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 №4

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: Обработка текстов с помощью нейронных сетей.

Цель: изучить особенности построения архитектур нейронных сетей для обработки текста и особенности обучения таких НС.

Ссылка: https://github.com/Artorias1469/NN_4.git

Ход работы:
Выполнение индивидуальных заданий:

Задание 1.

Условие: в домашнем задании Lite предлагается поработать подробнее с параметрами словаря и формированием гиперпараметров нейронной сети. Необходимо создать 9 нейросетей с различными гиперпараметрами.

Для этого необходимо:

1. Воссоздать ноутбук, аналогичный ноутбуку практической части №1, загрузив при этом необходимую нам базу (код уже доступен в ноутбуке).
2. Задать в ноутбуке следующие параметры для размера словаря, ширины окна и шага:

- Размер словаря - от 10000 до 20000 (выбрать меньшее значение диапазона, если будет перегрузка ОЗУ и перезапуск подключения к Colaboratory)
- Ширина окна - от 1000 до 2000
- Шаг - от 100 до 500 (на обучение лучше влияет наименьший шаг, но это может перегрузить ОЗУ).

3. Создать архитектуру сети и задать гиперпараметры. Можно воспользоваться шаблоном:

- Добавить модель прямого распространения Sequential()
- Добавить один или несколько полносвязных (Dense) слоёв
- Добавить слои Dropout() и BatchNormalization()
- Добавить выходной полносвязный слой с количеством нейронов, соответствующим количеству классов (число писателей).

Выполнение данного задания начнем с загрузки необходимых библиотек для работы

```python
# Стандартная библиотека
import os                                   # Работа с файловой системой
import pickle                               # Сериализация и десериализация Python объектов (сохранение и загрузка)
import re                                   # Работа с регулярными выражениями для обработки текста
import time                                 # Функции для измерения времени и задержек
import zipfile                              # Работа с zip-архивами

# Сторонние библиотеки
import gdown                                # Для загрузки файлов из облачных хранилищ
import matplotlib.pyplot as plt             # Для построения графиков и визуализации данных
import numpy as np                          # Работа с массивами и числовыми вычислениями
import pandas as pd                         # Работа с табличными данными
from sklearn.metrics import (
    ConfusionMatrixDisplay,                 # Визуализация матрицы ошибок классификации
    confusion_matrix                        # Вычисление матрицы ошибок классификации
)

# TensorFlow / Keras
from tensorflow.keras import utils          # Утилиты Keras (например, для one-hot кодирования)
from tensorflow.keras.layers import (
    Activation,                             # Функция активации
    BatchNormalization,                     # Нормализация по батчам
    Dense,                                  # Полносвязный слой
    Dropout,                                # Слой регуляризации Dropout для предотвращения переобучения
    Embedding,                              # Слой эмбеддингов для представления слов
    Flatten,                                # Слой выравнивания многомерного тензора в одномерный вектор
    Input,                                  # Слой входных данных
    SpatialDropout1D                        # Специализированный Dropout для 1D данных (например, последовательностей)
)
from tensorflow.keras.models import Sequential  # Модель последовательного стэка слоев Keras
from tensorflow.keras.preprocessing.text import Tokenizer  # Токенизация текста
```

Рисунок 1. Импорт библиотек

Далее выполним загрузку датасета и распаковку архива в папку writers

```python
if not os.path.exists("writers.zip"):
    # Загрузка архива с датасетом
    gdown.download(
        "https://storage.yandexcloud.net/aiueducation/Content/base/l7/writers.zip",
        None,
        quiet=True,
    )

# Распаковка архива в папку writers
with zipfile.ZipFile("writers.zip", "r") as zip_ref:
    zip_ref.extractall("writers")

# Просмотр содержимого папки
print("\n".join(os.listdir("writers")))
```

```
'(Булгаков) Обучающая_5 вместе.txt'
'(Булгаков) Тестовая_2 вместе.txt'
'(Клиффорд_Саймак) Обучающая_5 вместе.txt'
'(Клиффорд_Саймак) Тестовая_2 вместе.txt'
'(Макс Фрай) Обучающая_5 вместе.txt'
'(Макс Фрай) Тестовая_2 вместе.txt'
'(О. Генри) Обучающая_50 вместе.txt'
'(О. Генри) Тестовая_20 вместе.txt'
'(Рэй Брэдберри) Обучающая_22 вместе.txt'
'(Рэй Брэдберри) Тестовая_8 вместе.txt'
'(Стругацкие) Обучающая_5 вместе.txt'
'(Стругацкие) Тестовая_2 вместе.txt'
```

Рисунок 2. Загрузка датасета

Затем выполним настройку констант для загрузки данных .

```python
# Настройка констант для загрузки данных
FILE_DIR = "writers"
SIG_TRAIN = "обучающая"
SIG_TEST = "тестовая"
```

Рисунок 3. Настройка констант

Далее определим количество классов, для этого для начала преобразуем все тексты в строку и объединим для каждого класса и выборки .

```python
# Загрузка датасета. Добавляются имена классов и соответствующие тексты.
# Все тексты преобразуются в строку и объединяются для каждого класса и выборки
CLASS_LIST = []
text_train = []
text_test = []

for file_name in os.listdir(FILE_DIR):
    # Выделение имени класса и типа выборки из имени файла
    m = re.match(r"\((.+)\) (\S+)_", file_name)
    # Если выделение получилось, то файл обрабатывается
    if m:
        class_name = m[1]
        subset_name = m[2].lower()
        # Проверка типа выборки в имени файла
        is_train = SIG_TRAIN in subset_name
        is_test = SIG_TEST in subset_name

        # Если тип выборки обучающая либо тестовая - файл обрабатывается
        if is_train or is_test:
            # Добавление нового класса, если его еще нет в списке
            if class_name not in CLASS_LIST:
                print(f'Добавление класса "{class_name}"')
                CLASS_LIST.append(class_name)
                # Инициализация соответствующих классу строк текста
                text_train.append("")
                text_test.append("")

            # Поиск индекса 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.'
            )
            with open(f"{FILE_DIR}/{file_name}", "r") as f:
                # Загрузка содержимого файла в строку
                text = f.read()
            # Определение выборки, куда будет добавлено содержимое
            subset = text_train if is_train else text_test
            # Добавление текста к соответствующей выборке класса. Концы строк заменяются на пробел
            subset[cls] += " " + text.replace("\n", " ")

# Определение количества классов
CLASS_COUNT = len(CLASS_LIST)
```

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

Рисунок 4. Определение количества классов

Затем выведем прочитанные классы тестов, количество текстов в обучающей выборке и количество символов одного из текстов обучающей выборки

```python
# Прочитанные классы текстов
print(CLASS_LIST)

# Количество текстов в обучающей выборке
print(len(text_train))

# Количество символов в одном из текстов обучающей выборки
print(len(text_train[2]))
```

```
['О. Генри', 'Макс Фрай', 'Клиффорд_Саймак', 'Булгаков', 'Стругацкие', 'Рэй Брэдберри']
6
1609508
```

Рисунок 5. Вывод количества текстов

Далее выполним проверку загрузки, то есть вывод начальных отрывков из каждого класса

```
# Проверка загрузки: вывод начальных отрывков из каждого класса
for cls in range(CLASS_COUNT):
    print(f"Класс: {CLASS_LIST[cls]}")
    print(f"  train: {text_train[cls][:200]}")
    print(f"  test : {text_test[cls][:200]}")
    print()
```

```
Класс: О. Генри
  train:  «Лиса-на-рассвете»   Коралио нежился в полуденном зное, как томная красавица в сурово хранимом гареме. Город лежал у самого моря на полоске наносной земли. Он казался брильян
  test :  Багдадская птица   Без всякого сомнения, дух и гений калифа Гаруна аль-Рашида осенил маркграфа Августа-Михаила фон Паульсена Квигга.  Ресторан Квигга находится на Четвертой

Класс: Макс Фрай
  train:  Власть несбывшегося   – С тех пор как меня угораздило побывать в этой грешной Черхавле, мне ежедневно снится какая-то дичь! – сердито сказал я Джуффину. – Сглазили они меня,
  test :  Слишком много кошмаров   Когда балансируешь над пропастью на узкой, скользкой от крови доске, ответ на закономерный вопрос: «Как меня сюда занесло?» – вряд ли принесёт прак

Класс: Клиффорд_Саймак
  train:  Всё живое...   Когда я выехал из нашего городишка и повернул на шоссе, позади оказался грузовик. Этакая тяжелая громадина с прицепом, и неслась она во весь дух. Шоссе здес
  test :  Зачарованное паломничество   1  Гоблин со стропил следил за прячущимся монахом, который шпионил за ученым. Гоблин ненавидел монаха и имел для этого все основания. Монах ник

Класс: Булгаков
  train:  Белая гвардия   Посвящается[1]  Любови Евгеньевне Белозерской[2]  Пошел мелкий снег и вдруг повалил хло-  льями. Ветер завыл; сделалась метель.  В одно мгновение темное небо
  test :  Дон Кихот ДЕЙСТВУЮЩИЕ ЛИЦА Алонсо Кихано, он же Дон Кихот Ламанчский.  Антония – его племянница.  Ключница Дон Кихота.  Санчо Панса – оруженосец Дон Кихота.  Перо Перес – де

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 круглые, бурые, без окон, торчат на сваях, как с
  test :  ОТЕЛЬ «У ПОГИБШЕГО АЛЬПИНИСТА»   ГЛАВА 1   Я остановил машину, вылез и снял черные очки. Все было так, как рассказывал Згут. Отель был двухэтажный, желтый с зеленым, над

Класс: Рэй Брэдберри
  train:  451° по Фаренгейту   ДОНУ КОНГДОНУ С БЛАГОДАРНОСТЬЮ  Если тебе дадут линованную бумагу, пиши поперёк.  Хуан Рамон Хименес  Часть 1  ОЧАГ И САЛАМАНДРА  Жечь было наслажден
  test :  Марсианские хроники   МОЕЙ ЖЕНЕ МАРГАРЕТ С ИСКРЕННЕЙ ЛЮБОВЬЮ  «Великое дело – способность удивляться, – сказал философ. – Космические полеты снова сделали всех нас детьми».
```

Рисунок 6. Вывод начальных отрывков

Затем зададим параметры преобразования и напишем контекстный менеджер для измерения времени операций

```
# Задание параметров преобразования
PARAMS = ((10000, 1000, 100), (15000, 1500, 250), (20000, 2000, 500))

# 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 7. Задание параметров преобразования

Далее напишем функцию разбиения последовательности на отрезки скользящим окном и функцию формирования выборок из последовательностей индексов

```python
# Функция разбиения последовательности на отрезки скользящим окном
# На входе - последовательность индексов, размер окна, шаг окна
def split_sequence(sequence, win_size, hop):
    # Последовательность разбивается на части до последнего полного окна
    return [
        sequence[i : i + win_size] for i in range(0, len(sequence) - win_size + 1, hop)
    ]


# Функция формирования выборок из последовательностей индексов
# формирует выборку отрезков и соответствующих им меток классов в виде one hot encoding
def vectorize_sequence(seq_list, win_size, hop):
    # В списке последовательности следуют в порядке их классов
    # Всего последовательностей в списке ровно столько, сколько классов
    class_count = len(seq_list)

    # Списки для исходных векторов и категориальных меток класса
    x, y = [], []

    # Для каждого класса:
    for cls in range(class_count):
        # Разбиение последовательности класса cls на отрезки
        vectors = split_sequence(seq_list[cls], win_size, hop)
        # Добавление отрезков в выборку
        x += vectors
        # Для всех отрезков класса cls добавление меток класса в виде ОНЕ
        y += [utils.to_categorical(cls, class_count)] * len(vectors)

    # Возврат результатов как numpy-массивов
    return np.array(x), np.array(y)
```

Рисунок 8. Функция разбиения последовательности на отрезки

Так же напишем сервисную функцию компиляции и обучения модели нейронной сети

```python
# 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()
    print("Start training...")
    # Обучение модели с заданными параметрами
    history = model.fit(
        x_train,
        y_train,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(x_val, y_val),
        verbose=1,
    )

    # 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 9. Функция компиляции и обучения модели

Далее напишем функцию вывода результатов оценки модели на заданных данных

```python
# Функция вывода результатов оценки модели на заданных данных
def eval_model(
    model, x, y_true, class_labels=[], cm_round=3, title="", figsize=(10, 10)
):
    # Вычисление предсказания сети
    y_pred = model.predict(x)
    # Построение матрицы ошибок
    cm = confusion_matrix(
        np.argmax(y_true, axis=1), np.argmax(y_pred, axis=1), normalize="true"
    )
    # Округление значений матрицы ошибок
    cm = np.around(cm, cm_round)

    # Отрисовка матрицы ошибок
    fig, ax = plt.subplots(figsize=figsize)
    ax.set_title(f"Нейросеть {title}: матрица ошибок нормализованная", fontsize=18)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
    disp.plot(ax=ax)
    plt.gca().images[-1].colorbar.remove()  # Стирание ненужной цветовой шкалы
    plt.xlabel("Предсказанные классы", fontsize=16)
    plt.ylabel("Верные классы", fontsize=16)
    fig.autofmt_xdate(rotation=45)  # Наклон меток горизонтальной оси при необходимости
    plt.show()

    print("-" * 100)
    print(f"Нейросеть: {title}")

    # Для каждого класса:
    for cls in range(len(class_labels)):
        # Определяется индекс класса с максимальным значением предсказания (уверенности)
        cls_pred = np.argmax(cm[cls])
        # Формируется сообщение о верности или неверности предсказания
        msg = "ВЕРНО :-)" if cls_pred == cls else "НЕВЕРНО :-("
        # Выводится текстовая информация о предсказанном классе и значении уверенности
        print(
            "Класс: {:<20} {:3.0f}% сеть отнесла к классу {:<20} - {}".format(
                class_labels[cls],
                100.0 * cm[cls, cls_pred],
                class_labels[cls_pred],
                msg,
            )
        )

    # Средняя точность распознавания определяется как среднее диагональных элементов матрицы ошибок
    print(
        "\nСредняя точность распознавания: {:3.0f}%".format(
            100.0 * cm.diagonal().mean()
        )
    )
```

