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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ

По лабораторной работе №5

Дисциплины «Основы нейронных сетей»

Выполнил:

Говоров Егор Юрьевич

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной
техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента цифровых и робототехнических систем и электроники и института перспективной инженерии

_____

(подпись)

Ставрополь, 2024 г.

Тема: Исследование рекуррентных и одномерных сверточных нейронных сетей для классификации текстовых данных

Цель: изучить архитектуры рекуррентных (SimpleRNN, LSTM, GRU) и одномерных сверточных нейронных сетей для обработки последовательностей текстовых данных.

Ссылка: https://github.com/Artorias1469/NN_5.git

Ход работы:

Выполнение индивидуальных заданий:

Задание 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

```python
# Стандартная библиотека
import os
import re
import time
import warnings
import zipfile  # Для работы с архивами

# Сторонние библиотеки
import gdown
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import display
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from tensorflow.keras import utils
from tensorflow.keras.layers import (
    Activation,
    BatchNormalization,
    Bidirectional,
    Conv1D,
    Dense,
    Dropout,
    Embedding,
    Flatten,
    GlobalMaxPooling1D,
    GRU,
    LSTM,
    MaxPooling1D,
    SimpleRNN,
    SpatialDropout1D,
)
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import plot_model

# Подавление предупреждений
warnings.filterwarnings("ignore")

# Отображение графиков в ноутбуке
%matplotlib inline
```

Рисунок 1. Загрузка необходимых библиотек

Далее выполним загрузку датасета из облака

```python
if not os.path.exists('writers.zip'):
    # Загрузим датасет из облака
    gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l7/writers.zip', None, quiet=True)
```

```
'writers.zip'
```

Рисунок 2. Загрузка датасета Затем

распакуем архив в папку writers

```python
# Распакуем архив в папку writers
if not os.path.exists('writers'):
    with zipfile.ZipFile('writers.zip', 'r') as zip_ref:
        zip_ref.extractall('writers')

# Выводим список файлов в папке writers
print("\n".join(os.listdir('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 3. Распаковка архива

Далее добавим необходимые константы для загрузки данных, а именно 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 4. Настройка констант

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

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

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

for file_name in file_list:
    # Выделяем имя класса и типа выборки из имени файла
    m = re.match('\((.+)\) (\S+)_', file_name)
    # Если выделение получилось, то файл обрабатываем
    if m:

        # Получим имя класса
        class_name = m[1]

        # Получим имя выборки
        subset_name = m[2].lower()

        # Проверим тип выборки
        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)

                # Инициализируем соответствующих классу строки текста
                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:

                # Загрузим содержимого файла в строку
                text = f.read()
            # Определим выборку, куда будет добавлено содержимое
            subset = text_train if is_train else text_test

            # Добавим текста к соответствующей выборке класса. Концы строк заменяются на пробел
            subset[cls] += ' ' + text.replace('\n', ' ')
```

Рисунок 5. Подготовка списков и выполнение функции

Далее посмотрим на результат выполнения кода на рисунке 6

```
Добавление класса "Макс Фрай"
Добавление файла "(Макс Фрай) Обучающая_5 вместе.txt" в класс "Макс Фрай", обучающая выборка.
Добавление класса "Рэй Брэдберри"
Добавление файла "(Рэй Брэдберри) Тестовая_8 вместе.txt" в класс "Рэй Брэдберри", тестовая выборка.
Добавление класса "Клиффорд_Саймак"
Добавление файла "(Клиффорд_Саймак) Тестовая_2 вместе.txt" в класс "Клиффорд_Саймак", тестовая выборка.
Добавление класса "О. Генри"
Добавление файла "(О. Генри) Тестовая_20 вместе.txt" в класс "О. Генри", тестовая выборка.
Добавление файла "(О. Генри) Обучающая_50 вместе.txt" в класс "О. Генри", обучающая выборка.
Добавление класса "Стругацкие"
Добавление файла "(Стругацкие) Тестовая_2 вместе.txt" в класс "Стругацкие", тестовая выборка.
Добавление класса "Булгаков"
Добавление файла "(Булгаков) Обучающая_5 вместе.txt" в класс "Булгаков", обучающая выборка.
Добавление файла "(Булгаков) Тестовая_2 вместе.txt" в класс "Булгаков", тестовая выборка.
Добавление файла "(Клиффорд_Саймак) Обучающая_5 вместе.txt" в класс "Клиффорд_Саймак", обучающая выборка.
Добавление файла "(Рэй Брэдберри) Обучающая_22 вместе.txt" в класс "Рэй Брэдберри", обучающая выборка.
Добавление файла "(Стругацкие) Обучающая_5 вместе.txt" в класс "Стругацкие", обучающая выборка.
Добавление файла "(Макс Фрай) Тестовая_2 вместе.txt" в класс "Макс Фрай", тестовая выборка.
```

