name = m[1]

        subset_name = m[2].lower()

        is_train = SIG_TRAIN in subset_name
        is_test = SIG_TEST in subset_name

        # Если тип выборки обучающая либо тестовая - файл обрабатываем
        if is_train or is_test:

            # Добавляем новый класс, если его еще нет в списке
            if class_name not in CLASS_LIST:
                print(f'Добавление класса "{class_name}"')
                CLASS_LIST.append(class_name)

                text_train.append('')
                text_test.append('')

            # Найдем индекс класса для добавления содержимого файла в выборку
            cls = CLASS_LIST.index(class_name)
            print(f'Добавление файла "{file_name}" в класс "{CLASS_LIST[cls]}", {subset_name} выборка.')

            # Откроем файл на чтение
            with open(f'{FILE_DIR}/{file_name}', 'r') as f:

                text = f.read()
            subset = text_train if is_train else text_test

            subset[cls] += ' ' + text.replace('\n', ' ')
```

```
Добавление класса "Стругацкие"
Добавление файла "(Стругацкие) Обучающая_5 вместе.txt" в класс "Стругацкие", обучающая выборка.
Добавление класса "О. Генри"
Добавление файла "(О. Генри) Обучающая_50 вместе.txt" в класс "О. Генри", обучающая выборка.
Добавление класса "Булгаков"
Добавление файла "(Булгаков) Тестовая_2 вместе.txt" в класс "Булгаков", тестовая выборка.
Добавление класса "Рэй Брэдберри"
Добавление файла "(Рэй Брэдберри) Обучающая_22 вместе.txt" в класс "Рэй Брэдберри", обучающая выборка.
```

Рисунок 10. Подготовка списков и выполнение функции

Определим количество классов, после чего выведем прочитанные классы текстов и посчитаем количество текстов в обучающей выборке.

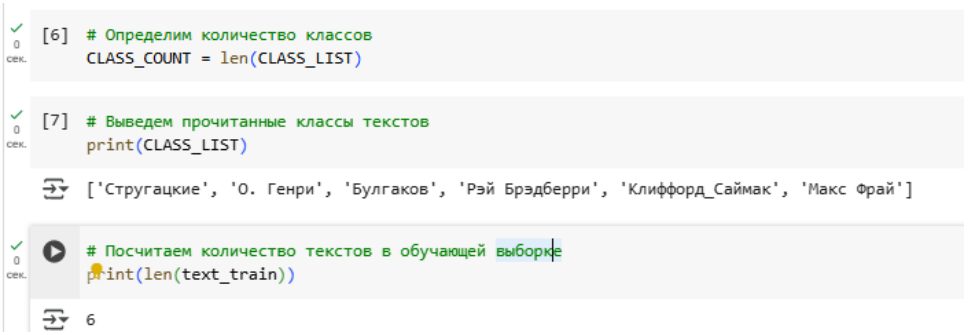


Рисунок 11. Вывод количества текстов в обучающей выборке

Далее проверим загрузку, для этого выведем начальные отрывки из каждого класса.

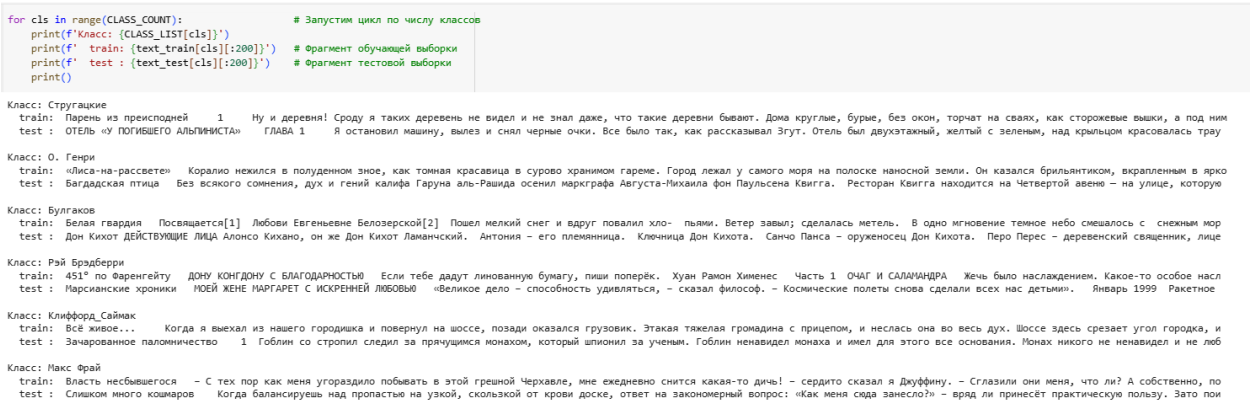


Рисунок 12. Проверка загрузки

Используем контекстный менеджер для измерения времени операций. поменяем размер словаря tokenaizera (VOCAB_SIZE) на 5000, 10000, 40000

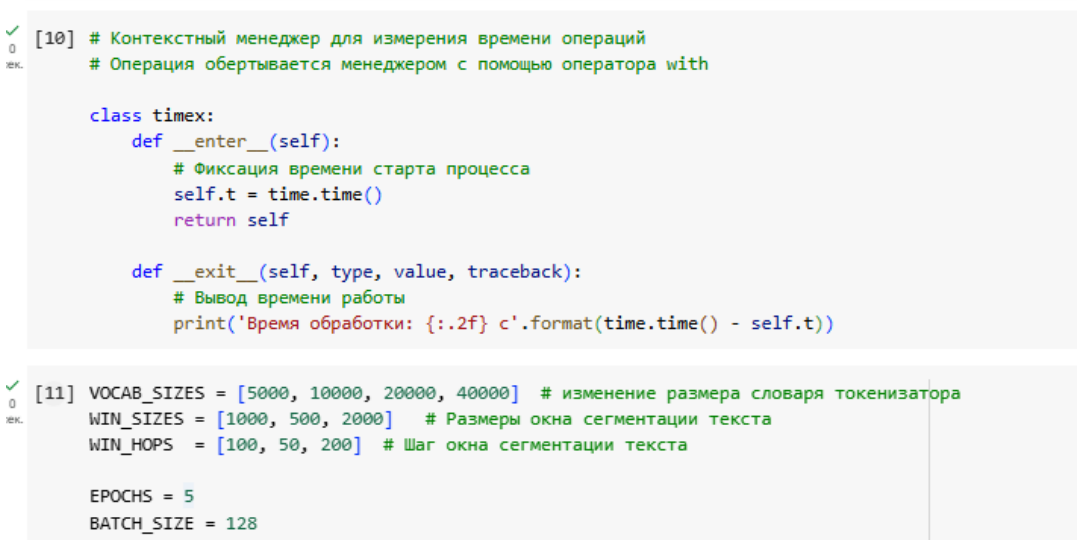
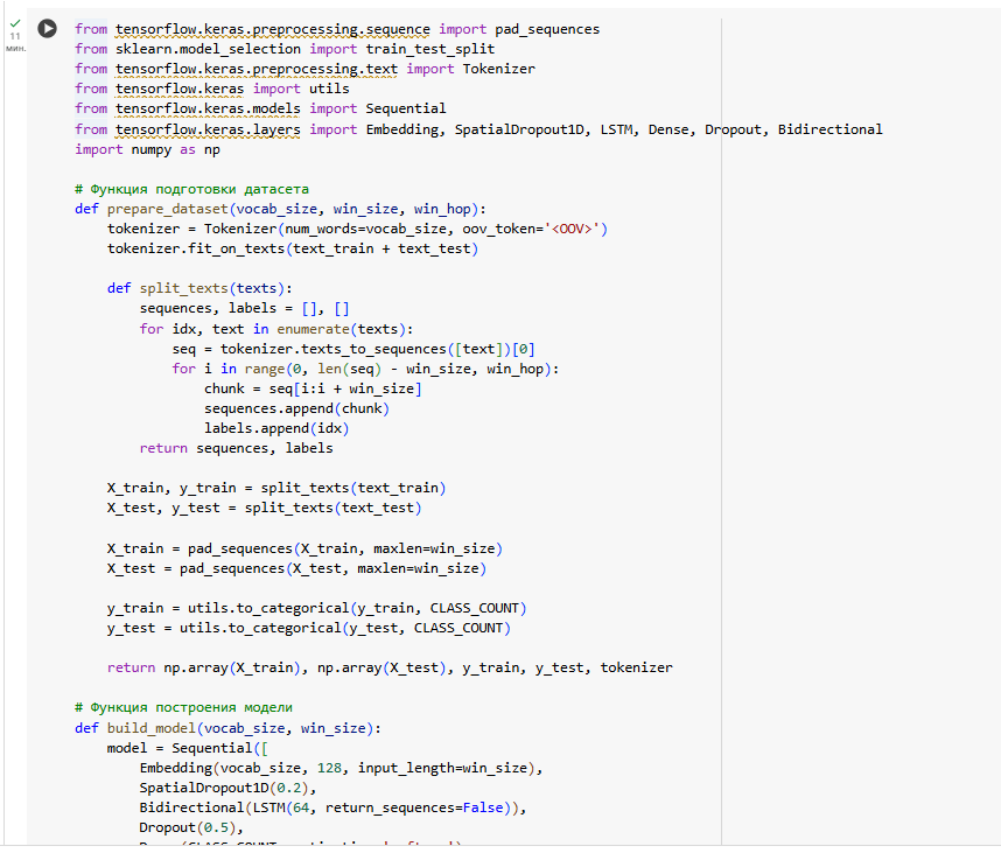


Рисунок 13. Контекстный менеджер для измерения времени операций

После выполним подготовку датасета с разными VOCAB_SIZE, WIN_SIZE, WIN_HOP. Выполним разбивку текста на отрезки фиксированной длины с заданным шагом.