Рисунок 10. Функция вывода результатов оценки модели

Напишем еще одну сервисную функцию обучения и оценки модели нейронной сети

```python
[ ]  # Совместная функция обучения и оценки модели нейронной сети
     def compile_train_eval_model(
         model,
         x_train,
         y_train,
         x_test,
         y_test,
         class_labels=CLASS_LIST,
         title="",
         optimizer="adam",
         epochs=50,
         batch_size=128,
         graph_size=(20, 5),
         cm_size=(10, 10),
     ):
         # Компиляция и обучение модели на заданных параметрах
         # В качестве проверочных используются тестовые данные
         compile_train_model(
             model,
             x_train,
             y_train,
             x_test,
             y_test,
             optimizer=optimizer,
             epochs=epochs,
             batch_size=batch_size,
             figsize=graph_size,
         )

         # Вывод результатов оценки работы модели на тестовых данных
         eval_model(
             model, x_test, y_test, class_labels=class_labels, title=title, figsize=cm_size
         )
         return model.evaluate(x_test, y_test)
```

Рисунок 11. Функция обучения и оценки модели

Далее напишем функцию для создания модели с заданным входным размером и архитектурой слоев. А также функцию для создания и обучения моделей с разной сложностью архитектуры

```python
# Функция для создания модели нейросети с заданным входным размером и архитектурой слоев
def build_model(size, layer_sizes):
    model = Sequential()
    model.add(Input((size,))) # Входной слой с заданным размером входных данных
    for units in layer_sizes:  # Добавляем скрытые слои согласно переданному списку размеров
        model.add(Dense(units))
        model.add(BatchNormalization())
        model.add(Activation("relu"))
        model.add(Dropout(0.25))
    model.add(Dense(CLASS_COUNT, activation="softmax")) # Добавляем скрытые слои согласно переданному списку размеров
    return model

# Функция для создания и обучения моделей с разной сложностью архитектуры
def create_and_train_models(size, results, x_train, y_train, x_test, y_test):
    # Список конфигураций моделей: название и структура скрытых слоев
    configurations = [
        ("Lite", [256]),  # Легкая модель с 1 скрытым слоем на 256 нейронов
        ("Middle", [256, 128]),  # Средняя модель с 2 скрытыми слоями
        ("Hard", [256, 128, 64]),   # Сложная модель с 3 скрытыми слоями
    ]

    # Обучаем и оцениваем каждую модель из списка конфигураций
    for name, layers in configurations:
        model = build_model(size, layers) # Построение модели
        _, acc = compile_train_eval_model( # Компиляция, обучение и оценка
            model,
            x_train,
            y_train,
            x_test,
            y_test,
            class_labels=CLASS_LIST,
            title=name,
        )
        results["Название модели"].append(name)
        results["Количество слоев"].append(len(layers))
        results["Нейроны в 1 слое"].append(layers[0])
        results["Точность (val_accuracy)"].append(acc)
```

Рисунок 12. Функция для создания и обучения модели

Затем выполним обучения моделей и построения графиков и матриц ошибок

```python
with timex():
    # Инициализация словаря для хранения результатов экспериментов
    results = {
        "Размер словаря": [],
        "Ширина окна": [],
        "Шаг окна": [],
        "Название модели": [],
        "Количество слоев": [],
        "Нейроны в 1 слое": [],
        "Точность (val_accuracy)": [],
    }
    # Проход по списку параметров (размер словаря, ширина окна, шаг окна)
    for vocab_size, win_size, win_hope in PARAMS:
        results["Размер словаря"].extend([vocab_size] * 3)
        results["Ширина окна"].extend([win_size] * 3)
        results["Шаг окна"].extend([win_hope] * 3)

        # 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,
            split=" ",
            oov_token="неизвестное_слово",
            char_level=False,
        )

        # Построение частотного словаря по обучающим текстам
        tokenizer.fit_on_texts(text_train)
        seq_train = tokenizer.texts_to_sequences(text_train)
        seq_test = tokenizer.texts_to_sequences(text_test)
        # Формирование обучающей выборки
        x_train, y_train = vectorize_sequence(seq_train, win_size, win_hope)
        # Формирование тестовой выборки
        x_test, y_test = vectorize_sequence(seq_test, win_size, win_hope)
        x_train = tokenizer.sequences_to_matrix(x_train.tolist()).astype("float16")
        x_test = tokenizer.sequences_to_matrix(x_test.tolist()).astype("float16")
        create_and_train_models(vocab_size, results, x_train, y_train, x_test, y_test)
```

Рисунок 13. Обучение моделей

Далее посмотрим на созданные модели и их графики и матрицы ошибок

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	2,560,256
batch_normalization (BatchNormalization)	(None, 256)	1,024
activation (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 6)	1,542

Total params: 2,562,822 (9.78 MB)
Trainable params: 2,562,310 (9.77 MB)
Non-trainable params: 512 (2.00 KB)

Рисунок 14. Структура первой модели

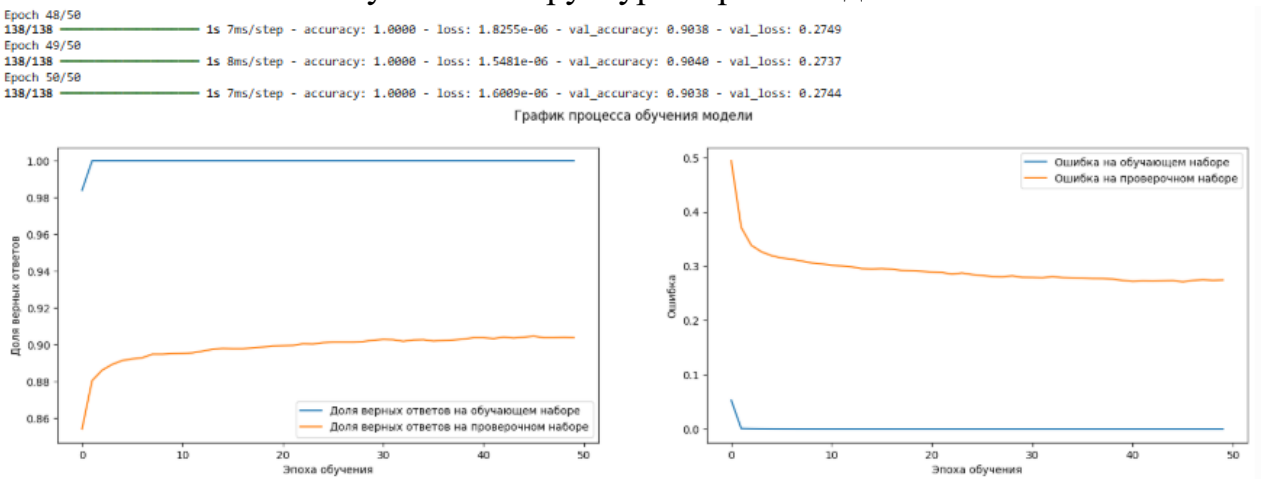


Рисунок 15. Графики процесса обучения модели первой модели

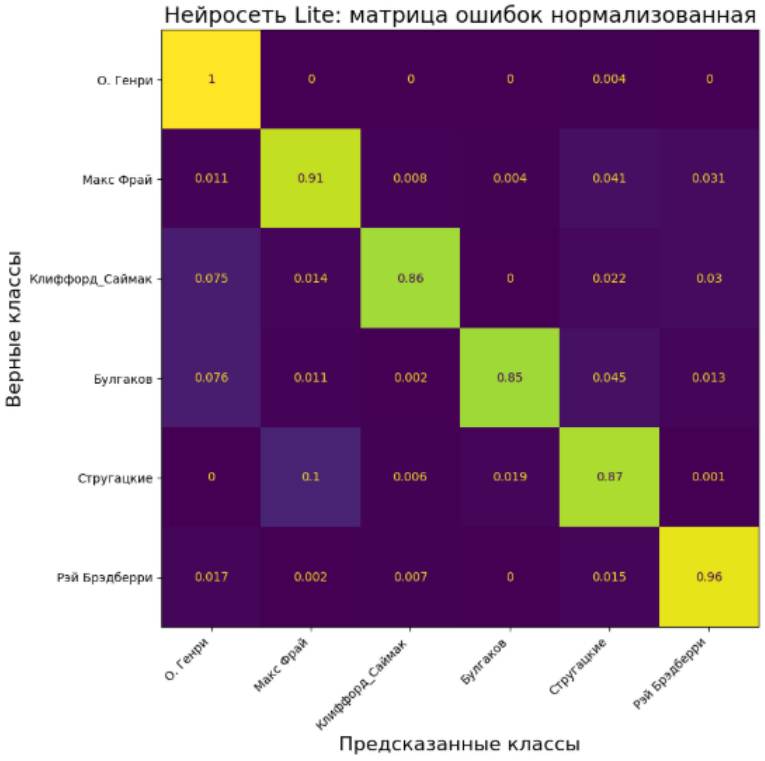


Рисунок 16. Матрица ошибок первой модели

```
Нейросеть: Lite
Класс: О. Генри          100% сеть отнесла к классу О. Генри          - ВЕРНО :-)
Класс: Макс Фрай          90% сеть отнесла к классу Макс Фрай         - ВЕРНО :-)
Класс: Клиффорд_Саймак    86% сеть отнесла к классу Клиффорд_Саймак   - ВЕРНО :-)
Класс: Булгаков           85% сеть отнесла к классу Булгаков          - ВЕРНО :-)
Класс: Стругацкие         87% сеть отнесла к классу Стругацкие        - ВЕРНО :-)
Класс: Рэй Брэдберри      96% сеть отнесла к классу Рэй Брэдберри     - ВЕРНО :-)

Средняя точность распознавания:  91%
```

Рисунок 17. Средняя точность распознавания первой модели

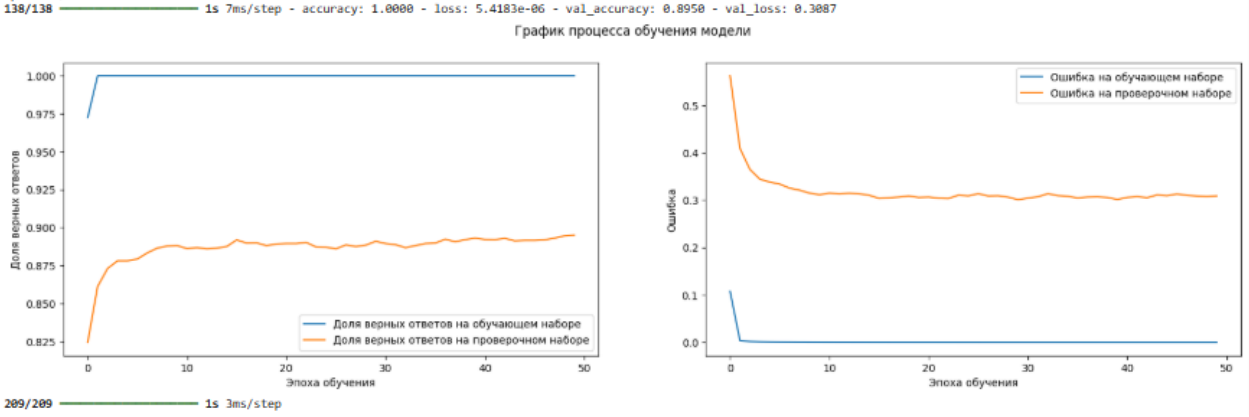
Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 256)	2,560,256
batch_normalization_1 (BatchNormalization)	(None, 256)	1,024
activation_1 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32,896
batch_normalization_2 (BatchNormalization)	(None, 128)	512
activation_2 (Activation)	(None, 128)	0
dropout_2 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 6)	774

Total params: 2,595,462 (9.90 MB)
Trainable params: 2,594,694 (9.90 MB)
Non-trainable params: 768 (3.00 KB)
Start training...