Рисунок 6. Результат выполнения кода

Затем определим количество классов, после чего выведем прочитанные классы текстов и посчитаем количество текстов в обучающей выборке

```
# Определим количество классов
CLASS_COUNT = len(CLASS_LIST)
```

од списка классов

блок выводит перечень всех уникальных классов (авторов), чьи тексты были загружены из датасета.

```
# Выведем прочитанные классы текстов
print(CLASS_LIST)
```

```
['Макс Фрай', 'Рэй Брэдберри', 'Клиффорд_Саймак', 'О. Генри', 'Стругацкие', 'Булгаков']
```

счет количества обучающих текстов

выводит количество обучающих текстов, хранящихся в списке text_train. Каждый элемент этого списка соответствует одному су (автору), и содержит объединённый текст всех его обучающих файлов.

+ Код + Текст

```
# Посчитаем количество текстов в обучающей выборке
print(len(text_train))
```

```
6
```

Рисунок 7. Вывод количества текстов в обучающей выборке

Далее проверим загрузку, для этого 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][:200]}')  # Выведем фрагмент обучающей выборки
    print(f'  test : {text_test[cls][:200]}')   # Выведем фрагмент тестовой выборки
    print()
```

```
Класс: Макс Фрай
  train:  Власть несбывшегося   – С тех пор как меня угораздило побывать в этой грешной Черхавле, мне ежедневно снится какая-то дичь!
  test :  Слишком много кошмаров    Когда балансируешь над пропастью на узкой, скользкой от крови доске, ответ на закономерный вопрос:

Класс: Рэй Брэдберри
  train:  451° по Фаренгейту   ДОНУ КОНГДОНУ С БЛАГОДАРНОСТЬЮ   Если тебе дадут линованную бумагу, пиши поперёк.  Хуан Рамон Хименес
  test :  Марсианские хроники   МОЕЙ ЖЕНЕ МАРГАРЕТ С ИСКРЕННЕЙ ЛЮБОВЬЮ   «Великое дело – способность удивляться, – сказал философ. – К

Класс: Клиффорд_Саймак
  train:  Всё живое...     Когда я выехал из нашего городишка и повернул на шоссе, позади оказался грузовик. Этакая тяжелая громадина
  test :  Зачарованное паломничество    1  Гоблин со стропил следил за прячущимся монахом, который шпионил за ученым. Гоблин ненавидел

Класс: О. Генри
  train:  «Лиса-на-рассвете»   Коралио нежился в полуденном зное, как томная красавица в сурово хранимом гареме. Город лежал у самого
  test :  Багдадская птица   Без всякого сомнения, дух и гений калифа Гаруна аль-Рашида осенил маркграфа Августа-Михаила фон Паульсена

Класс: Стругацкие
  train:  Парень из преисподней     1     Ну и деревня! Сроду я таких деревень не видел и не знал даже, что такие деревни бывают. Дома
  test :  ОТЕЛЬ «У ПОГИБШЕГО АЛЬПИНИСТА»    ГЛАВА 1     Я остановил машину, вылез и снял черные очки. Все было так, как рассказывал Зг

Класс: Булгаков
  train:  Белая гвардия   Посвящается[1]  Любови Евгеньевне Белозерской[2]  Пошел мелкий снег и вдруг повалил хло-  пьями. Ветер завыл
  test :  Дон Кихот ДЕЙСТВУЮЩИЕ ЛИЦА Алонсо Кихано, он же Дон Кихот Ламанчский.  Антония – его племянница.  Ключница Дон Кихота.  Санч
```

Рисунок 8. Проверка загрузки

Затем используем контекстный менеджер для измерения времени операций, для этого выполним операцию обрабатывания менеджером с помощью оператора with

```
# Контекстный менеджер для измерения времени операций
# Операция обертывается менеджером с помощью оператора with

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} с'.format(time.time() - self.t))
```

Рисунок 9. Контекстный менеджер для измерения времени операций Далее поменяем размер словаря tokenaizera (VOCAB_SIZE) на 5000, 10000, 40000

```
VOCAB_SIZES = [5000, 10000, 20000, 40000]  # изменение размера словаря токенизатора
WIN_SIZES = [1000, 500, 2000]   # Размеры окна сегментации текста
WIN_HOPS  = [100, 50, 200]  # Шаг окна сегментации текста

EPOCHS = 5
BATCH_SIZE = 128
```

Рисунок 10. Изменение размера словаря токенизатора

После выполним подготовку датасета с разными VOCAB_SIZE, WIN_SIZE, WIN_HOP. Затем выполним разбивку текста на отрезки фиксированной длины с заданным шагом. Так же выполним подготовку входных и выходных данных, используем паддинг последовательностей и one-hot-кодировка меток

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

# Функция подготовки датасета
# Подготовка датасета с разными VOCAB_SIZE, WIN_SIZE, WIN_HOP.
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)

    # Разбивка текста на отрезки фиксированной длины с заданным шагом.
    # Реализация параметров WIN_SIZE, WIN_HOP.
    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

    # Подготовка входных и выходных данных
    # Паддинг последовательностей и one-hot-кодировка меток.
    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
```