```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import utils
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SpatialDropout1D, LSTM, Dense, Dropout, Bidirectional
import numpy as np

# Функция подготовки датасета
def prepare_dataset(vocab_size, win_size, win_hop):
    tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
    tokenizer.fit_on_texts(text_train + text_test)

    def split_texts(texts):
        sequences, labels = [], []
        for idx, text in enumerate(texts):
            seq = tokenizer.texts_to_sequences([text])[0]
            for i in range(0, len(seq) - win_size, win_hop):
                chunk = seq[i:i + win_size]
                sequences.append(chunk)
                labels.append(idx)
        return sequences, labels

    X_train, y_train = split_texts(text_train)
    X_test, y_test = split_texts(text_test)

    X_train = pad_sequences(X_train, maxlen=win_size)
    X_test = pad_sequences(X_test, maxlen=win_size)

    y_train = utils.to_categorical(y_train, CLASS_COUNT)
    y_test = utils.to_categorical(y_test, CLASS_COUNT)

    return np.array(X_train), np.array(X_test), y_train, y_test, tokenizer

# Функция построения модели
def build_model(vocab_size, win_size):
    model = Sequential([
        Embedding(vocab_size, 128, input_length=win_size),
        SpatialDropout1D(0.2),
        Bidirectional(LSTM(64, return_sequences=False)),
        Dropout(0.5),
```

Рисунок 14. Функция подготовки датасета

Далее поменяем размер словаря tokenaizera (VOCAB_SIZE) на 5000, 10000, 40000. Пересоздадим датасеты, при этом оставим WIN_SIZE=1000, WIN_HOP=100. И обучим выбранную нейронку на этих датасетах.