Рисунок 18. Структура второй модели

```
Epoch 48/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 1.0000 - loss: 6.2765e-06 - val_accuracy: 0.8931 - val_loss: 0.3085
Epoch 49/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 1.0000 - loss: 4.5289e-06 - val_accuracy: 0.8946 - val_loss: 0.3076
Epoch 50/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 1.0000 - loss: 5.4183e-06 - val_accuracy: 0.8950 - val_loss: 0.3087
```




```
209/209 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

Рисунок 19. Графики процесса обучения модели второй модели

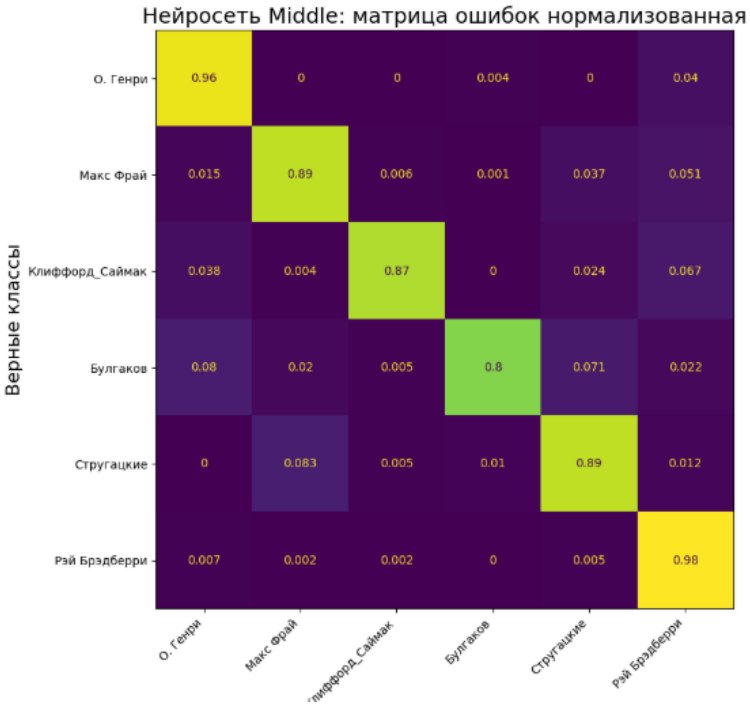


Рисунок 20. Матрица ошибок второй модели

```
Нейросеть: Middle
Класс: О. Генри                 96% сеть отнесла к классу О. Генри                 - ВЕРНО :-)
Класс: Макс Фрай                89% сеть отнесла к классу Макс Фрай                - ВЕРНО :-)
Класс: Клиффорд_Саймак          87% сеть отнесла к классу Клиффорд_Саймак          - ВЕРНО :-)
Класс: Булгаков                 80% сеть отнесла к классу Булгаков                 - ВЕРНО :-)
Класс: Стругацкие               89% сеть отнесла к классу Стругацкие               - ВЕРНО :-)
Класс: Рэй Брэдберри            98% сеть отнесла к классу Рэй Брэдберри            - ВЕРНО :-)
```

Рисунок 21. Средняя точность распознавания второй модели

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 256)	2,560,256
batch_normalization_3 (BatchNormalization)	(None, 256)	1,024
activation_3 (Activation)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32,896
batch_normalization_4 (BatchNormalization)	(None, 128)	512
activation_4 (Activation)	(None, 128)	0
dropout_4 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8,256
batch_normalization_5 (BatchNormalization)	(None, 64)	256
activation_5 (Activation)	(None, 64)	0
dropout_5 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 6)	390

Total params: 2,603,590 (9.93 MB)
Trainable params: 2,602,694 (9.93 MB)
Non-trainable params: 896 (3.50 KB)

Рисунок 22. Структура третьей модели

```
Epoch 47/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9960 - loss: 0.0140 - val_accuracy: 0.8232 - val_loss: 0.9004
Epoch 48/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9992 - loss: 0.0037 - val_accuracy: 0.8459 - val_loss: 0.6513
Epoch 49/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9996 - loss: 0.0012 - val_accuracy: 0.8319 - val_loss: 0.6709
Epoch 50/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 6.5170e-04 - val_accuracy: 0.8503 - val_loss: 0.5857
```

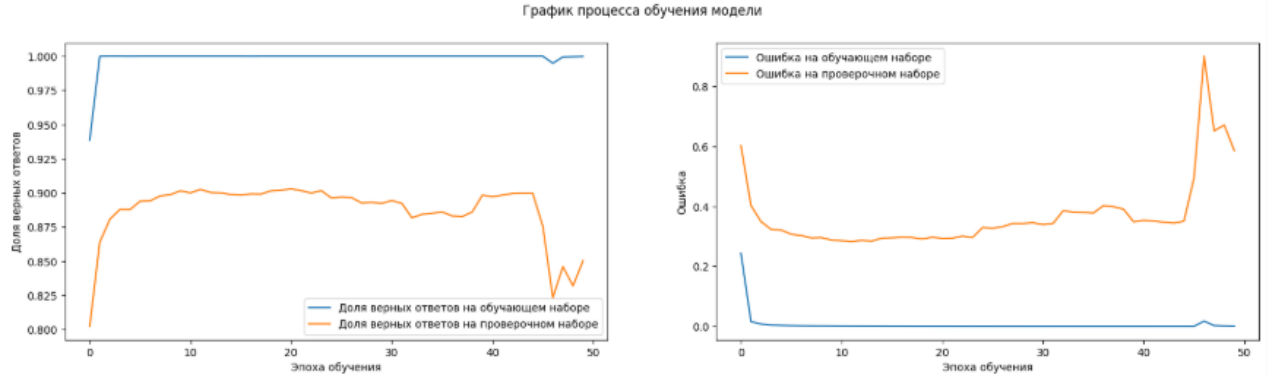


Рисунок 23. Графики процесса обучения модели третьей модели

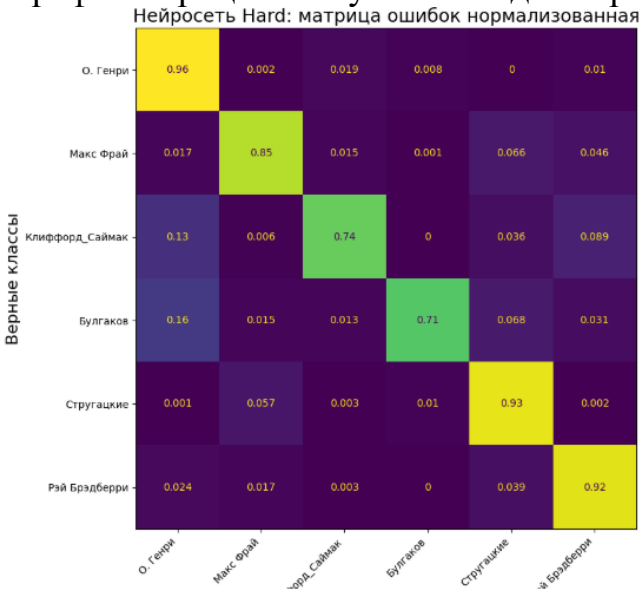


Рисунок 24. Матрица ошибок третьей модели

```
Нейросеть: Hard
Класс: О. Генри                 96% сеть отнесла к классу О. Генри                 - ВЕРНО :-)
Класс: Макс Фрай                86% сеть отнесла к классу Макс Фрай                - ВЕРНО :-)
Класс: Клиффорд_Саймак          74% сеть отнесла к классу Клиффорд_Саймак          - ВЕРНО :-)
Класс: Булгаков                 71% сеть отнесла к классу Булгаков                 - ВЕРНО :-)
Класс: Стругацкие               93% сеть отнесла к классу Стругацкие               - ВЕРНО :-)
Класс: Рэй Брэдберри            92% сеть отнесла к классу Рэй Брэдберри            - ВЕРНО :-)
```

Рисунок 25. Средняя точность распознавания третьей модели

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 256)	3,840,256
batch_normalization_6 (BatchNormalization)	(None, 256)	1,024
activation_6 (Activation)	(None, 256)	0
dropout_6 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 6)	1,542

```
Total params: 3,842,822 (14.66 MB)
Trainable params: 3,842,310 (14.66 MB)
Non-trainable params: 512 (2.00 KB)
```

Рисунок 26. Структура четвертой модели

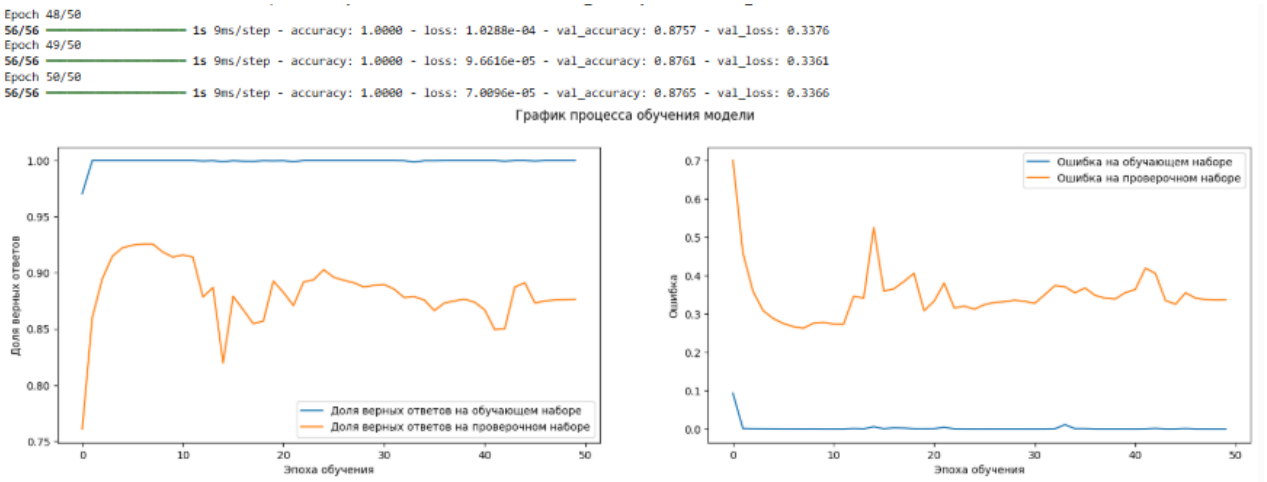


Рисунок 27. Графики процесса обучения модели четвертой модели

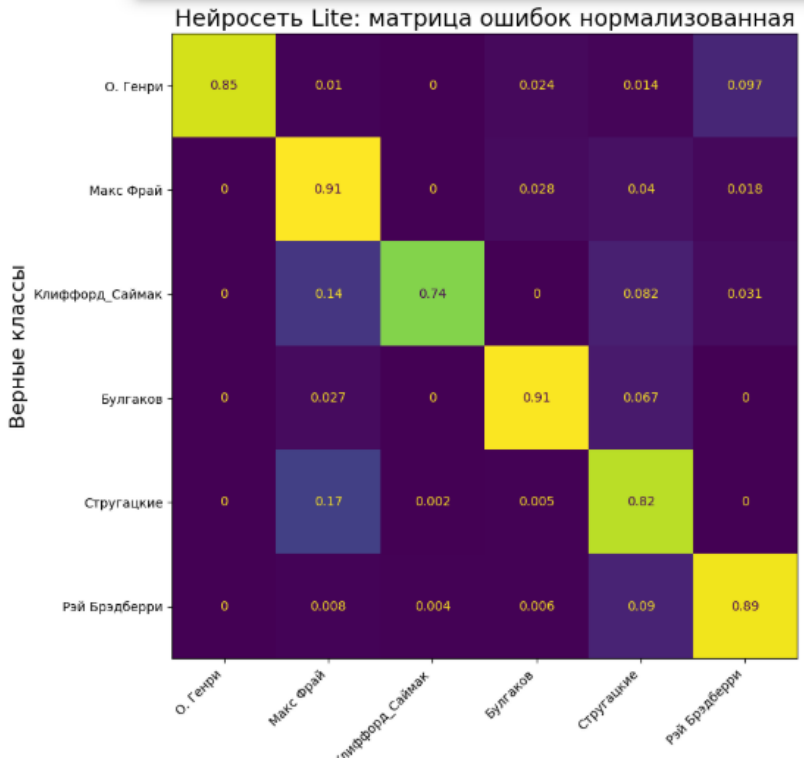


Рисунок 28. Матрица ошибок четвертой модели

```
Нейросеть: Lite
Класс: О. Генри                 86% сеть отнесла к классу О. Генри                 - ВЕРНО :-)
Класс: Макс Фрай                91% сеть отнесла к классу Макс Фрай                - ВЕРНО :-)
Класс: Клиффорд_Саймак          74% сеть отнесла к классу Клиффорд_Саймак          - ВЕРНО :-)
Класс: Булгаков                 91% сеть отнесла к классу Булгаков                 - ВЕРНО :-)
Класс: Стругацкие               82% сеть отнесла к классу Стругацкие               - ВЕРНО :-)
Класс: Рэй Брэдберри            89% сеть отнесла к классу Рэй Брэдберри            - ВЕРНО :-)
```

Рисунок 29. Средняя точность распознавания четвертой модели

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_11 (Dense)	(None, 256)	3,840,256
batch_normalization_7 (BatchNormalization)	(None, 256)	1,024
activation_7 (Activation)	(None, 256)	0
dropout_7 (Dropout)	(None, 256)	0
dense_12 (Dense)	(None, 128)	32,896
batch_normalization_8 (BatchNormalization)	(None, 128)	512
activation_8 (Activation)	(None, 128)	0
dropout_8 (Dropout)	(None, 128)	0
dense_13 (Dense)	(None, 6)	774

Total params: 3,875,462 (14.78 MB)
Trainable params: 3,874,694 (14.78 MB)
Non-trainable params: 768 (3.00 KB)

Рисунок 30. Структура пятой модели