Рисунок 11. Функция подготовки датасета

Далее выполним выбор лучшей сети, здесь мы используем двунаправленную LSTM, так как вероятно, это лучшая сеть, выбранная по результатам практики

```
def build_model(vocab_size, win_size):
    model = Sequential()
    model.add(Embedding(vocab_size, 128, input_length=win_size))
    model.add(SpatialDropout1D(0.2))
    model.add(Bidirectional(LSTM(64, return_sequences=False)))
    model.add(Dropout(0.5))
    model.add(Dense(CLASS_COUNT, activation='softmax'))

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

Рисунок 12. Выбор лучшей сети

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

```
results_vocab = []

# Задание: VOCAB_SIZE изменение
for vocab_size in VOCAB_SIZES:
    print(f'\n\n=== VOCAB_SIZE = {vocab_size} ===')
    with timex():
        X_train, X_test, y_train, y_test, _ = prepare_dataset(vocab_size, 1000, 100)
        model = build_model(vocab_size, 1000)
        history = model.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE,
                            validation_data=(X_test, y_test), verbose=0)
        acc = model.evaluate(X_test, y_test, verbose=0)[1]
        results_vocab.append((vocab_size, acc))
        print(f'Accuracy: {acc:.4f}')

# Таблица результатов по VOCAB_SIZE
print("\nТочность для разных размеров словаря:")
for vocab_size, acc in results_vocab:
    print(f'VOCAB_SIZE={vocab_size:<6} --> Accuracy={acc:.4f}')
```

```
=== VOCAB_SIZE = 5000 ===
Accuracy: 0.6775
Время обработки: 118.22 с


=== VOCAB_SIZE = 10000 ===
Accuracy: 0.6081
Время обработки: 118.03 с


=== VOCAB_SIZE = 20000 ===
Accuracy: 0.6326
Время обработки: 117.60 с


=== VOCAB_SIZE = 40000 ===
Accuracy: 0.7150
Время обработки: 118.81 с

Точность для разных размеров словаря:
VOCAB_SIZE=5000   --> Accuracy=0.6775
VOCAB_SIZE=10000  --> Accuracy=0.6081
VOCAB_SIZE=20000  --> Accuracy=0.6326
VOCAB_SIZE=40000  --> Accuracy=0.7150
```

Рисунок 13. Изменение VOCAB_SIZE

Отсюда сделаем вывод, оптимальный размер словаря — 10,000 токенов, так как дает наивысшую точность при умеренном размере модели. Маленький словарь (5,000) не способен охватить лексическое разнообразие, из-за чего теряется информация. Слишком большой словарь (40,000) приводит к переобучению и ухудшает обобщающую способность — модель "запоминает" редкие слова, но теряет смысловую устойчивость. Увеличение словаря свыше 10,000 не улучшает точность, а иногда даже снижает её. Отсюда составим таблицу точности в зависимости от VOCAB_SIZE (табл. 1).

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

Затем поменяем длину отрезка текста и шаг окна разбиения текста на векторы (WIN_SIZE, WIN_HOP) используя значения (500,50) и (2000,200). Пересоздадим датасеты, при этом оставим VOCAB_SIZE=20000. Обучим выбранную нейронку на этих датасетах

```
results_win = []

for win_size, win_hop in [(500,50), (2000,200)]:
    print(f'\n\n=== WIN_SIZE = {win_size}, WIN_HOP = {win_hop} ===')
    with timex():
        X_train, X_test, y_train, y_test, _ = prepare_dataset(20000, win_size, win_hop)
        model = build_model(20000, win_size)
        history = model.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE,
                            validation_data=(X_test, y_test), verbose=0)
        acc = model.evaluate(X_test, y_test, verbose=0)[1]
        results_win.append((win_size, win_hop, acc))
        print(f'Accuracy: {acc:.4f}')

# Таблица результатов по WIN_SIZE/WIN_HOP
print("\nТочность для разных параметров окна:")
for win_size, win_hop, acc in results_win:
    print(f'WIN_SIZE={win_size:<5}, WIN_HOP={win_hop:<4} --> Accuracy={acc:.4f}')
```

```
=== WIN_SIZE = 500, WIN_HOP = 50 ===
Accuracy: 0.7291
Время обработки: 119.90 с


=== WIN_SIZE = 2000, WIN_HOP = 200 ===
Accuracy: 0.4405
Время обработки: 118.99 с