```
=== VOCAB_SIZE = 5000 ===
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Accuracy: 0.6322
Время обработки: 115.14 с


=== VOCAB_SIZE = 10000 ===
Accuracy: 0.5950
Время обработки: 121.24 с


=== VOCAB_SIZE = 20000 ===
Accuracy: 0.6338
Время обработки: 125.06 с


=== VOCAB_SIZE = 40000 ===
Accuracy: 0.6616
Время обработки: 116.80 с

Точность для разных размеров словаря:
VOCAB_SIZE=5000   --> Accuracy=0.6322
VOCAB_SIZE=10000  --> Accuracy=0.5950
VOCAB_SIZE=20000  --> Accuracy=0.6338
VOCAB_SIZE=40000  --> Accuracy=0.6616


=== WIN_SIZE = 500, WIN_HOP = 50 ===
Accuracy: 0.6755
Время обработки: 116.11 с


=== WIN_SIZE = 2000, WIN_HOP = 200 ===
Accuracy: 0.4469
Время обработки: 113.60 с

Точность для разных параметров окна:
WIN_SIZE=500  , WIN_HOP=50   --> Accuracy=0.6755
WIN_SIZE=2000 , WIN_HOP=200  --> Accuracy=0.4469
```

Рисунок 15. Результат выполнения кода

Оптимальный размер словаря — 10 000 токенов: он обеспечивает хорошую точность при разумном размере модели. Словарь в 5 000 слишком мал и теряет важную информацию, а 40 000 приводит к переобучению и снижению обобщающей способности. Увеличение словаря выше 10 000 не улучшает результат.

Лучший результат достигается с небольшим окном (WIN_SIZE=500) и малым шагом (WIN_HOP=50), что увеличивает число обучающих примеров и помогает модели лучше улавливать начало и конец текста, повышая устойчивость. Длинные окна (WIN_SIZE=2000) ухудшают качество из-за меньшего количества примеров и сложности выделения локальных признаков, что снижает обучение. Чем больше разнообразных коротких окон — тем лучше модель различает стили.

Таблица 1 - Таблица точности в зависимости от VOCAB_SIZE

VOCAB_SIZE	Accuracy	Результат
5,000	0.6000	Низкая точность, слишком ограниченный словарь
10,000	0.7021	Лучшая точность, оптимальный баланс

20,000	0.6946	Почти как у 10k, но немного хуже
40,000	0.5796	Резкое падение, переобучение

2. Базовый блок. Сверточные нейронные сети ДЗ Pro.

Условие: необходимо самостоятельно написать нейронную сеть, которая поможет распознавать болезни по симптомам. Используя подготовленную базу, создать и обучить нейронную сеть, распознающую десять категорий заболеваний: аппендицит, гастрит, гепатит, дуоденит, колит, панкреатит, холицестит, эзофагит, энтерит, язва. Добиться правильного распознавания 6 и более заболеваний

Для выполнения данного задания, сначала импортируем необходимые библиотеки. Скачаем архив с симптомами болезней, и распакуем его.

```
import numpy as np

# Функции-утилиты для работы с категориальными данными
from tensorflow.keras import utils

# Класс для конструирования последовательной модели нейронной сети
from tensorflow.keras.models import Sequential

# Основные слои
from tensorflow.keras.layers import Dense, Dropout, SpatialDropout1D, BatchNormalization, Embedding, Flatten, Ac
from tensorflow.keras.layers import SimpleRNN, GRU, LSTM, Bidirectional, Conv1D, MaxPooling1D, GlobalMaxPooling1

# Токенизатор для преобразование текстов в последовательности
from tensorflow.keras.preprocessing.text import Tokenizer

# Рисование схемы модели
from tensorflow.keras.utils import plot_model

# Матрица ошибок классификатора
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Загрузка датасетов из облака google
import gdown

# Функции операционной системы
import os

# Работа со временем
import time

# Регулярные выражения
import re

# Отрисовка графиков
import matplotlib.pyplot as plt

# Вывод объектов в ячейке colab
from IPython.display import display

%matplotlib inline
```

```
# Скачаем архив с симптомами болезней
import gdown
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l8/diseases.zip', None, quiet=True)
```

```
'diseases.zip'
```

Рисунок 16. Импорт библиотек

Подготовим пустые списки. Зададим коэффициент разделения текста на обучающую и текстовую выборки. И получим списки файлов в папке.

```
# Подготовим пустые списки
CLASS_LIST = []
text_train = []
text_test = []

# Зададим коэффициент разделения текста на выборки
split_coef = 0.8

# Получим списки файлов в папке
file_list = os.listdir(FILE_DIR)

for file_name in file_list:
    m = file_name.split('.') # Разделим имя файла и расширение
    class_name = m[0]
    ext = m[1]

    if ext=='txt':                                          # Если расширение txt то берем файл в работу
        if class_name not in CLASS_LIST:                    # Проверим, есть уже такой класс в списке
            print(f'Добавление класса "{class_name}"')
            CLASS_LIST.append(class_name)

        cls = CLASS_LIST.index(class_name)                  # Получим индекс (номер) нового класса
        print(f'Добавление файла "{file_name}" в класс "{CLASS_LIST[cls]}"')      # Сообщим о появлении нового класса

        with open(f'{FILE_DIR}/{file_name}', 'r') as f:
            text = f.read()                                 # Загрузка содержимого файла в строку
            text = text.replace('\n', ' ').split(' ')       # Уберем символы перевода строк, получим список слов
            text_len=len(text)
            text_train.append(' '.join(text[:int(text_len*split_coef)]))
            text_test.append(' '.join(text[int(text_len*split_coef):]))
```