```
Epoch 48/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9995 - loss: 0.0042 - val_accuracy: 0.9095 - val_loss: 0.2943
Epoch 49/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9050 - val_loss: 0.3126
Epoch 50/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 1.0000 - loss: 7.2616e-04 - val_accuracy: 0.8990 - val_loss: 0.3247
```

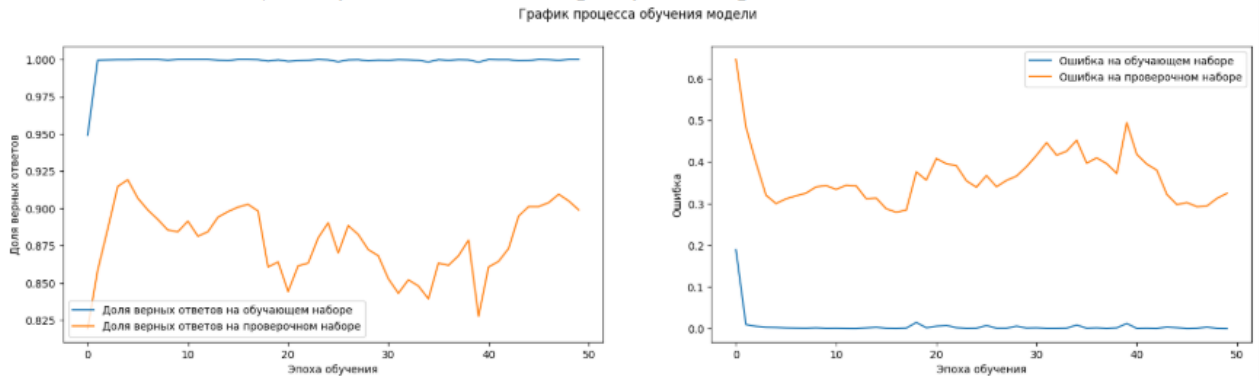


Рисунок 31. Графики процесса обучения модели пятой модели

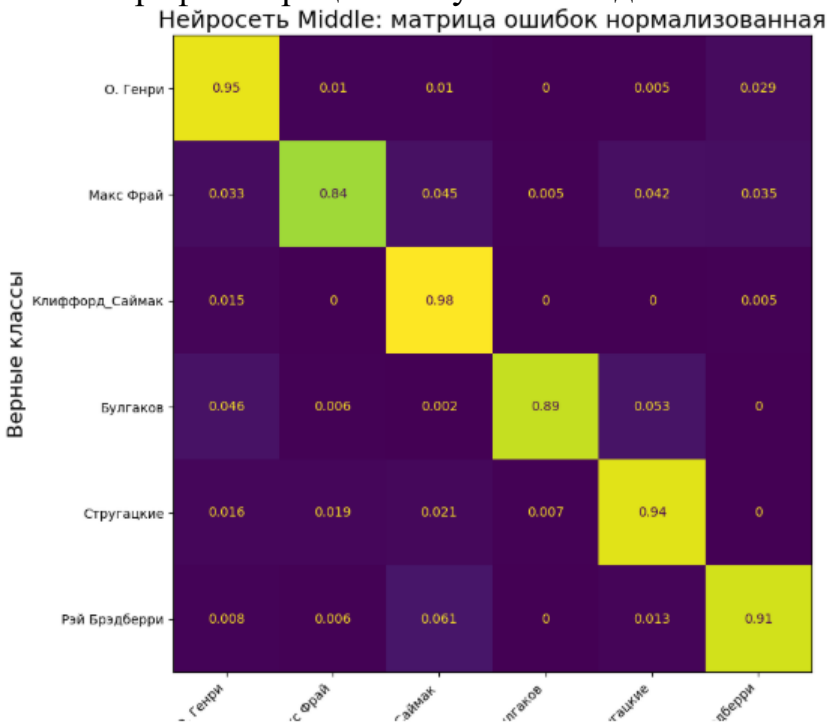


Рисунок 32. Матрица ошибок пятой модели

```
Нейросеть: Middle
Класс: О. Генри          95% сеть отнесла к классу О. Генри          - ВЕРНО :-)
Класс: Макс Фрай         84% сеть отнесла к классу Макс Фрай         - ВЕРНО :-)
Класс: Клиффорд_Саймак   98% сеть отнесла к классу Клиффорд_Саймак   - ВЕРНО :-)
Класс: Булгаков          89% сеть отнесла к классу Булгаков          - ВЕРНО :-)
Класс: Стругацкие        94% сеть отнесла к классу Стругацкие        - ВЕРНО :-)
Класс: Рэй Брэдберри     91% сеть отнесла к классу Рэй Брэдберри     - ВЕРНО :-)
```

Рисунок 33. Средняя точность распознавания пятой модели

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_14 (Dense)	(None, 256)	3,840,256
batch_normalization_9 (BatchNormalization)	(None, 256)	1,024
activation_9 (Activation)	(None, 256)	0
dropout_9 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 128)	32,896
batch_normalization_10 (BatchNormalization)	(None, 128)	512
activation_10 (Activation)	(None, 128)	0
dropout_10 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 64)	8,256
batch_normalization_11 (BatchNormalization)	(None, 64)	256
activation_11 (Activation)	(None, 64)	0
dropout_11 (Dropout)	(None, 64)	0
dense_17 (Dense)	(None, 6)	390

Total params: 3,883,590 (14.81 MB)
Trainable params: 3,882,694 (14.81 MB)
Non-trainable params: 896 (3.50 KB)

Рисунок 34. Структура шестой модели

```
Epoch 48/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9999 - loss: 0.0012 - val_accuracy: 0.8295 - val_loss: 0.5050
Epoch 49/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 0.8400 - val_loss: 0.4785
Epoch 50/50
56/56 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 1.0000 - loss: 9.0071e-04 - val_accuracy: 0.8186 - val_loss: 0.5262
```

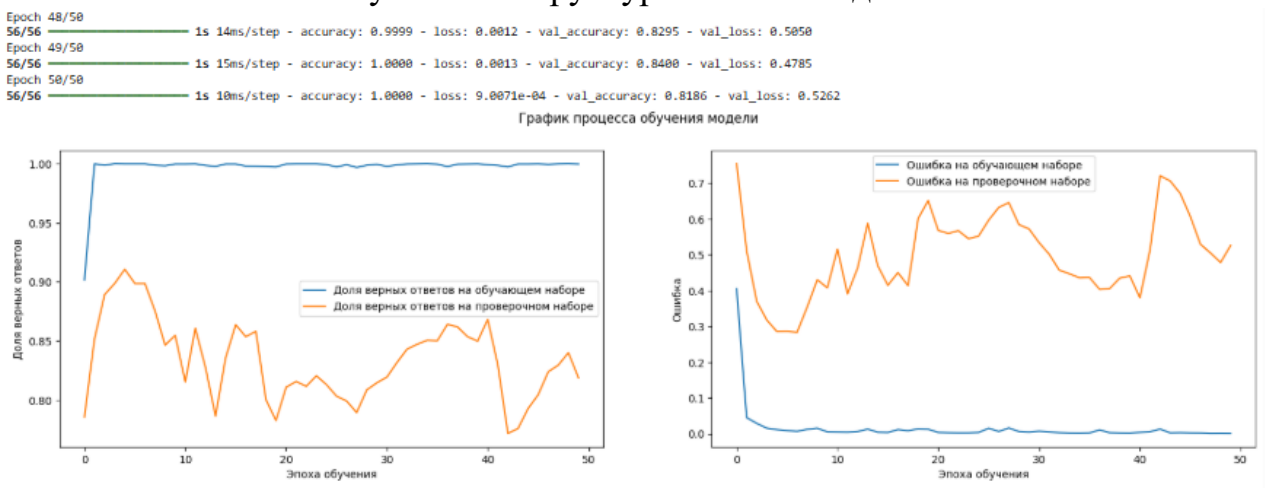


Рисунок 35. Графики процесса обучения модели шестой модели

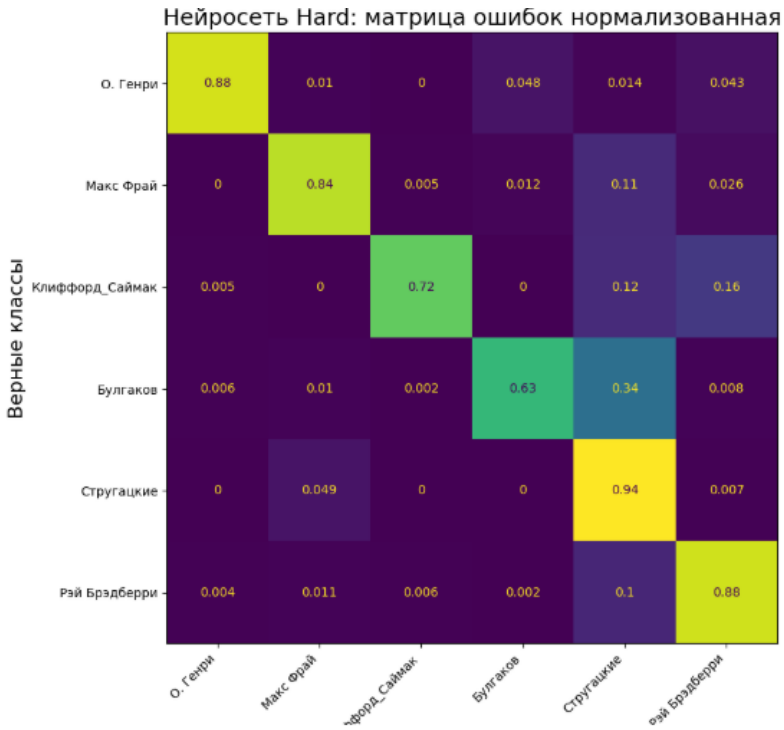


Рисунок 36. Матрица ошибок шестой модели

```
Нейросеть: Hard
Класс: О. Генри                 88% сеть отнесла к классу О. Генри                 - ВЕРНО :-)
Класс: Макс Фрай                84% сеть отнесла к классу Макс Фрай                - ВЕРНО :-)
Класс: Клиффорд_Саймак          72% сеть отнесла к классу Клиффорд_Саймак          - ВЕРНО :-)
Класс: Булгаков                 63% сеть отнесла к классу Булгаков                 - ВЕРНО :-)
Класс: Стругацкие               94% сеть отнесла к классу Стругацкие               - ВЕРНО :-)
Класс: Рэй Брэдберри            88% сеть отнесла к классу Рэй Брэдберри            - ВЕРНО :-)
```

Рисунок 37. Средняя точность распознавания шестой модели

```
Model: "sequential_6"
```

Layer (type)	Output Shape	Param #
dense_18 (Dense)	(None, 256)	5,120,256
batch_normalization_12 (BatchNormalization)	(None, 256)	1,024
activation_12 (Activation)	(None, 256)	0
dropout_12 (Dropout)	(None, 256)	0
dense_19 (Dense)	(None, 6)	1,542

```
Total params: 5,122,822 (19.54 MB)
Trainable params: 5,122,310 (19.54 MB)
Non-trainable params: 512 (2.00 KB)
```

Рисунок 38. Структура седьмой модели

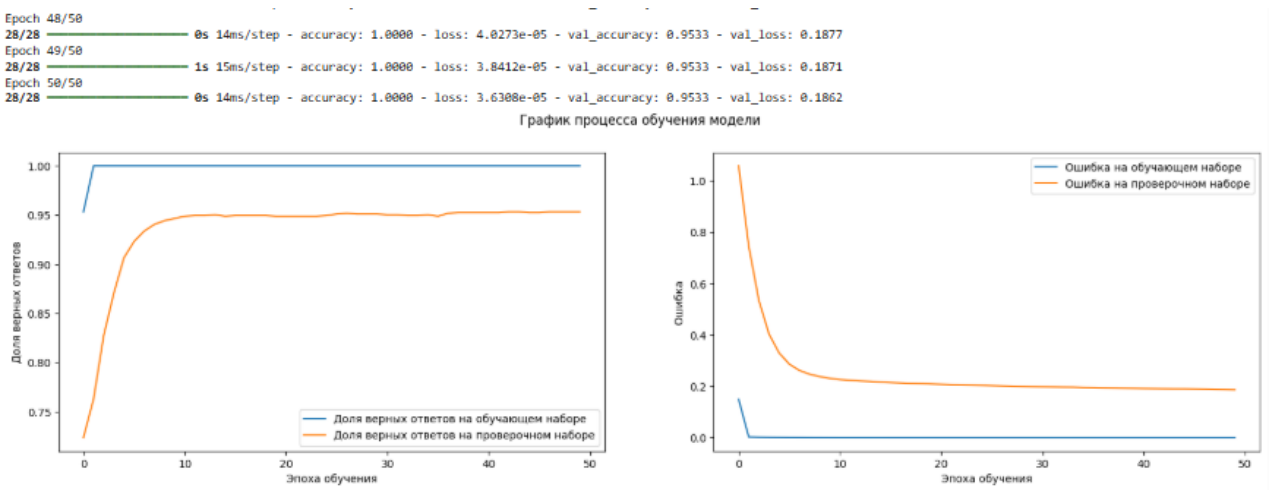


Рисунок 39. Графики процесса обучения модели седьмой модели

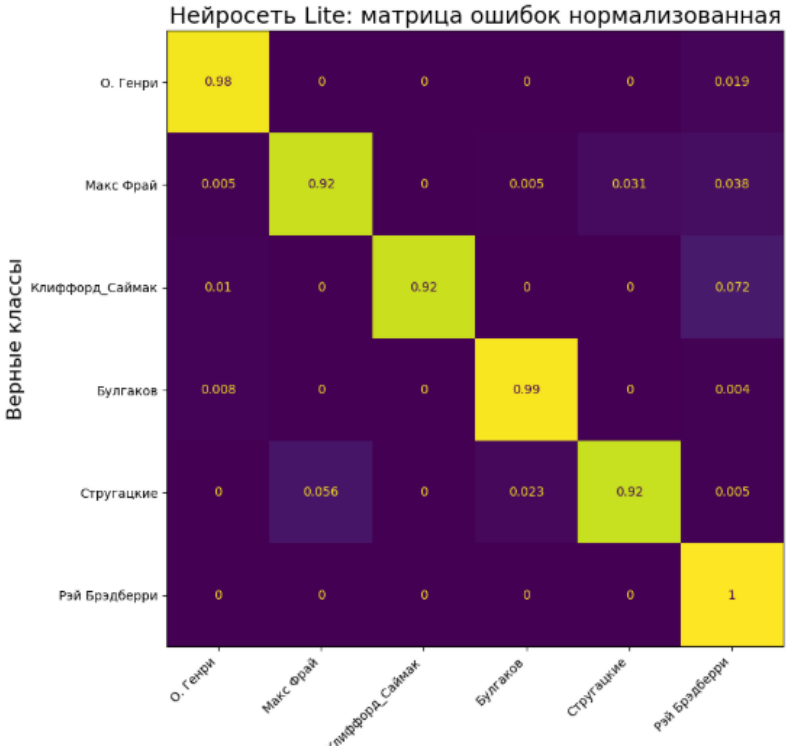


Рисунок 40. Матрица ошибок седьмой модели