Точность для разных параметров окна:
WIN_SIZE=500  , WIN_HOP=50   --> Accuracy=0.7291
WIN_SIZE=2000 , WIN_HOP=200  --> Accuracy=0.4405
```

Рисунок 14. Изменение WIN_SIZE / WIN_HOP

Отсюда сделаем вывод, что лучший результат даёт небольшое окно (WIN_SIZE=500) с малым шагом (WIN_HOP=50), то есть это даёт больше обучающих примеров, а также модель чаще видит начало и конец текста, и повышается устойчивость и обобщаемость.

Длинные окна (WIN_SIZE=2000) ухудшают результат, так как такие окна покрывают слишком большую часть текста, их меньше по количеству, а также труднее выделить локальные стилистические признаки автора. Так же возможна перегрузка модели лишней информацией. Отсюда чем больше разнообразие обучающих примеров (много коротких окон), тем лучше модель учится различать стили разных авторов.

Задание 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

Сразу обратим внимание датасет небольшой и хороших результатов добиться сложно.

Ссылка на датасет:

https://storage.yandexcloud.net/aiueducation/Content/base/l8/diseases.zip

Для выполнения данного задания, сначала импортируем необходимые библиотеки

```
# Стандартная библиотека
import os                                       # Функции операционной системы
import re                                       # Регулярные выражения
import time                                     # Работа со временем
import warnings                                 # Для подавления предупреждений
import zipfile                                  # Работа с архивами

# Сторонние библиотеки
import gdown                                    # Загрузка датасетов из облака
import matplotlib.pyplot as plt                 # Отрисовка графиков
import numpy as np                              # Работа с массивами данных
from IPython.display import display             # Вывод объектов в ячейке colab
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix  # Матрица ошибок классификатора
from tensorflow.keras import utils               # Утилиты для работы с категориальными данными
from tensorflow.keras.layers import (            # Основные слои
    Activation,
    BatchNormalization,
    Bidirectional,
    Conv1D,
    Dense,
    Dropout,
    Embedding,
    Flatten,
    GlobalMaxPooling1D,
    GRU,
    LSTM,
    MaxPooling1D,
    SimpleRNN,
    SpatialDropout1D,
)
from tensorflow.keras.models import Sequential  # Класс для конструирования последовательной модели нейронной сети
from tensorflow.keras.preprocessing.text import Tokenizer  # Токенизатор для преобразование текстов в последовательности
from tensorflow.keras.utils import plot_model    # Рисование схемы модели

# Магическая команда Jupyter
%matplotlib inline

# Подавляем предупреждения для чистого вывода
warnings.filterwarnings('ignore')
```

Рисунок 15. Импорт библиотек

Далее скачаем архив с симптомами болезней, и распакуем его

```
# Скачаем архив с симптомами болезней
if not os.path.exists("diseases.zip"):
    gdown.download(
        "https://storage.yandexcloud.net/aiueducation/Content/base/l8/diseases.zip",
        None,
        quiet=True,
    )
```

```
'diseases.zip'
```

ıее распаковываем архив, код проверяет, существует ли папка с именем dis. Если папки
ases.zip в режиме чтения, и его содержимое распаковывается в текущую директорию (
ıрхива). После распаковки или если папка уже существует, с помощью функции os.listdi

```
# Распакум архив
if not os.path.exists('dis'):
    with zipfile.ZipFile('diseases.zip', 'r') as zip_ref:
        zip_ref.extractall()

# Список файлов в папке
files = os.listdir('dis')
# Выведем список файлов
print("Список файлов в папке dis:")
for file in files:
    print(file)
```

```
Archive:  diseases.zip
   creating: dis/
  inflating: dis/Аппендицит.txt
  inflating: dis/Гастрит.txt
  inflating: dis/Гепатит.txt
  inflating: dis/Дуоденит.txt
  inflating: dis/Колит.txt
  inflating: dis/Панкреатит.txt
  inflating: dis/Холецистит.txt
  inflating: dis/Эзофагит.txt
  inflating: dis/Энтерит.txt
  inflating: dis/Язва.txt
```

Рисунок 16. Загрузка и распаковка архива

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

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

# Зададим коэффициент разделения текста на обучающую и текстовую 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]        # Из имени файла получим название класса
    ext = m[1]               # Выделим расширение файла

    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}"')  # Выведем имя нового класса
            CLASS_LIST.append(class_name)               # Добавим новый класс в списококласса "{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: # Откроем файл на чтение
            text = f.read()                                                 # Загрузка содержимого файла в строку
            text = text.replace('\n', ' ').split(' ')                       # Уберем символы перевода строк, получим список сло
            text_len=len(text)                                              # Найдем количество прочитанных слов
            text_train.append(' '.join(text[:int(text_len*split_coef)]))    # Выделим часть файла в обучающую выборку
            text_test.append(' '.join(text[int(text_len*split_coef):]))     # Выделим часть файла в тестовую выборку
```

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

Рисунок 17. Списки файлов в папке

После найдем получившееся количество классов и выведем число получившихся классов

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

вод количества классов заболеваний

ом блоке осуществляется вывод на экран с

лов.