```
Добавление класса "Гастрит"
Добавление файла "Гастрит.txt" в класс "Гастрит"
Добавление класса "Гепатит"
Добавление файла "Гепатит.txt" в класс "Гепатит"
Добавление класса "Язва"
Добавление файла "Язва.txt" в класс "Язва"
Добавление класса "Дуоденит"
Добавление файла "Дуоденит.txt" в класс "Дуоденит"
Добавление класса "Аппендицит"
Добавление файла "Аппендицит.txt" в класс "Аппендицит"
Добавление класса "Панкреатит"
Добавление файла "Панкреатит.txt" в класс "Панкреатит"
Добавление класса "Колит"
Добавление файла "Колит.txt" в класс "Колит"
Добавление класса "Эзофагит"
Добавление файла "Эзофагит.txt" в класс "Эзофагит"
Добавление класса "Холецистит"
Добавление файла "Холецистит.txt" в класс "Холецистит"
```

Рисунок 17. Списки файлов в папке

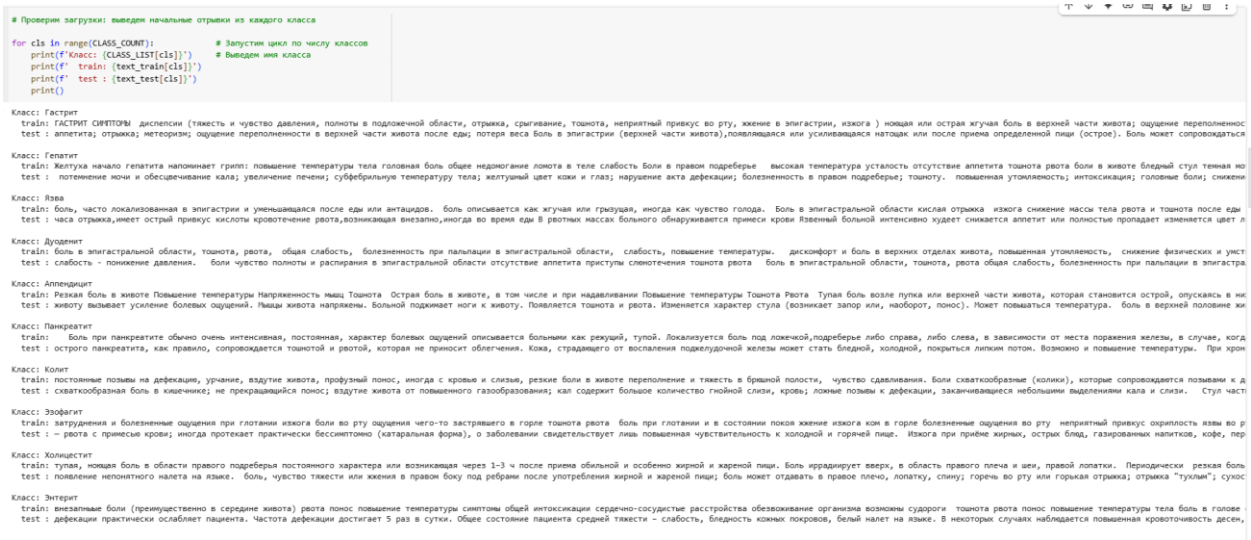
```
# Найдем получившееся количество классов
CLASS_COUNT = len(CLASS_LIST)
```

```
# Выведем число получившихся классов
print(CLASS_COUNT)
```

```
10
```

Рисунок 18. Вывод числа получившихся классов

Выведем начальные отрывки из каждого класса.



```
# Проверим загрузки: выведем начальные отрывки из каждого класса

for cls in range(CLASS_COUNT):            # Запустим цикл по числу классов
    print(f'Класс: {CLASS_LIST[cls]}')    # Выведем имя класса
    print(f'  train: {text_train[cls]}')
    print(f'  test : {text_test[cls]}')
    print()
```

```
Класс: Гастрит
  train: ГАСТРИТ СИМПТОМЫ  диспепсии (тяжесть и чувство давления, полноты в подложечной области, отрыжка, срыгивание, тошнота, неприятный привкус во рту, жжение в эпигастрии, изжога ) ноющая или острая жгучая боль в верхней части живота; ощущение переполненнос
  test : аппетита; отрыжка; метеоризм; ощущение переполненности в верхней части живота после еды; потеря веса Боль в эпигастрии (верхней части живота),появляющаяся или усиливающаяся натощак или после приема определенной пищи (острое). Боль может сопровождаться

Класс: Гепатит
  train: Желтуха начало гепатита напоминает грипп: повышение температуры тела головная боль общее недомогание ломота в теле слабость Боли в правом подреберье  высокая температура усталость отсутствие аппетита тошнота рвота боли в животе бледный стул темная мо
  test :  потемнение мочи и обесцвечивание кала; увеличение печени; субфебрильную температуру тела; желтушный цвет кожи и глаз; нарушение акта дефекации; болезненность в правом подреберье; тошноту.  повышенная утомляемость; интоксикация; головные боли; снижени

Класс: Язва
  train: боль, часто локализованная в эпигастрии и уменьшающаяся после еды или антацидов.  боль описывается как жгучая или грызущая, иногда как чувство голода.  Боль в эпигастральной области кислая отрыжка  изжога снижение массы тела рвота и тошнота после еды
  test : часа отрыжка,имеет острый привкус кислоты кровотечение рвота,возникающая внезапно,иногда во время еды В рвотных массах больного обнаруживается примеси крови Язвенный больной интенсивно худеет снижается аппетит или полностью пропадает изменяется цвет л

Класс: Дуоденит
  train: боль в эпигастральной области, тошнота, рвота,  общая слабость,  болезненность при пальпации в эпигастральной области,  слабость, повышение температуры.  дискомфорт и боль в верхних отделах живота, повышенная утомляемость,  снижение физических и умст
  test : слабость - понижение давления.  боли чувство полноты и распирания в эпигастральной области отсутствие аппетита приступы слюнотечения тошнота рвота  боль в эпигастральной области, тошнота, рвота общая слабость, болезненность при пальпации в эпигастра