```
Нейросеть: Lite
Класс: О. Генри                 98% сеть отнесла к классу О. Генри                 - ВЕРНО :-)
Класс: Макс Фрай                92% сеть отнесла к классу Макс Фрай                - ВЕРНО :-)
Класс: Клиффорд_Саймак          92% сеть отнесла к классу Клиффорд_Саймак          - ВЕРНО :-)
Класс: Булгаков                 99% сеть отнесла к классу Булгаков                 - ВЕРНО :-)
Класс: Стругацкие               92% сеть отнесла к классу Стругацкие               - ВЕРНО :-)
Класс: Рэй Брэдберри           100% сеть отнесла к классу Рэй Брэдберри            - ВЕРНО :-)
```

Рисунок 41. Средняя точность распознавания седьмой модели

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_20 (Dense)	(None, 256)	5,120,256
batch_normalization_13 (BatchNormalization)	(None, 256)	1,024
activation_13 (Activation)	(None, 256)	0
dropout_13 (Dropout)	(None, 256)	0
dense_21 (Dense)	(None, 128)	32,896
batch_normalization_14 (BatchNormalization)	(None, 128)	512
activation_14 (Activation)	(None, 128)	0
dropout_14 (Dropout)	(None, 128)	0
dense_22 (Dense)	(None, 6)	774

Total params: 5,155,462 (19.67 MB)
Trainable params: 5,154,694 (19.66 MB)
Non-trainable params: 768 (3.00 KB)

Рисунок 42. Структура восьмой модели

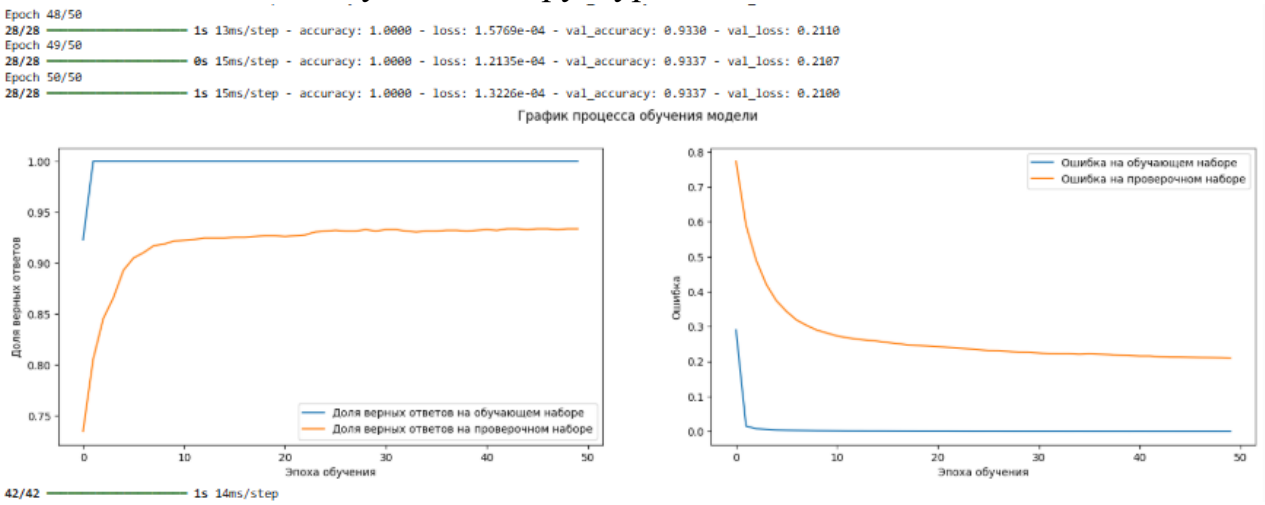


Рисунок 43. Графики процесса обучения модели восьмой модели

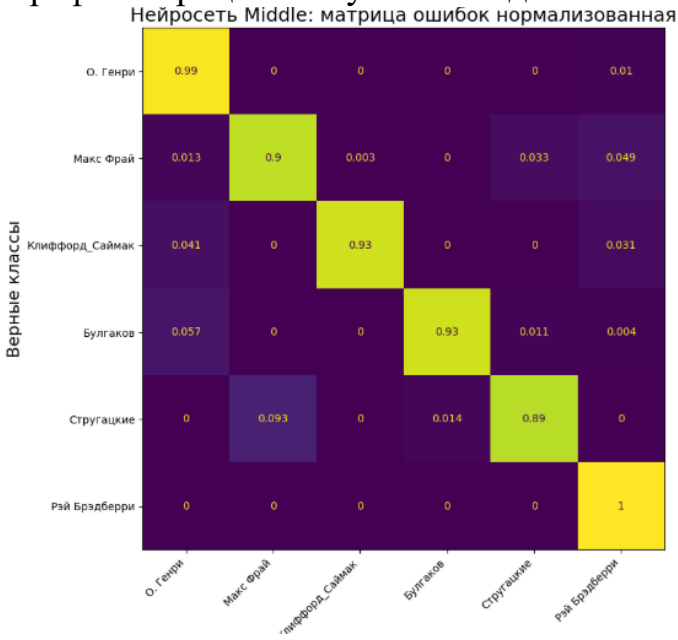


Рисунок 44. Матрица ошибок восьмой модели

```
Нейросеть: Middle
Класс: О. Генри          99% сеть отнесла к классу О. Генри          - ВЕРНО :-)
Класс: Макс Фрай         90% сеть отнесла к классу Макс Фрай         - ВЕРНО :-)
Класс: Клиффорд_Саймак   93% сеть отнесла к классу Клиффорд_Саймак   - ВЕРНО :-)
Класс: Булгаков          93% сеть отнесла к классу Булгаков          - ВЕРНО :-)
Класс: Стругацкие        89% сеть отнесла к классу Стругацкие        - ВЕРНО :-)
Класс: Рэй Брэдберри    100% сеть отнесла к классу Рэй Брэдберри     - ВЕРНО :-)

Средняя точность распознавания:  94%
```

Рисунок 45. Средняя точность распознавания восьмой модели

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_23 (Dense)	(None, 256)	5,120,256
batch_normalization_15 (BatchNormalization)	(None, 256)	1,024
activation_15 (Activation)	(None, 256)	0
dropout_15 (Dropout)	(None, 256)	0
dense_24 (Dense)	(None, 128)	32,896
batch_normalization_16 (BatchNormalization)	(None, 128)	512
activation_16 (Activation)	(None, 128)	0
dropout_16 (Dropout)	(None, 128)	0
dense_25 (Dense)	(None, 64)	8,256
batch_normalization_17 (BatchNormalization)	(None, 64)	256
activation_17 (Activation)	(None, 64)	0
dropout_17 (Dropout)	(None, 64)	0
dense_26 (Dense)	(None, 6)	390

Total params: 5,163,590 (19.70 MB)
Trainable params: 5,162,694 (19.69 MB)
Non-trainable params: 896 (3.50 KB)

Рисунок 46. Структура девятой модели

```
Epoch 48/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 1.0000 - loss: 6.3645e-04 - val_accuracy: 0.8765 - val_loss: 0.3350
Epoch 49/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 1.0000 - loss: 5.8072e-04 - val_accuracy: 0.8758 - val_loss: 0.3355
Epoch 50/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 6.7820e-04 - val_accuracy: 0.8750 - val_loss: 0.3344
```

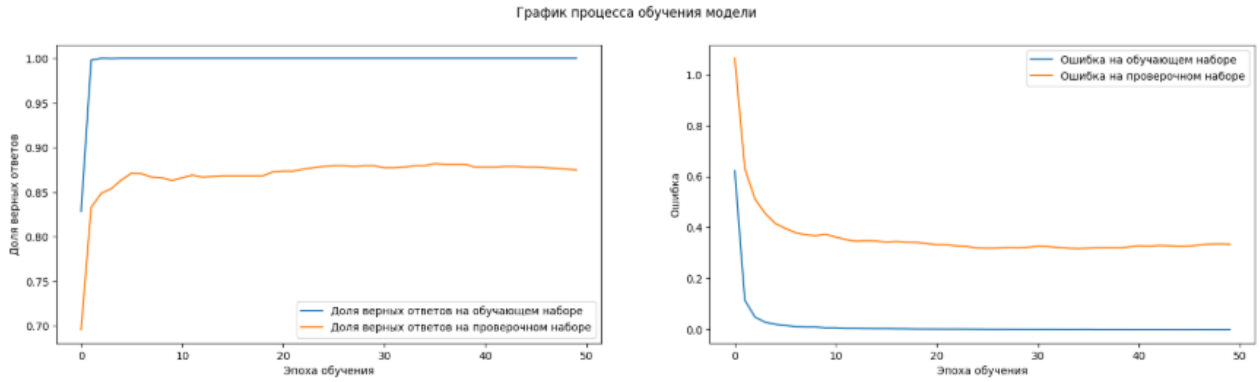


Рисунок 47. Графики процесса обучения модели девятой модели

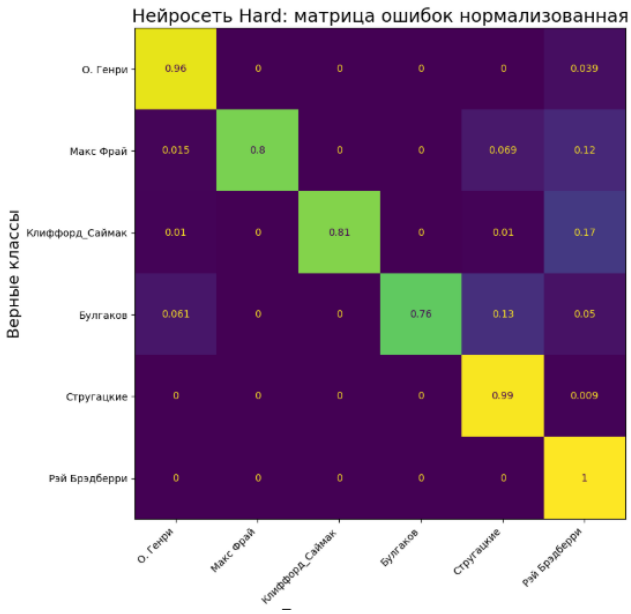


Рисунок 48. Матрица ошибок девятой модели

```
Нейросеть: Hard
Класс: О. Генри          96% сеть отнесла к классу О. Генри          - ВЕРНО :-)
Класс: Макс Фрай         80% сеть отнесла к классу Макс Фрай         - ВЕРНО :-)
Класс: Клиффорд_Саймак   81% сеть отнесла к классу Клиффорд_Саймак   - ВЕРНО :-)
Класс: Булгаков          76% сеть отнесла к классу Булгаков          - ВЕРНО :-)
Класс: Стругацкие        99% сеть отнесла к классу Стругацкие        - ВЕРНО :-)
Класс: Рэй Брэдберри    100% сеть отнесла к классу Рэй Брэдберри     - ВЕРНО :-)

Средняя точность распознавания:  89%
```

Рисунок 49. Средняя точность распознавания девятой модели

Далее посмотрим на точности полученных моделей, для этого выведем таблицу с моделями и их точностью

```
import pandas as pd

df = pd.DataFrame(results)
df
```

	Размер словаря	Ширина окна	Шаг окна	Название модели	Количество слоев	Нейроны в 1 слое	Точность (val_accuracy)
0	10000	1000	100	Lite	1	256	0.903829
1	10000	1000	100	Middle	2	256	0.895005
2	10000	1000	100	Hard	3	256	0.850284
3	15000	1500	250	Lite	1	256	0.876455
4	15000	1500	250	Middle	2	256	0.898988
5	15000	1500	250	Hard	3	256	0.818626
6	20000	2000	500	Lite	1	256	0.953313
7	20000	2000	500	Middle	2	256	0.933735
8	20000	2000	500	Hard	3	256	0.875000

Рисунок 50. Таблица с полученными моделями

Задание 2.

Условие: в данном задании предоставляется возможность поработать с задачей по распознаванию позитивных и негативных отзывов людей по автомобилю Tesla. База для обучения содержит два текстовых файла с рядом строчных отзывов с мнением людей об автомобиле Tesla, соответственно негативного и позитивного содержания. Ссылка на скачивание базы уже включена в ноутбук задания.

Необходимо выполнить следующие действия:

1. Загрузить базу по ссылке и подговить файлы базы для обработки.
2. Создать обучающую и проверочную выборки, обратив особое внимание на балансировку базы: количество примеров каждого класса должно быть примерно одного порядка.
3. Подготовить выборки для обучения и обучить сеть. Добиться результата точности сети в 85-90% на проверочной выборке.

Выполнения задания начнем с импорта необходимых библиотек

```
# Стандартная библиотека
import os                                   # Для работы с файлами в Colaboratory
import zipfile                              # Работа с архивами

# Сторонние библиотеки
import gdown                                # Загрузка датасетов из облака Google
import matplotlib.pyplot as plt             # Отрисовка графиков
import numpy as np                          # Работа с массивами данных
import pandas as pd                         # Работа с таблицами