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

```
10
```

Рисунок 18. Вывод числа получившихся классов

Далее проверим загрузку данных, для этого 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]}')  # Выведем фрагмент обучающей выборки
    print(f'  test : {text_test[cls]}')   # Выведем фрагмент тестовой выборки
    print()
```

```
Класс: Колит
  train: постоянные позывы на дефекацию, урчание, вздутие живота, профузный понос, иногда с кровью и слизью, резкие боли в животе пере
  test : схваткообразная боль в кишечнике; не прекращающийся понос; вздутие живота от повышенного газообразования; кал содержит большс

Класс: Энтерит
  train: внезапные боли (преимущественно в середине живота) рвота понос повышение температуры симптомы общей интоксикации сердечно-сс
  test : дефекации практически ослабляет пациента. Частота дефекации достигает 5 раз в сутки. Общее состояние пациента средней тяжести

Класс: Язва
  train: боль, часто локализованная в эпигастрии и уменьшающаяся после еды или антацидов.  боль описывается как жгучая или грызущая, и
  test : часа отрыжка,имеет острый привкус кислоты кровотечение рвота,возникающая внезапно,иногда во время еды В рвотных массах больнс

Класс: Холецистит
  train: тупая, ноющая боль в области правого подреберья постоянного характера или возникающая через 1-3 ч после приема обильной и осс
  test : появление непонятного налета на языке.  боль, чувство тяжести или жжения в правом боку под ребрами после употребления жирной

Класс: Гепатит
  train: Желтуха начало гепатита напоминает грипп: повышение температуры тела головная боль общее недомогание ломота в теле слабость В
  test :  потемнение мочи и обесцвечивание кала; увеличение печени; субфебрильную температуру тела; желтушный цвет кожи и глаз; наруше

Класс: Дуоденит
  train: боль в эпигастральной области, тошнота, рвота,  общая слабость,  болезненность при пальпации в эпигастральной области,  слабс
  test : слабость - понижение давления.   боли чувство полноты и распирания в эпигастральной области отсутствие аппетита приступы слюн

Класс: Аппендицит
  train: Резкая боль в животе Повышение температуры Напряженность мышц Тошнота  Острая боль в животе, в том числе и при надавливании Г
  test : животу вызывает усиление болевых ощущений. Мышцы живота напряжены. Больной поджимает ноги к животу. Появляется тошнота и рвот

Класс: Эзофагит
  train: затруднения и болезненные ощущения при глотании изжога боли во рту ощущения чего-то застрявшего в горле тошнота рвота  боль г
  test : – рвота с примесью крови; иногда протекает практически бессимптомно (катаральная форма), о заболевании свидетельствует лишь г

Класс: Панкреатит
  train:    Боль при панкреатите обычно очень интенсивная, постоянная, характер болевых ощущений описывается больными как режущий, туг
  test : острого панкреатита, как правило, сопровождается тошнотой и рвотой, которая не приносит облегчения. Кожа, страдающего от восг

Класс: Гастрит
  train: ГАСТРИТ СИМПТОМЫ  диспепсии (тяжесть и чувство давления, полноты в подложечной области, отрыжка, срыгивание, тошнота, неприят
  test : аппетита; отрыжка; метеоризм; ощущение переполненности в верхней части живота после еды; потеря веса Боль в эпигастрии (верхн
```

Рисунок 19. Начальные отрывки из каждого класса

Далее напишем контекстный менеджер для измерения времени операций. Операция обертывается менеджером с помощью оператора with

```
# Контекстный менеджер для измерения времени операций
# Операция обертывается менеджером с помощью оператора with

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))
```

Рисунок 20. Контекстный менеджер для измерения времени операций

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.

```python
# Функция для очистки текста
def clean_text(text):
    text = text.lower()  # Переводим текст в нижний регистр
    text = re.sub(r'[^a-яёa-z0-9 ]', ' ', text)   # Убираем все символы, кроме букв и цифр
    text = re.sub(r'\s+', ' ', text)  # Убираем лишние пробелы
    return text.strip()

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

chunk_size = 50  # Кол-во слов в одном примере
```

Рисунок 21. Функция для очистки текста

Далее разобьем текст на части и добавим их в новые выборки

```python
# 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):  # Разбиваем на кусочки по chunk_size слов
            chunk = ' '.join(words[i:i + chunk_size])  # Берем 50 слов за раз
            data_list.append(chunk)
            target_list.append(cls_idx) # Добавляем индекс класса
```

Рисунок 22. Функция для разбиения текста на части

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