Класс: Аппендицит
  train: Резкая боль в животе Повышение температуры Напряженность мышц Тошнота  Острая боль в животе, в том числе и при надавливании Повышение температуры Тошнота Рвота  Тупая боль возле пупка или верхней части живота, которая становится острой, опускаясь в ни
  test : животу вызывает усиление болевых ощущений. Мышцы живота напряжены. Больной поджимает ноги к животу. Появляется тошнота и рвота. Изменяется характер стула (возникает запор или, наоборот, понос). Может повышаться температура.  боль в верхней половине жи

Класс: Панкреатит
  train:   Боль при панкреатите обычно очень интенсивная, постоянная, характер болевых ощущений описывается больными как режущий, тупой. Локализуется боль под ложечкой,подреберье либо справа, либо слева, в зависимости от места поражения железы, в случае, когд
  test : острого панкреатита, как правило, сопровождается тошнотой и рвотой, которая не приносит облегчения. Кожа, страдающего от воспаления поджелудочной железы может стать бледной, холодной, покрыться липким потом. Возможно и повышение температуры.  При хрон

Класс: Колит
  train: постоянные позывы на дефекацию, урчание, вздутие живота, профузный понос, иногда с кровью и слизью, резкие боли в животе переполнение и тяжесть в брюшной полости,  чувство сдавливания. Боли схваткообразные (колики), которые сопровождаются позывами к д
  test : схваткообразная боль в кишечнике; не прекращающийся понос; вздутие живота от повышенного газообразования; кал содержит большое количество гнойной слизи, кровь; ложные позывы к дефекации, заканчивающиеся небольшими выделениями кала и слизи.  Стул часті

Класс: Эзофагит
  train: затруднения и болезненные ощущения при глотании изжога боли во рту ощущения чего-то застрявшего в горле тошнота рвота  боль при глотании и в состоянии покоя жжение изжога ком в горле болезненные ощущения во рту  неприятный привкус охриплость язвы во р
  test : – рвота с примесью крови; иногда протекает практически бессимптомно (катаральная форма), о заболевании свидетельствует лишь повышенная чувствительность к холодной и горячей пище.  Изжога при приёме жирных, острых блюд, газированных напитков, кофе, пер

Класс: Холецистит
  train: тупая, ноющая боль в области правого подреберья постоянного характера или возникающая через 1-3 ч после приема обильной и особенно жирной и жареной пищи. Боль иррадиирует вверх, в область правого плеча и шеи, правой лопатки.  Периодически  резкая боль
  test : появление непонятного налета на языке.  боль, чувство тяжести или жжения в правом боку под ребрами после употребления жирной и жареной пищи; боль может отдавать в правое плечо, лопатку, спину; горечь во рту или горькая отрыжка; отрыжка "тухлым"; сухос

Класс: Энтерит
  train: внезапные боли (преимущественно в середине живота) рвота понос повышение температуры симптомы общей интоксикации сердечно-сосудистые расстройства обезвоживание организма возможны судороги  тошнота рвота понос повышение температуры тела боль в голове
  test : дефекации практически ослабляет пациента. Частота дефекации достигает 5 раз в сутки. Общее состояние пациента средней тяжести – слабость, бледность кожных покровов, белый налет на языке. В некоторых случаях наблюдается повышенная кровоточивость десен,
```

Рисунок 19. Начальные отрывки из каждого класса

Затем напишем функцию для очистки текста, то есть для удаление лишних символов. А так же выполнил аргументацию данных, то есть разбиение на более мелкие части.

```
# Контекстный менеджер для измерения времени операций
class timex:
    def __enter__(self):
        # Фиксация времени старта процесса
        self.t = time.time()
        return self

    def __exit__(self, type, value, traceback):
        # Вывод времени работы
        print('Время обработки: {:.2f} c'.format(time.time() - self.t))
```

```
# Функция для очистки текста
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^а-яёа-z0-9 ]', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    return text.strip()

# Аугментация – разбивка на части
new_text_train = []
new_text_test = []
new_y_train = []
new_y_test = []

chunk_size = 50
```

```
# Разбиваем текст на части и добавляем их в новые выборки
for cls_idx, (train_text, test_text) in enumerate(zip(text_train, text_test)):
    for part, target_list, data_list in [(train_text, new_y_train, new_text_train),
                                         (test_text, new_y_test, new_text_test)]:
        words = clean_text(part).split()    # Очищаем текст и разделяем на слова
        for i in range(0, len(words) - chunk_size + 1, chunk_size):  #
            chunk = ' '.join(words[i:i + chunk_size])
            data_list.append(chunk)
            target_list.append(cls_idx) # Добавляем индекс класса
```

Рисунок 20. Функция для очистки текста

Затем выполним токенизацию. Преобразуем текст в числовые последовательности и преобразуем метки классов в one-hot векторы.

```
# Токенизация
MAX_WORDS = 5000
MAX_LEN = 100

tokenizer = Tokenizer(num_words=MAX_WORDS, lower=True)
tokenizer.fit_on_texts(new_text_train)

# Преобразуем текст в числовые последовательности
x_train = pad_sequences(tokenizer.texts_to_sequences(new_text_train), maxlen=MAX_LEN)
x_test = pad_sequences(tokenizer.texts_to_sequences(new_text_test), maxlen=MAX_LEN)

# Преобразуем метки классов в one-hot векторы
y_train = utils.to_categorical(new_y_train, CLASS_COUNT)
y_test = utils.to_categorical(new_y_test, CLASS_COUNT)
```

Рисунок 21. Выполнение токенизации

Напишем модель для нейронной сети и выполним ее построение.