# TensorFlow / Keras
from tensorflow.keras import utils          # Функции-утилиты для работы с категориальными данными
from tensorflow.keras.layers import (       # Основные слои нейронной сети
    Activation,
    BatchNormalization,
    Dense,
    Dropout,
    Embedding,
    Flatten,
    Input,
    SpatialDropout1D,
)
from tensorflow.keras.models import Sequential  # Класс для построения последовательной модели нейронной сети
from tensorflow.keras.optimizers import Adam    # Оптимизаторы
from tensorflow.keras.preprocessing.sequence import pad_sequences  # Приведение длины текстов к фиксированному значению
from tensorflow.keras.preprocessing.text import Tokenizer          # Токенизация текстов

%matplotlib inline  # Для отображения графиков внутри ноутбука
```

Рисунок 51. Импорт библиотек

Далее выполним загрузку датасаета

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

```
'tesla.zip'
```

Рисунок 52. Загрузка датасета

После чего выполним распаковку скаченного архива

```
# Распаковка архива
with zipfile.ZipFile('tesla.zip', 'r') as zip_ref:
    zip_ref.extractall('tesla')

# Просмотр содержимого папки
print("\n".join(os.listdir('tesla')))
```

```
'Негативный отзыв.txt'  'Позитивный отзыв.txt'
```

Рисунок 53. Распаковка архива

Далее напишем функцию для чтения файла и посчитаем количества классов

```
# Обьявляем функции для чтения файла. На вход отправляем путь к файлу
def read_text(file_name):

  # Задаем открытие нужного файла в режиме чтения
  read_file = open(file_name, 'r')

  # Читаем текст
  text = read_file.read()

  # Переносы строки переводим в пробелы
  text = text.replace("\n", " ")

  # Возвращаем текст файла
  return text

# Обьявляем интересующие нас классы
class_names = ["Негативный отзыв", "Позитивный отзыв"]

# Считаем количество классов
num_classes = len(class_names)
```

Рисунок 54. Функция для чтения файла

Затем создадим список под тексты для обучающей выборки и выполним цикл по текстовым файлам в папке отзывов

```
# Создаём список под тексты для обучающей выборки
texts_list = []

# Циклом проводим итерацию по текстовым файлам в папке отзывов
for j in os.listdir('tesla/'):

  # Добавляем каждый файл в общий список для выборки
        texts_list.append(read_text('tesla/' + j))

        # Выводим на экран сообщение о добавлении файла
        print(j, 'добавлен в обучающую выборку')
```

```
Негативный отзыв.txt добавлен в обучающую выборку
Позитивный отзыв.txt добавлен в обучающую выборку
```

Рисунок 55. Тексты в один список

Далее узнаем объем каждого текста в словах и символах

```
# Узнаем объём каждого текста в словах и символах
texts_len = [len(text) for text in texts_list]

# Устанавливаем "счётчик" номера текста
t_num = 0

# Выводим на экран  информационное сообщение
print(f'Размеры текстов по порядку (в токенах):')

# Циклом проводим итерацию по списку с объёмами текстов
for text_len in texts_len:

  # Запускаем "счётчик" номера текста
  t_num += 1

  # Выводим на экран сообщение о номере и объёме текста
  print(f'Текст №{t_num}: {text_len}')
```

```
Размеры текстов по порядку (в токенах):
Текст №1: 134535
Текст №2: 213381
```

Рисунок 56. Выводим на экран сообщение о номере и объёме текста

Далее рассчитаем, сколько символов составит 80% объёма каждого текста, чтобы по полученному индексу отделить эти 80% на обучающую и оставшиеся 20% на проверочную выборку. Эти значения необходимы для подготовки деления на выборки слайсингом по индексу

```
# Создаём список с вложенным циклом по длинам текстов, где i - 100% текста, i/5 - 20% текста
train_len_shares = [(i - round(i/5)) for i in texts_len]

# Устанавливаем "счётчик" номера текста
t_num = 0

# Циклом проводим итерацию по списку с объёмами текстов равными 80% от исходных
for train_len_share in train_len_shares:

  # Запускаем "счётчик" номера текста
  t_num += 1

  # Выводим на экран сообщение о номере и объёме текста в 80% от исходного
  print(f'Доля 80% от текста №{t_num}: {train_len_share} символов')
```

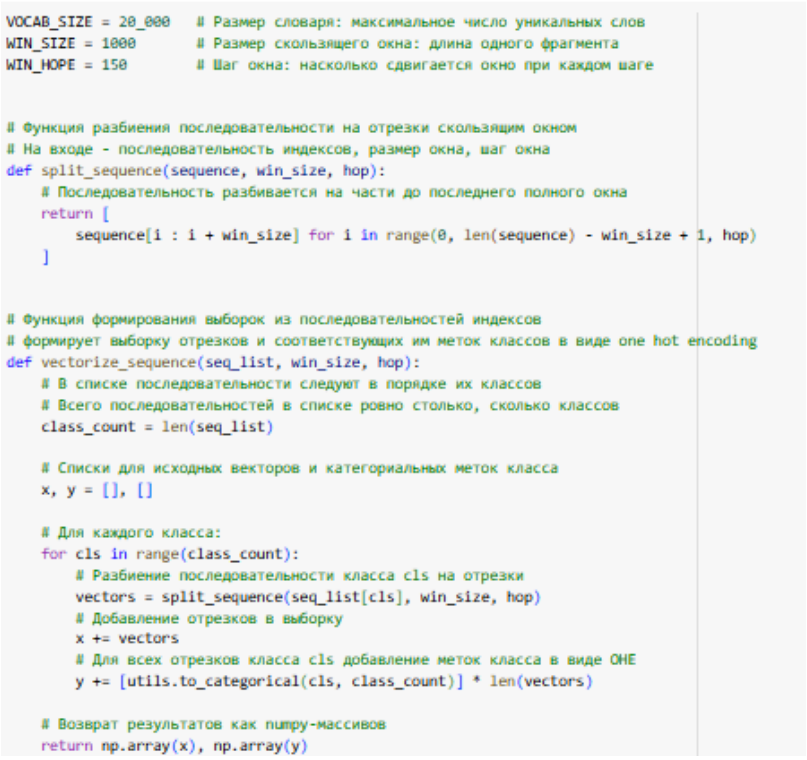
```
Доля 80% от текста №1: 107628 символов
Доля 80% от текста №2: 170705 символов
```

Рисунок 57. Вывод на экран сообщения о номере и объёме текста Далее произведем нарезку (метод слайсинга) по полученному ранее индексу для формирования текстов отдельно для обучающей(80%) и проверочной(20%) выборок

```
from itertools import chain

text_train = [] # Список для хранения обучающих фрагментов текстов
text_validate = [] # Список для хранения валидационных (проверочных) фрагментов текстов

# Проход по списку текстов и соответствующих длин обучающих частей
for i, text in enumerate(texts_list):
    text_train.append(text[:train_len_shares[i]])
    text_validate.append(text[train_len_shares[i]:])
```

Рисунок 58. Проход по списку текстов

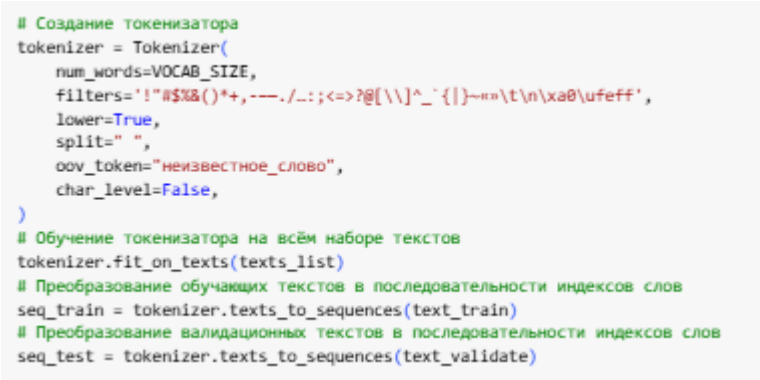
Затем зададим параметры и выполним функцию разбиения последовательности на отрезки скользящим окном и функцию формирования выборок из последовательностей индексов



```
VOCAB_SIZE = 20_000   # Размер словаря: максимальное число уникальных слов
WIN_SIZE = 1000       # Размер скользящего окна: длина одного фрагмента
WIN_HOPE = 150        # Шаг окна: насколько сдвигается окно при каждом шаге


# Функция разбиения последовательности на отрезки скользящим окном
# На входе - последовательность индексов, размер окна, шаг окна
def split_sequence(sequence, win_size, hop):
    # Последовательность разбивается на части до последнего полного окна
    return [
        sequence[i : i + win_size] for i in range(0, len(sequence) - win_size + 1, hop)
    ]


# Функция формирования выборок из последовательностей индексов
# формирует выборку отрезков и соответствующих им меток классов в виде one hot encoding
def vectorize_sequence(seq_list, win_size, hop):
    # В списке последовательности следуют в порядке их классов
    # Всего последовательностей в списке ровно столько, сколько классов
    class_count = len(seq_list)

    # Списки для исходных векторов и категориальных меток класса
    x, y = [], []

    # Для каждого класса:
    for cls in range(class_count):
        # Разбиение последовательности класса cls на отрезки
        vectors = split_sequence(seq_list[cls], win_size, hop)
        # Добавление отрезков в выборку
        x += vectors
        # Для всех отрезков класса cls добавление меток класса в виде OHE
        y += [utils.to_categorical(cls, class_count)] * len(vectors)

    # Возврат результатов как numpy-массивов
    return np.array(x), np.array(y)
```

Рисунок 59. Функция формирования выборок из последовательностей индексов

Далее создадим токенизатор и выполним его обучение на всём наборе текстов

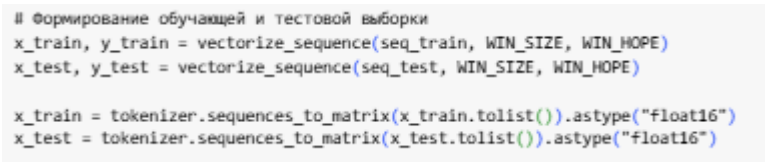


```
# Создание токенизатора
tokenizer = Tokenizer(
    num_words=VOCAB_SIZE,
    filters='!"#$%&()*+,-—./…:;<=>?@[\\]^_`{|}~«»\t\n\xa0\ufeff',
    lower=True,
    split=" ",
    oov_token="неизвестное_слово",
    char_level=False,
)
# Обучение токенизатора на всём наборе текстов
tokenizer.fit_on_texts(texts_list)
# Преобразование обучающих текстов в последовательности индексов слов
seq_train = tokenizer.texts_to_sequences(text_train)
# Преобразование валидационных текстов в последовательности индексов слов
seq_test = tokenizer.texts_to_sequences(text_validate)
```

Рисунок 60. Создание токенизатора

После выполним формирование обучающей и тестовой выборок

.



```
# Формирование обучающей и тестовой выборки
x_train, y_train = vectorize_sequence(seq_train, WIN_SIZE, WIN_HOPE)
x_test, y_test = vectorize_sequence(seq_test, WIN_SIZE, WIN_HOPE)

x_train = tokenizer.sequences_to_matrix(x_train.tolist()).astype("float16")
x_test = tokenizer.sequences_to_matrix(x_test.tolist()).astype("float16")
```

Рисунок 61. Формирование обучающей и тестовой выборок

Затем преобразуем one-hot обратно в метки классов

```
# Преобразуем one-hot обратно в метки классов
train_labels = np.argmax(y_train, axis=1)
test_labels = np.argmax(y_test, axis=1)
```

Рисунок 62. Преобразование one-hot обратно в метки классов

Далее выполним подсчет с помощью bincount и произведем вывод результатов

```
# Подсчёт с помощью bincount
train_counts = np.bincount(train_labels)
test_counts = np.bincount(test_labels)

# Вывод результатов
print("Обучающая выборка:")
for cls, count in enumerate(train_counts):
    print(f"Класс {cls}: {count} окон")

print("\nТестовая выборка:")
for cls, count in enumerate(test_counts):
    print(f"Класс {cls}: {count} окон")
```

```
Обучающая выборка:
Класс 0: 107 окон
Класс 1: 166 окон

Тестовая выборка:
Класс 0: 23 окон
Класс 1: 38 окон
```

Рисунок 63. Вывод результатов

Затем создадим модель и выполним компиляцию

```
# Создание модели
model = Sequential(
    [
        Input((VOCAB_SIZE,)),
        Dense(200, activation="relu"),
        Dropout(0.4),
        Dense(100),
        Dropout(0.4),
        Dense(2, activation="softmax"),
    ]
)

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

model.summary()
```

```
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 200)	4,000,200
dropout_16 (Dropout)	(None, 200)	0
dense_25 (Dense)	(None, 100)	20,100
dropout_17 (Dropout)	(None, 100)	0
dense_26 (Dense)	(None, 2)	202

```
Total params: 4,020,502 (15.34 MB)
Trainable params: 4,020,502 (15.34 MB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 64. Создание модели

Далее выполним обучение модели

```
# Обучение модели
history = model.fit(
    x_train,
    y_train,
    epochs=150,
    batch_size=16,
    validation_data=(x_test, y_test),
    verbose=1,
)
```