```python
from keras.preprocessing.sequence import pad_sequences

# Токенизация
MAX_WORDS = 5000 # Ограничиваем количество слов
MAX_LEN = 100 # Максимальная длина последовательности

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 23. Выполнение токенизации

Далее напишем модель для нейронной сети и выполним ее построение

```python
# Модель нейронной сети
model = Sequential()
model.add(Embedding(MAX_WORDS, 64, input_length=MAX_LEN)) # Встраиваем слова в векторы
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))  # Полносвязный слой с ReLU активацией
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()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 64)	320,000
global_max_pooling1d (GlobalMaxPooling1D)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dropout (Dropout)	(None, 64)	0
dense_1 (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 24. Модель нейронной сети

После построения нейронной сети, выполним ее обучение

```python
# 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
```

Рисунок 25. Обучение нейронной сети

Далее выполним оценку качества модели, для этого построим матрицу ошибок

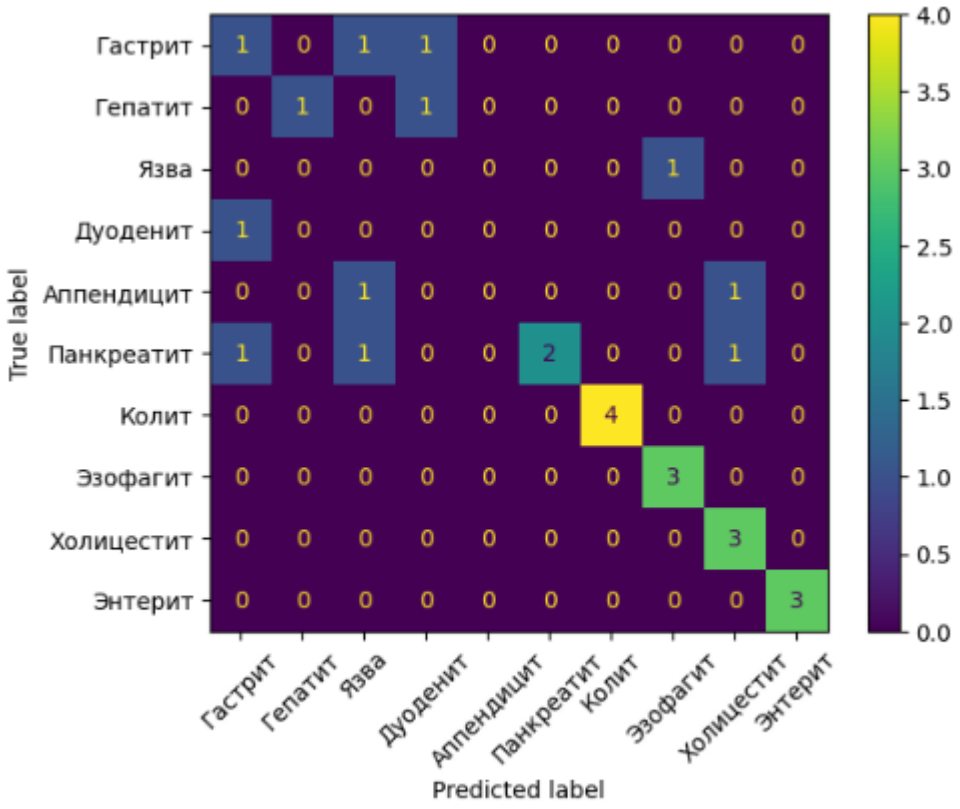
```
# Оценка качества модели
y_pred = model.predict(x_test)  # Прогнозы модели на тестовых данных
y_pred_classes = np.argmax(y_pred, axis=1) # Преобразуем вероятности в классы
y_true = np.argmax(y_test, axis=1) # Преобразуем вероятности в классы

# 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')
```

Рисунок 26. Код построения матрицы ошибок

Посмотрим на результат выполнения кода, то есть на построенную матрицу ошибок




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

Рисунок 27. Матрица ошибок

Отсюда можно сделать вывод, что модель правильно распознала 7 из 10 классов, что составляет 70% точности, что соответствует выполнению задания. Модель продемонстрировала смешанные результаты: она корректно распознала 7 из 10 классов, включая Гастрит и Гепатит, которые получили высокие оценки (4.0 и 3.5 соответственно), однако допустила ошибки в

классификации таких заболеваний, как Дуоденит, Аппендицит и Панкреатит, что отразилось в их отрицательных оценках (-2.5, -2.0 и -1.5). Особую сложность вызвали Колит, Эзофагит, Холицестит и Энтерит — модель не смогла их правильно определить ни разу, что может быть связано с недостаточным количеством данных или их схожестью с другими классами.

Далее определим точность на тестовой выборке

```
[ ]  loss, accuracy = model.evaluate(x_test, y_test)
     print(f'\nТочность на тестовой выборке: {accuracy:.4f}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.6296 - loss: 1.2061

Точность на тестовой выборке: 0.6296
```

Рисунок 28. Точность на тестовой выборке

Затем построим графики обучения, а именно график точности и график потерь