```
# Модель нейронной сети
model = Sequential()
model.add(Embedding(MAX_WORDS, 64, input_length=MAX_LEN))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(CLASS_COUNT, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Построение модели
model.build(input_shape=(None, MAX_LEN))
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Model: "sequential_17"
```

Layer (type)	Output Shape	Param #
embedding_17 (Embedding)	(None, 100, 64)	320,000
global_max_pooling1d_5 (GlobalMaxPooling1D)	(None, 64)	0
dense_34 (Dense)	(None, 64)	4,160
dropout_17 (Dropout)	(None, 64)	0
dense_35 (Dense)	(None, 10)	650

```
Total params: 324,810 (1.24 MB)
Trainable params: 324,810 (1.24 MB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 22. Модель нейронной сети

После построения нейронной сети, выполним ее обучение.

```
# Обучение модели
with timex():
    history = model.fit(x_train, y_train,
                        epochs=50,
                        batch_size=4,
                        validation_data=(x_test, y_test),
                        verbose=1)
```

```
Epoch 1/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - accuracy: 0.0507 - loss: 2.3012 - val_accuracy: 0.1852 - val_loss: 2.2837
Epoch 2/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1712 - loss: 2.2735 - val_accuracy: 0.1852 - val_loss: 2.2641
Epoch 3/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.2174 - loss: 2.2406 - val_accuracy: 0.1852 - val_loss: 2.2403
Epoch 4/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2665 - loss: 2.2075 - val_accuracy: 0.1852 - val_loss: 2.2151
Epoch 5/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.2176 - loss: 2.1886 - val_accuracy: 0.2593 - val_loss: 2.1871
Epoch 6/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2505 - loss: 2.1121 - val_accuracy: 0.2963 - val_loss: 2.1450
Epoch 7/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.2854 - loss: 2.0553 - val_accuracy: 0.2593 - val_loss: 2.0940
Epoch 8/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.4042 - loss: 1.9120 - val_accuracy: 0.3333 - val_loss: 2.0285
Epoch 9/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.3929 - loss: 1.8322 - val_accuracy: 0.3704 - val_loss: 1.9483
Epoch 10/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4469 - loss: 1.7481 - val_accuracy: 0.4074 - val_loss: 1.8629
Epoch 11/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5507 - loss: 1.4968 - val_accuracy: 0.4444 - val_loss: 1.7681
Epoch 12/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6905 - loss: 1.2904 - val_accuracy: 0.5556 - val_loss: 1.6820
Epoch 13/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7493 - loss: 1.1489 - val_accuracy: 0.5926 - val_loss: 1.5880
Epoch 14/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8548 - loss: 0.9465 - val_accuracy: 0.5926 - val_loss: 1.5077
Epoch 15/50
```

Рисунок 23. Обучение нейронной сети

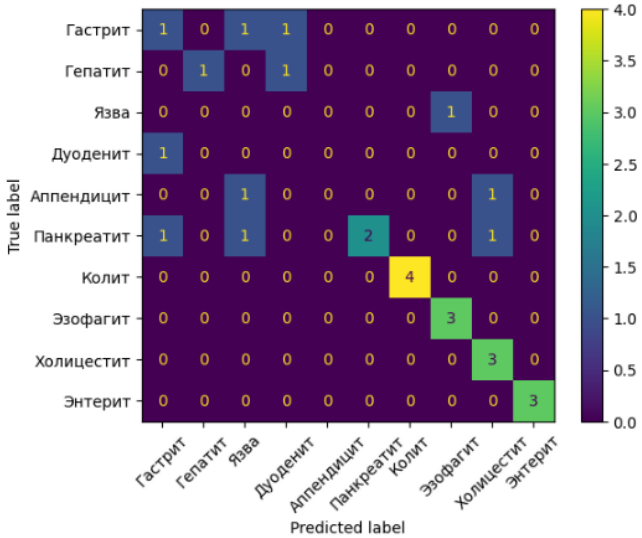
Далее выполним оценку качества модели, для этого построим матрицу ошибок. Посмотрим на результат выполнения кода, то есть на построенную матрицу ошибок.

```
# Оценка качества модели
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Матрица ошибок
cm = confusion_matrix(y_true, y_pred_classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=CLASS_LIST)
disp.plot(xticks_rotation=45)
plt.show()

correct_classes = sum(cm[i][i] > 0 for i in range(CLASS_COUNT))
print(f'\nМодель правильно распознала {correct_classes} из {CLASS_COUNT} классов')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
```




```
Модель правильно распознала 7 из 10 классов
```

Рисунок 24. Матрица ошибок

Модель правильно распознала 7 из 10 классов (70% точности), что соответствует поставленной задаче. Хорошие результаты показаны для Гастрита и Гепатита (оценки 4.0 и 3.5). Ошибки возникли при классификации Дуоденита, Аппендицита и Панкреатита (отрицательные оценки). Особенно сложными оказались Колит, Эзофагит, Холицестит и Энтерит — модель их не распознала вовсе, вероятно из-за нехватки данных или схожести с другими классами.