```
Epoch 1/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 91ms/step - accuracy: 0.8913 - loss: 0.2424 - val_accuracy: 0.8033 - val_loss: 0.3875
Epoch 2/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 1.0000 - loss: 1.3132e-06 - val_accuracy: 0.6721 - val_loss: 0.6022
Epoch 3/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 1.0000 - loss: 5.5735e-08 - val_accuracy: 0.6721 - val_loss: 0.6491
Epoch 4/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 1.0000 - loss: 7.6660e-07 - val_accuracy: 0.6721 - val_loss: 0.6468
Epoch 5/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 1.0000 - loss: 3.0429e-07 - val_accuracy: 0.6721 - val_loss: 0.6335
Epoch 6/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 1.0000 - loss: 6.8352e-08 - val_accuracy: 0.6721 - val_loss: 0.6271
Epoch 7/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 1.0000 - loss: 2.0708e-07 - val_accuracy: 0.6721 - val_loss: 0.6226
Epoch 8/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 2.3972e-07 - val_accuracy: 0.6721 - val_loss: 0.6179
Epoch 9/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 1.0000 - loss: 1.9345e-07 - val_accuracy: 0.6885 - val_loss: 0.6000
Epoch 10/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 1.0000 - loss: 1.1960e-07 - val_accuracy: 0.7049 - val_loss: 0.5893
Epoch 11/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 1.0000 - loss: 4.9038e-08 - val_accuracy: 0.7213 - val_loss: 0.5849
Epoch 12/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 1.0000 - loss: 2.4991e-08 - val_accuracy: 0.7213 - val_loss: 0.5806
Epoch 13/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 1.0000 - loss: 5.1684e-07 - val_accuracy: 0.7213 - val_loss: 0.5737
Epoch 14/150
```

Рисунок 65. Обучение модели

После проверим точность на проверочной выборке, она составляет 0,88, что соответствует решению задания

```
[ ] loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
    print(f"Точность на проверочной выборке: {accuracy:.4f}")
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8506 - loss: 0.4316
Точность на проверочной выборке: 0.8852
```

Рисунок 66. Точность на проверочной выборке Далее построим графики процесса обучения модели

```
# Построение графиков процесса обуения модели
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
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()
```

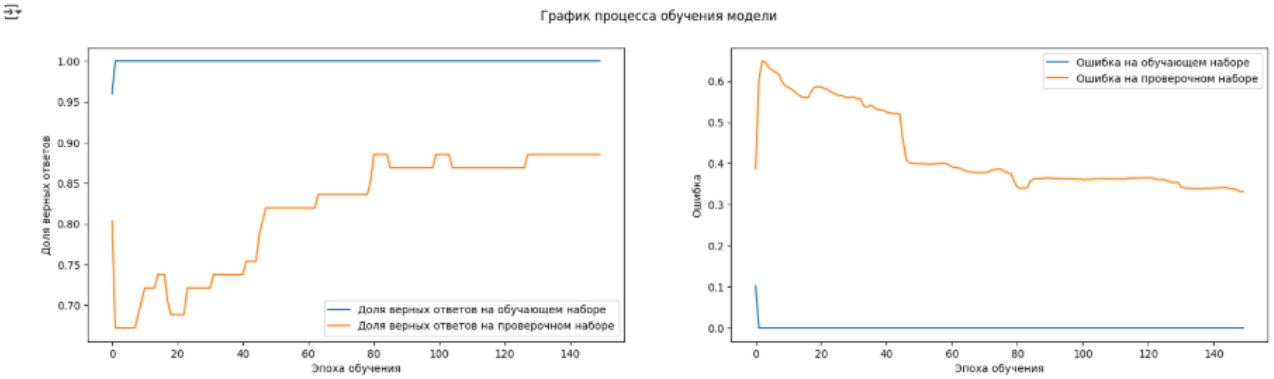


Рисунок 67. Построение графиков процесса обучения модели

Задание 3.

Условие: в данном задании занятия по обработке тектсов с помощью НС необходимо распознать уже не 6, как ранее, а целых 20 русских писателей! Это подразумевает и больший размер базы для обучения соответственно. Ячейка для скачивания базы уже включена в ноутбук задания.

В задании необходимо выполнить следующие пункты:

1. Загрузить саму базу по ссылке и подговить файлы базы для обработки.

2. Создать обучающую и проверочную выборки, обратив особое внимание на балансировку базы: количество примеров каждого класса должно быть примерно одного порядка. При этом для разбивки необходимо применить цикл. Проверочная выборка должна быть 20% от общей выборки.

3. Подготовить выборки для обучения и обучить сеть. Добиться результата точности сети не менее 95% на проверочной выборке модели Bag of Words и 75-80% - для модели Embedding.

Выполнение задания начнем с загрузки необходимых библиотек для выполнения задания

```python
# Стандартная библиотека
import time
import warnings
import zipfile  # работа с zip-архивами
from pathlib import Path

# Сторонние библиотеки
import gdown
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# TensorFlow / Keras
from tensorflow.keras import utils
from tensorflow.keras.layers import (
    Activation,
    BatchNormalization,
    Dense,
    Dropout,
    Embedding,
    Flatten,
    Input,
    SpatialDropout1D,
)
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

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

Рисунок 68. Импорт библиотек

Далее выполним загрузку архива

```python
if not Path("20writers").exists():
    # Скачивание архива с данными
    gdown.download(
        "https://storage.yandexcloud.net/aiueducation/Content/base/l7/20writers.zip",
        None,
        quiet=True,
    )
```

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

Рисунок 69. Загрузка архива

После выполним распаковку данного архива и просмотрим его содержимое

```python
# Распаковка архива
with zipfile.ZipFile("20writers.zip", "r") as zip_ref:
    zip_ref.extractall("20writers")

# Получаем отсортированный список файлов
files = sorted([p.name for p in Path("20writers").glob("*.txt")])

# Вывод в виде таблицы по 5 файлов в строке
print("Список файлов в папке 20writers:\n")
for i in range(0, len(files), 5):
    print("\t".join(files[i:i+5]))
```

```
Список файлов в папке 20writers:

Беляев.txt	Гончаров.txt	Каверин.txt	Лесков.txt	Толстой.txt
Булгаков.txt	Горький.txt	Катаев.txt	Носов.txt	Тургенев.txt
Васильев.txt	Грибоедов.txt	Куприн.txt	Пастернак.txt	Чехов.txt
Гоголь.txt	Достоевский.txt	Лермонтов.txt	Пушкин.txt	Шолохов.txt
```

Рисунок 70. Распаковка архива

Затем укажем путь к папке. И выполним проход по всем файлам/папкам внутри указанной директории

```
path = Path("20writers") # Указание пути к папке
text = []
class_names = []
# Проход по всем файлам/папкам внутри указанной директории
for f in path.iterdir():
    class_names.append(f.name)
    text.append(f.read_text().replace("\n", " "))
```

Рисунок 71. Указание пути к папке

Далее вычислим длины каждого текста в символах и выполним вывод списка длин всех текстов

```
# Вычисление длины каждого текста в символах
text_lens = [len(t) for t in text]
split_indices = [int(lenth * 0.8) for lenth in text_lens]

# Вывод списка длин всех текстов
print(text_lens)
```

```
[2328900, 2579246, 3089426, 3357061, 3104712, 1972541, 2399413, 6611627, 3386268, 3408339, 2001064, 2152214, 1980206, 5178950, 2952792, 969109, 2255254, 1965119, 2523380, 1992830]
```

Рисунок 72. Вывод списка длин всех текстов

Затем выполним создание списков для хранения обучающих и текстовых фрагментов и построим гистограмму

```
train_text = [] # Список для хранения обучающих фрагментов текстов
test_text = [] # Список для хранения тестовых (валидационных) фрагментов текстов

# Разделение каждого текста на обучающую и тестовую части
for i, t in enumerate(text):
    train_text.append(t[:split_indices[i]])
    test_text.append(t[split_indices[i]:])

# Проверка сбалансированности выборок по классам
fig = plt.figure(figsize=(10, 5))
c_train = split_indices
c_test = [i-j for i, j in zip(text_lens, split_indices)]
plt.bar(class_names, c_train)
plt.bar(class_names, c_test)
fig.autofmt_xdate(rotation=45)
plt.show()
```

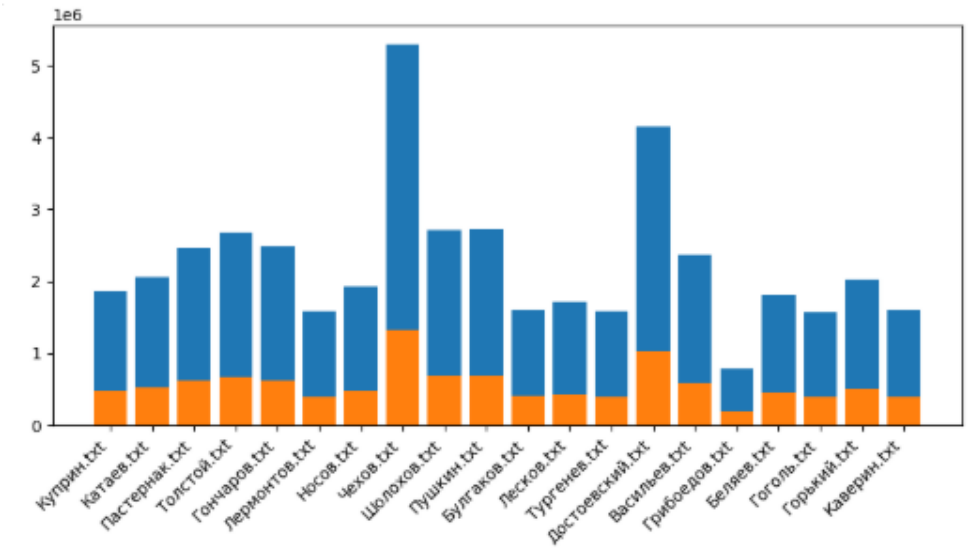


Рисунок 73. Построение гистограммы

Далее зададим параметры

```
VOCAB_SIZE = 25_000
WIN_SIZE = 4000
WIN_HOPE = 500
```

Рисунок 74. Задание параметров

Затем напишем функцию разбиения последовательности на отрезки скользящим окном и функцию формирования выборок из последовательностей индексов

```
# Функция разбиения последовательности на отрезки скользящим окном
# На входе - последовательность индексов, размер окна, шаг окна
def split_sequence(sequence, win_size, hop):
    # Последовательность разбивается на части до последнего полного окна
    return [
        sequence[i : i + win_size] for i in range(0, len(sequence) - win_size + 1, hop)
    ]


# Функция формирования выборок из последовательностей индексов
# формирует выборку отрезков и соответствующих им меток классов в виде one hot encoding
def vectorize_sequence(seq_list, win_size, hop):
    # В списке последовательности следуют в порядке их классов
    # Всего последовательностей в списке ровно столько, сколько классов
    class_count = len(seq_list)

    # Списки для исходных векторов и категориальных меток класса
    x, y = [], []

    # Для каждого класса:
    for cls in range(class_count):
        # Разбиение последовательности класса cls на отрезки
        vectors = split_sequence(seq_list[cls], win_size, hop)
        # Добавление отрезков в выборку
        x += vectors
        # Для всех отрезков класса cls добавление меток класса в виде ОНЕ
        y += [utils.to_categorical(cls, class_count)] * len(vectors)

    # Возврат результатов как numpy-массивов
    return np.array(x), np.array(y)
```

Рисунок 75. Написание функций

Далее напишем контекстный менеджер для измерения времени выполнения блока кода

```
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 76. Контекстный менеджер для измерения времени

Затем создадим токенизатор и выполним формирование обучающей и тестовой выборки

```
with timex():
    tokenizer = Tokenizer(
        num_words=VOCAB_SIZE,
        filters='!"#$%&()*+,-—./…:;<=>?@[\\]^_`{|}~«»\t\n\xa0\ufeff',
        lower=True,
        split=" ",
        oov_token="неизвестное_слово",
        char_level=False,
    )
    tokenizer.fit_on_texts(text)
    seq_train = tokenizer.texts_to_sequences(train_text)
    seq_test = tokenizer.texts_to_sequences(test_text)

    # Формирование обучающей и тестовой выборки
    x_train, y_train = vectorize_sequence(seq_train, WIN_SIZE, WIN_HOPE)
    x_test, y_test = vectorize_sequence(seq_test, WIN_SIZE, WIN_HOPE)
```

```
Время обработки: 27.20 с
```

Рисунок 77. Формирование обучающей и тестовой выборки

Выполним замер времени преобразования обучающих и тестовых последовательностей в матрицу признаков

```
# Замер времени преобразования обучающих последовательностей в матрицу признаков
with timex():
    x_train_matrix = tokenizer.sequences_to_matrix(x_train.tolist()).astype("float16")

# Замер времени преобразования тестовых (валидационных) последовательностей в матрицу признаков
with timex():
    x_test_matrix = tokenizer.sequences_to_matrix(x_test.tolist()).astype("float16")
```

```
Время обработки: 19.56 с
Время обработки: 4.13 с
```

Рисунок 78. Замер времени

После выполним преобразование one-hot обратно в метки классов

```
# Преобразуем one-hot обратно в метки классов
train_labels = np.argmax(y_train, axis=1)
test_labels = np.argmax(y_test, axis=1)
```

Рисунок 79. Преобразуем one-hot обратно в метки классов Далее выполним подсчет с помощью bincount и произведем вывод результатов

```
# Подсчёт с помощью bincount
train_counts = np.bincount(train_labels)
test_counts = np.bincount(test_labels)

# Вывод результатов
print("Обучающая выборка:")
for cls, count in enumerate(train_counts):
    print(f"Класс {cls}: {count} окон")

print("\nТестовая выборка:")
for cls, count in enumerate(test_counts):
    print(f"Класс {cls}: {count} окон")
```

Рисунок 80. Вывод результатов

```
Обучающая выборка:
Класс 0: 558 окон
Класс 1: 602 окон
Класс 2: 726 окон
Класс 3: 829 окон
Класс 4: 780 окон
Класс 5: 488 окон
Класс 6: 586 окон
Класс 7: 1656 окон
Класс 8: 791 окон
Класс 9: 823 окон
Класс 10: 470 окон
Класс 11: 537 окон
Класс 12: 484 окон
Класс 13: 1294 окон
Класс 14: 706 окон
Класс 15: 239 окон
Класс 16: 532 окон
Класс 17: 479 окон
Класс 18: 647 окон
Класс 19: 479 окон