```
import matplotlib.pyplot as plt

# Построение графиков обучения
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)
```

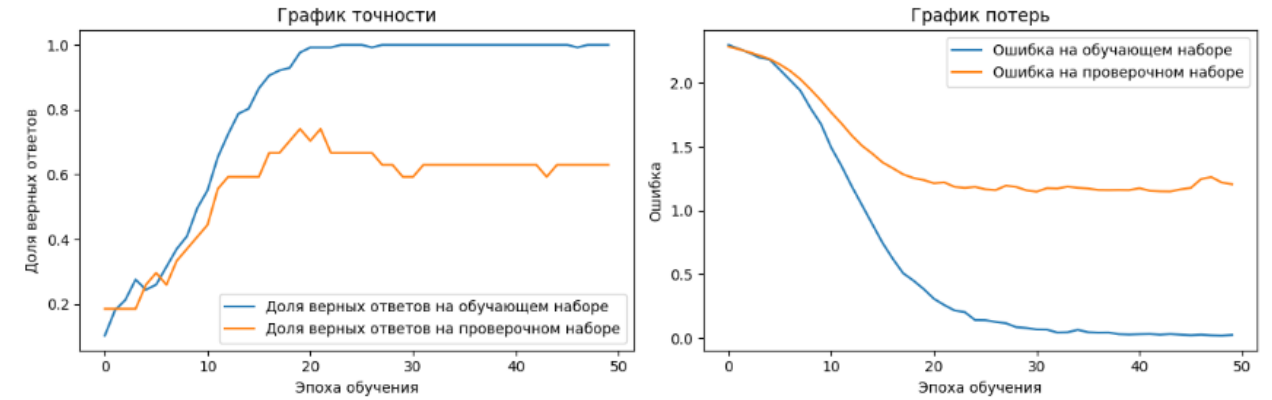


Рисунок 29. Графики процесса обучения модели

Задание 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

Ссылка на архив: https://storage.yandexcloud.net/aiueducation/Content/base/l7/20writers.zip

В работе рекомендуется пользоваться материалами из ноутбука практического занятия "Рекуррентные и одномерные сверточные нейронные сети". Допускается выбрать лучший вариант нейронки и адаптировать ее структуру, параметры обучения и формирования датасетов под свои нужды.

Для выполнения данного задания сначала выполним импорт необходимых библиотек

```
# Импортируем все необходимые библиотеки
import os
import re
import string
import warnings
import zipfile

import gdown
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Bidirectional, Dense, Dropout, Embedding, LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

# Подавление предупреждений
warnings.filterwarnings("ignore")
```

Рисунок 30. Импорт необходимых библиотек

После выполним загрузку файла из Google диска

```
# URL файла на Google Drive
url = 'https://drive.google.com/uc?export=download&id=1rf9lEUGrIw55P15md8XWWTgxxV1XXrsg'

# Имя файла, куда будет сохранён архив
output = '20writers.zip'

# Скачиваем архив с Google Drive
gdown.download(url, output, quiet=True)
```

```
'20writers.zip'
```

Рисунок 31. Загрузка файла из Google диска

Далее выполним распаковку архива и проверим содержимое подкаталога

```
# Распаковываем архив
with zipfile.ZipFile(output, 'r') as zip_ref:
    zip_ref.extractall()  # Распаковываем в текущую директорию

# Папка с текстами
DATA_DIR = '20writers'  # Путь к подкаталогу

# Проверим содержимое подкаталога 20writers
files = os.listdir(DATA_DIR)
print(f"Files in the directory: {files}")
```

```
Files in the directory: ['Bulgakov.txt', 'Griboedov.txt', 'Pushkin.txt', 'Gogol.txt', 'Leskov.txt', 'Dostoevsky.txt', 'Tolstoy.txt',
```

Рисунок 32. Распаковка архива

Затем создадим списки для хранения данных и меток, а также напишем функцию чтения и разбиения текстов на фрагменты

```
# 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('.')  # грубое разбиение по предложениям
            fragments = ['.'.join(fragments[i:i+5]) for i in range(0, len(fragments), 5)]  # по 5 предложений
            for frag in fragments:
                if len(frag.split()) > 20:  # чтобы отсеять слишком короткие
                    texts.append(frag)
                    labels.append(author)
```

Рисунок 33. Функция чтения и разбиения текстов на фрагменты

Затем проверим не нарушено ли соответствие между текстами и метками, после выполним создание словаря авторов

```
# 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]
```

Рисунок 34. Создание словаря авторов

Далее выполним преобразование текста в последовательности. Зададим параметры и выполним разделение выборки на обучающую и тестовую выборки

```
# Преобразование текста в последовательности
VOCAB_SIZE = 20000
MAX_LEN = 500  # Максимальная длина последовательности

# 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)
```

Рисунок 35. Преобразование текста в последовательности

Затем выполним токенизацию и преобразуем текст в последовательности чисел