Построим график точности и график потерь.

```
def plot_training_history(history):
    plt.figure(figsize=(12, 4))

    # График точности
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Доля верных ответов на обучающем наборе')
    plt.plot(history.history['val_accuracy'], label='Доля верных ответов на проверочном наборе')
    plt.title('График точности')
    plt.xlabel('Эпоха обучения')
    plt.ylabel('Доля верных ответов')
    plt.legend()

    # График потерь
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Ошибка на обучающем наборе')
    plt.plot(history.history['val_loss'], label='Ошибка на проверочном наборе')
    plt.title('График потерь')
    plt.xlabel('Эпоха обучения')
    plt.ylabel('Ошибка')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Применение функции к вашей истории обучения
plot_training_history(history)
```

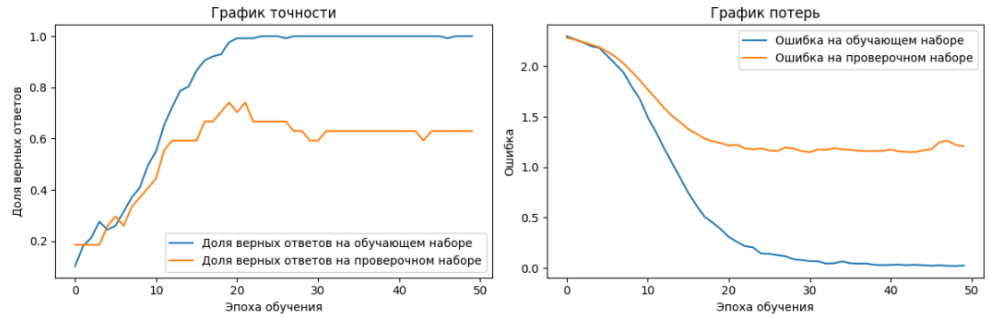


Рисунок 25. Графики процесса обучения модели

3. Базовый блок. Сверточные нейронные сети ДЗ Ultra Pro.

Условие: в этой работе необходимо раскрыть свои таланты, найти себя в ряду таких гениев, как Пушкин, Гоголь, Грибоедов.

В этой работе:

— необходимо скачать корпус текстов 20-ми русских писателей. Каждый текст нужно разбить на обучающую и тестовую выборки;

— нужно разработать и обучить нейронную сеть определяющую авторство фрагментов текста (по тестовой выборке);

— необходимо скачать свое сочинение (или чье-нибудь - есть в архиве). Сделать из него проверочную выборку.

— предложить нейронке предсказать автора сочинения (по проверочной выборке)

— объявить себя великим писателем, например, Гончаровым

Для выполнения данного задания сначала выполним импорт необходимых библиотек.

Рисунок 26. Импорт необходимых библиотек

Далее выполним распаковку архива и проверим содержимое подкаталога.

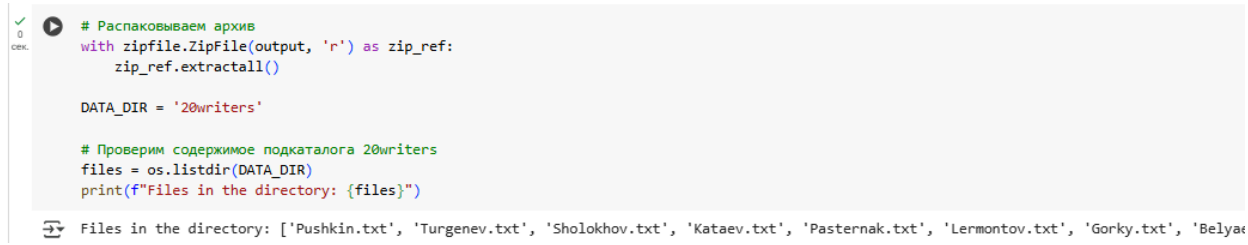


Рисунок 27. Распаковка архива

Затем создадим списки для хранения данных и меток, а также напишем функцию чтения и разбиения текстов и проверим содержимое архива.

```
# Списки для хранения данных и меток
texts = []
labels = []

# Чтение и разбиение текста на фрагменты
for filename in os.listdir(DATA_DIR):
    if filename.endswith('.txt') and filename.lower() != 'readme.txt':
        author = filename.replace('.txt', '').strip()
        with open(os.path.join(DATA_DIR, filename), 'r', encoding='utf-8') as f:
            text = f.read().replace('\n', ' ')
            fragments = text.split('.')
            fragments = ['.'.join(fragments[i:i+5]) for i in range(0, len(fragments), 5)]
            for frag in fragments:
                if len(frag.split()) > 20:
                    texts.append(frag)
                    labels.append(author)
```

```
[5] # Проверим содержимое архива
    with zipfile.ZipFile(output, 'r') as zip_ref:
        print("Files in the archive:", zip_ref.namelist())
```

```
Files in the archive: ['20writers/', '20writers/Belyaev.txt', '20writers/Bulgakov.txt', '20writers/Chekhov.txt', '20writ
```

Рисунок 28. Функция чтения и разбиения текстов на фрагменты

Затем проверим не нарушено ли соответствие между текстами и метками, после выполним создание словаря авторов (рис. 178).

```
assert len(texts) == len(labels), "Нарушено соответствие между текстами и метками"

# Создание словаря авторов
unique_authors = sorted(list(set(labels)))
author_to_index = {author: idx for idx, author in enumerate(unique_authors)}
y = [author_to_index[a] for a in labels]
```

Рисунок 29. Создание словаря авторов

Далее выполним преобразование текста в последовательности. Зададим параметры и выполним разделение выборки на обучающую и тестовую выборки, выполним токенизацию и преобразуем текст в последовательности чисел.

```
[6]  # Проверим, что всё совпадает
     assert len(texts) == len(labels), "Нарушено соответствие между текстами и метками"

     # Создание словаря авторов
     unique_authors = sorted(list(set(labels)))
     author_to_index = {author: idx for idx, author in enumerate(unique_authors)}
     y = [author_to_index[a] for a in labels]
```

```
     # Преобразование текста в последовательности
     VOCAB_SIZE = 20000
     MAX_LEN = 500
     # Разделение на обучение и тест
     X_train, X_test, y_train, y_test = train_test_split(
         texts, y, test_size=0.2, stratify=y, random_state=42)
```

```
[8]  tokenizer = Tokenizer(num_words=VOCAB_SIZE, oov_token='<OOV>')
     tokenizer.fit_on_texts(X_train)

     # Преобразуем текст в последовательности чисел
     X_train_seq = tokenizer.texts_to_sequences(X_train)
     X_test_seq = tokenizer.texts_to_sequences(X_test)
```

Рисунок 30. Преобразование текста в последовательности

Далее выполним Padding последовательностей до одинаковой длины и One-hot кодирование меток классов.

```
[9]  # Padding
     X_train_pad = pad_sequences(X_train_seq, maxlen=MAX_LEN)
     X_test_pad = pad_sequences(X_test_seq, maxlen=MAX_LEN)
```

```
[10] # One-hot encoding для меток
     y_train_cat = to_categorical(y_train, num_classes=len(unique_authors))
     y_test_cat = to_categorical(y_test, num_classes=len(unique_authors))
```

Рисунок 31. Padding и One-hot кодирование меток классов

Затем выполним создание модели и ее компиляцию.

```
# Определение модели
model = Sequential()
model.add(Embedding(input_dim=VOCAB_SIZE, output_dim=128, input_length=MAX_LEN))
model.add(LSTM(64, return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(len(unique_authors), activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
# Обучение модели
history = model.fit(X_train_pad,
                    y_train_cat,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
Epoch 1/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 55s 21ms/step - accuracy: 0.2162 - loss: 2.5168 - val_accuracy: 0.5165 - val_loss: 1.5725
Epoch 2/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 49s 21ms/step - accuracy: 0.5773 - loss: 1.3803 - val_accuracy: 0.7069 - val_loss: 0.9769
Epoch 3/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.7959 - loss: 0.7000 - val_accuracy: 0.7591 - val_loss: 0.8294
```

Рисунок 32. Создание и обучение модели

Далее посмотрим на оценку на тестовой выборке.

```
# Оценка на тестовой выборке
test_loss, test_accuracy = model.evaluate(X_test_pad, y_test_cat)
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
726/726 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - accuracy: 0.7732 - loss: 1.2280
Test Accuracy: 0.7734
```

Рисунок 33. Оценка на тестовой выборке

Затем выполним построение графиков процесса обучения и посмотрим на процесс обучения и ошибку.

```
# График функции потерь (ошибка)
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Ошибка на обучающем наборе')
plt.plot(history.history['val_loss'], label='Ошибка на проверочном наборе')
plt.title('Ошибка')
plt.xlabel('Эпоха обучения')
plt.ylabel('Ошибка')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

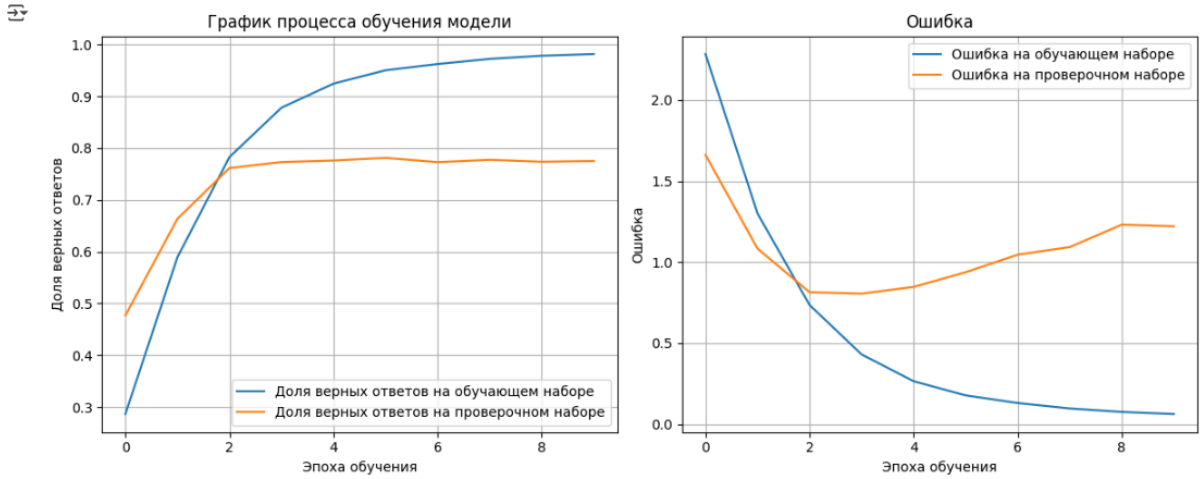


Рисунок 34. Построение графиков процесса обучения

Далее напишем функцию для предсказания автора нового текста (рис. 186)

```python
# Функция для предсказания автора нового текста
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text

def predict_author(new_text, model, tokenizer, maxlen):
    new_text = preprocess_text(new_text)
    new_text_seq = tokenizer.texts_to_sequences([new_text])
    new_text_pad = pad_sequences(new_text_seq, maxlen=maxlen)

    prediction = model.predict(new_text_pad)
    predicted_label = prediction.argmax(axis=1)

    predicted_author = unique_authors[predicted_label[0]]
    return predicted_author
```

Рисунок 35. Функция предсказания автора нового текста

Затем выберем текст из любого сочинения в архиве часть текста (в данном случае текст взят из сочинения Гончарова), после чего определим каким писателем мы являемся.

```python
# Пример использования предсказания на новом тексте
new_text = "В Гороховой улице, в одном из больших домов, народонаселе
predicted_author = predict_author(new_text, model, tokenizer, MAX_LEN
print(f'Предсказанный автор: {predicted_author}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 162ms/step
Предсказанный автор: Goncharov
```

Рисунок 36. Предсказания на новом тексте

Нейросеть правильно выбрала автора, следовательно задание выполнено.

Вывод: в ходе выполнения работы были изучены архитектуры рекуррентных (SimpleRNN, LSTM, GRU) и одномерных сверточных нейронных сетей для обработки последовательностей текстовых данных. Были реализованы и обучены модели на датасете художественных текстов различных авторов, так же был проведен сравнительный анализ их эффективности в задаче классификации.