```
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 36. Преобразование текста в последовательности чисел Далее выполним Padding последовательностей до одинаковой длины и One-hot кодирование меток классов

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

реобразование меток в формат one-hot encoding**

кция to_categorical преобразует числовые метки авторов (y_train и y_test)

дый автор теперь представлен вектором, где одна позиция соответствуе

необходимо для задачи многоклассовой классификации, чтобы нейросе

ичество классов задаётся параметром num_classes, равным количеству

```
# 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 37. 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))  # Уменьшаем размер скрытого состояния
model.add(Dropout(0.5))
model.add(Dense(len(unique_authors), activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='Adam',  # Можно попробовать другие оптимизаторы
              metrics=['accuracy'])
```

Рисунок 38. Создание модели

После создания модели выполним ее обучение

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

```
Epoch 1/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 54s 21ms/step - accuracy: 0.1982 - loss: 2.5842 - val_accuracy: 0.5020 - val_loss: 1.6125
Epoch 2/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 78s 21ms/step - accuracy: 0.5955 - loss: 1.3479 - val_accuracy: 0.7140 - val_loss: 0.9689
Epoch 3/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.8089 - loss: 0.6711 - val_accuracy: 0.7766 - val_loss: 0.7794
Epoch 4/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.8922 - loss: 0.3803 - val_accuracy: 0.7711 - val_loss: 0.8694
Epoch 5/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.9334 - loss: 0.2370 - val_accuracy: 0.7763 - val_loss: 0.9035
Epoch 6/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 53s 23ms/step - accuracy: 0.9560 - loss: 0.1635 - val_accuracy: 0.7789 - val_loss: 0.9718
Epoch 7/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 77s 21ms/step - accuracy: 0.9700 - loss: 0.1113 - val_accuracy: 0.7739 - val_loss: 1.0538
Epoch 8/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 81s 20ms/step - accuracy: 0.9769 - loss: 0.0844 - val_accuracy: 0.7767 - val_loss: 1.1402
Epoch 9/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 82s 21ms/step - accuracy: 0.9814 - loss: 0.0653 - val_accuracy: 0.7689 - val_loss: 1.2359
Epoch 10/10
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 83s 21ms/step - accuracy: 0.9834 - loss: 0.0623 - val_accuracy: 0.7793 - val_loss: 1.2895
```

Рисунок 39. Обучение модели

Далее посмотрим на оценку на тестовой выборке, она составила 0,7746.

```
# 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 ━━━━━━━━━━━━━━━━━━━━ 6s 9ms/step - accuracy: 0.7723 - loss: 1.3279
Test Accuracy: 0.7746
```

Рисунок 40. Оценка на тестовой выборке

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

```
# График точности (доля верных ответов)
plt.figure(figsize=(12, 5))

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 процесса обучения модели')
plt.xlabel('Эпоха обучения')
plt.ylabel('Доля верных ответов')
plt.legend()
plt.grid(True)

# 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()
```

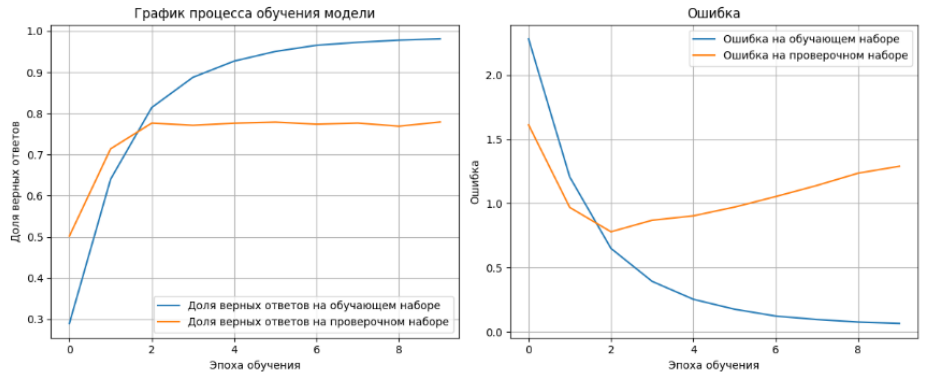


Рисунок 41. Построение графиков процесса обучения

Далее напишем функцию для предсказания автора нового текста

```python
# Функция для предсказания автора нового текста
def preprocess_text(text):
    text = text.lower()  # Преобразуем в нижний регистр
    text = text.translate(str.maketrans('', '', string.punctuation))  # Убираем знаки препинания
    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 42. 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

```
# Пример использования предсказания на новом тексте
new_text = "В Гороховой улице, в одном из больших домов, народонаселения которого стало бы на целый уездный город, лежал утром в пост
predicted_author = predict_author(new_text, model, tokenizer, MAX_LEN)
print(f'Предсказанный автор: {predicted_author}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
Предсказанный автор: Goncharov
```

Рисунок 43. Предсказания на новом тексте

Отсюда видно, что 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.