Тестовая выборка:
Класс 0: 134 окон
Класс 1: 143 окон
Класс 2: 171 окон
Класс 3: 206 окон
Класс 4: 188 окон
Класс 5: 117 окон
Класс 6: 150 окон
Класс 7: 422 окон
Класс 8: 201 окон
Класс 9: 194 окон
Класс 10: 111 окон
Класс 11: 121 окон
Класс 12: 116 окон
Класс 13: 327 окон
Класс 14: 169 окон
Класс 15: 52 окон
Класс 16: 127 окон
Класс 17: 115 окон
Класс 18: 153 окон
Класс 19: 115 окон
```

Рисунок 81. Результат вывода

Далее выполним вычисление весов классов для компенсации дисбаланса классов в обучающей выборке

```python
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Вычисление весов классов для компенсации дисбаланса классов в обучающей выборке
class_weights = compute_class_weight(
    "balanced",
    classes=np.unique(np.argmax(y_train, axis=1)),
    y=np.argmax(y_train, axis=1),
)
# Преобразование массива весов в словарь
class_weight_dict = dict(enumerate(class_weights))
```

Рисунок 82. Преобразование массива весов в словарь Затем выполним создание модели и ее компиляцию

```python
# Создание модели
model = Sequential(
    [
        Input((VOCAB_SIZE,)),
        Dropout(0.6),
        Dense(64, activation="relu"),
        Dropout(0.6),
        Dense(20, activation="softmax"),
    ]
)

model.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dropout (Dropout)	(None, 25000)	0
dense (Dense)	(None, 64)	1,600,064
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 20)	1,300

```
Total params: 1,601,364 (6.11 MB)
Trainable params: 1,601,364 (6.11 MB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 83. Создание модели

Далее выполним обучение созданной модели

```python
# Обучение модели
history = model.fit(
    x_train_matrix,
    y_train,
    epochs=15,
    batch_size=256,
    validation_data=(x_test_matrix, y_test),
    class_weight=class_weight_dict,
    verbose=1,
)
```

```
Epoch 1/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 8s 104ms/step - accuracy: 0.2623 - loss: 2.4332 - val_accuracy: 0.9619 - val_loss: 0.5547
Epoch 2/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.8812 - loss: 0.4484 - val_accuracy: 0.9715 - val_loss: 0.2520
Epoch 3/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9518 - loss: 0.2030 - val_accuracy: 0.9727 - val_loss: 0.1845
Epoch 4/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9645 - loss: 0.1414 - val_accuracy: 0.9742 - val_loss: 0.1493
Epoch 5/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9754 - loss: 0.1075 - val_accuracy: 0.9736 - val_loss: 0.1304
Epoch 6/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.9795 - loss: 0.0895 - val_accuracy: 0.9730 - val_loss: 0.1220
Epoch 7/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9794 - loss: 0.0797 - val_accuracy: 0.9742 - val_loss: 0.1111
Epoch 8/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9798 - loss: 0.0677 - val_accuracy: 0.9751 - val_loss: 0.1090
Epoch 9/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.9820 - loss: 0.0617 - val_accuracy: 0.9754 - val_loss: 0.1017
```

Рисунок 84. Создание модели

После обучение построим графики процесса обучения модели

```python
# Построение графиков процесса обучения модели
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
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()
```

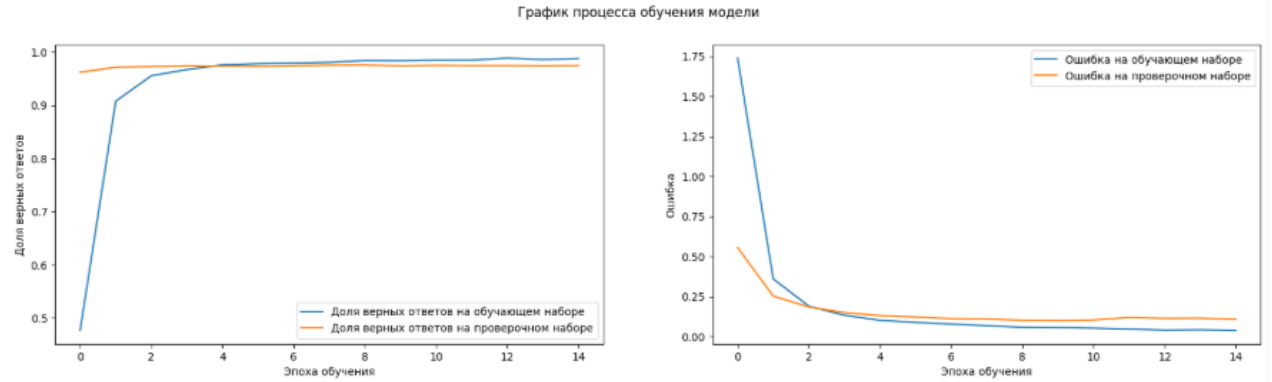


Рисунок 85. Графики процесса обучения модели

Далее напишем сервисные функции необходимые для построения матрицы ошибок

```python
def eval_model(
    model, x, y_true, class_labels=[], cm_round=3, title="", figsize=(10, 10)
):
    # Вычисление предсказания сети
    y_pred = model.predict(x)
    # Построение матрицы ошибок
    cm = confusion_matrix(
        np.argmax(y_true, axis=1), np.argmax(y_pred, axis=1), normalize="true"
    )
    # Округление значений матрицы ошибок
    cm = np.around(cm, cm_round)

    # Отрисовка матрицы ошибок
    fig, ax = plt.subplots(figsize=figsize)
    ax.set_title(f"Нейросеть {title}: матрица ошибок нормализованная", fontsize=18)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
    disp.plot(ax=ax)
    plt.gca().images[-1].colorbar.remove()  # Стирание ненужной цветовой шкалы
    plt.xlabel("Предсказанные классы", fontsize=16)
    plt.ylabel("Верные классы", fontsize=16)
    fig.autofmt_xdate(rotation=45)  # Наклон меток горизонтальной оси при необходимости
    plt.show()

    print("-" * 100)
    print(f"Нейросеть: {title}")

    # Для каждого класса:
    for cls in range(len(class_labels)):
        # Определяется индекс класса с максимальным значением предсказания (уверенности)
        cls_pred = np.argmax(cm[cls])
        # Формируется сообщение о верности или неверности предсказания
        msg = "ВЕРНО :-)" if cls_pred == cls else "НЕВЕРНО :-("
        # Выводится текстовая информация о предсказанном классе и значении уверенности
        print(
            "Класс: {:<20} {:3.0f}% сеть отнесла к классу {:<20} - {}".format(
                class_labels[cls],
                100.0 * cm[cls, cls_pred],
                class_labels[cls_pred],
                msg,
            )
        )

    # Средняя точность распознавания определяется как среднее диагональных элементов матрицы ошибок
    print(
        "\nСредняя точность распознавания: {:3.0f}%".format(
            100.0 * cm.diagonal().mean()
        )
    )
```

Рисунок 86. Сервисные функции

После сделаем вывод матрицы ошибок

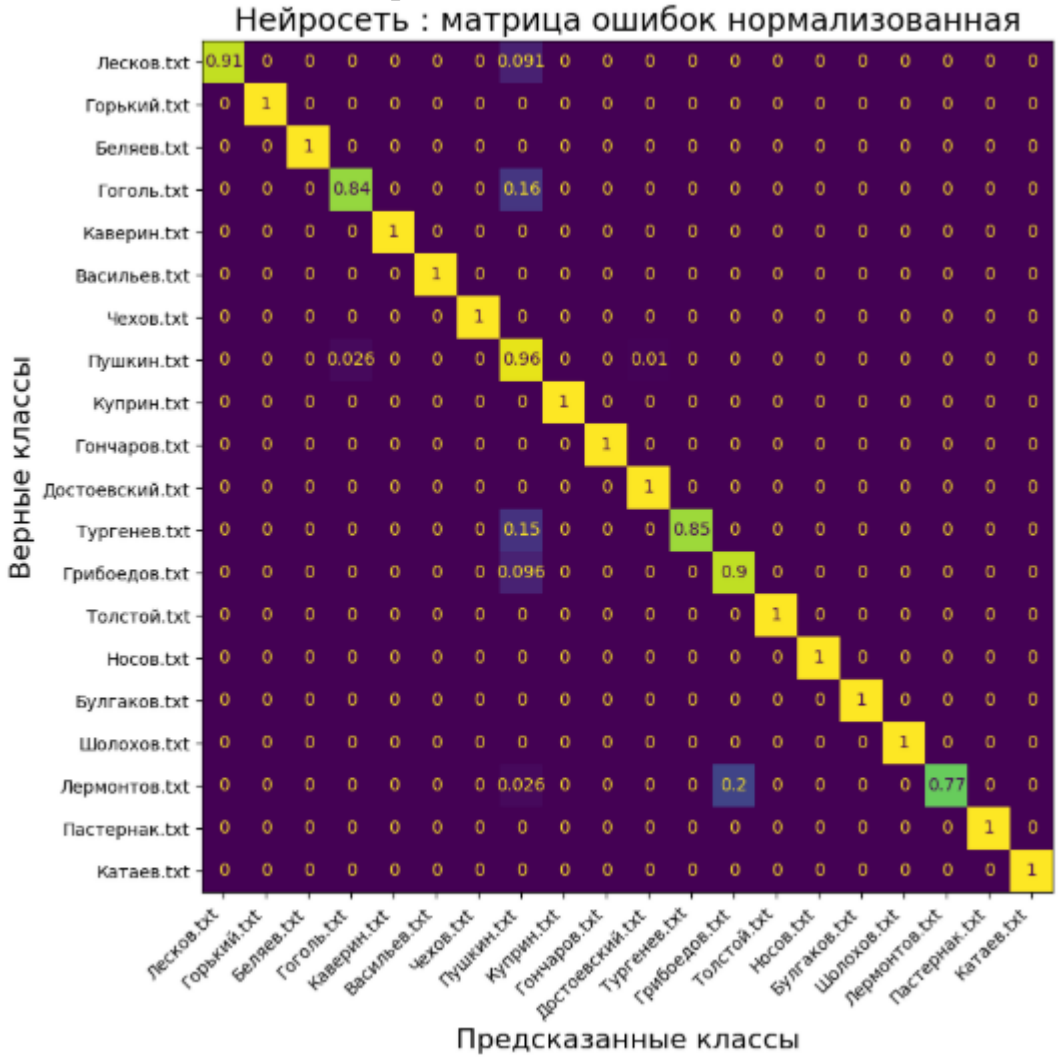


Рисунок 87. Матрица ошибок

```
----------------------------------------------------------------------------------------------------
Нейросеть:
Класс: Лесков.txt          91% сеть отнесла к классу Лесков.txt          - ВЕРНО :-)
Класс: Горький.txt        100% сеть отнесла к классу Горький.txt        - ВЕРНО :-)
Класс: Беляев.txt         100% сеть отнесла к классу Беляев.txt         - ВЕРНО :-)
Класс: Гоголь.txt          84% сеть отнесла к классу Гоголь.txt         - ВЕРНО :-)
Класс: Каверин.txt        100% сеть отнесла к классу Каверин.txt        - ВЕРНО :-)
Класс: Васильев.txt       100% сеть отнесла к классу Васильев.txt       - ВЕРНО :-)
Класс: Чехов.txt          100% сеть отнесла к классу Чехов.txt          - ВЕРНО :-)
Класс: Пушкин.txt          96% сеть отнесла к классу Пушкин.txt         - ВЕРНО :-)
Класс: Куприн.txt         100% сеть отнесла к классу Куприн.txt         - ВЕРНО :-)
Класс: Гончаров.txt       100% сеть отнесла к классу Гончаров.txt       - ВЕРНО :-)
Класс: Достоевский.txt    100% сеть отнесла к классу Достоевский.txt    - ВЕРНО :-)
Класс: Тургенев.txt        85% сеть отнесла к классу Тургенев.txt       - ВЕРНО :-)
Класс: Грибоедов.txt       90% сеть отнесла к классу Грибоедов.txt      - ВЕРНО :-)
Класс: Толстой.txt        100% сеть отнесла к классу Толстой.txt        - ВЕРНО :-)
Класс: Носов.txt          100% сеть отнесла к классу Носов.txt          - ВЕРНО :-)
Класс: Булгаков.txt       100% сеть отнесла к классу Булгаков.txt       - ВЕРНО :-)
Класс: Шолохов.txt        100% сеть отнесла к классу Шолохов.txt        - ВЕРНО :-)
Класс: Лермонтов.txt       77% сеть отнесла к классу Лермонтов.txt      - ВЕРНО :-)
Класс: Пастернак.txt      100% сеть отнесла к классу Пастернак.txt      - ВЕРНО :-)
Класс: Катаев.txt         100% сеть отнесла к классу Катаев.txt         - ВЕРНО :-)

Средняя точность распознавания:  96%
```

Рисунок 88. Средняя точность распознания

Отсюда видно, что точность матрицы ошибок составила 96%, что входит в диапазон выполнения задания, следовательно задание выполнено.

Далее выполним создание модели Embedding и выполним ее компиляцию.

```python
# Создание модели
model_emb = Sequential(
    [
        Embedding(VOCAB_SIZE, 400, input_length=WIN_SIZE),
        SpatialDropout1D(0.4),
        Flatten(),
        Dense(64, activation="relu"),
        Dropout(0.4),
        Dense(20, activation="softmax"),
    ]
)
model_emb.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

model_emb.build(input_shape=(None, WIN_SIZE))
model_emb.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 4000, 400)	10,000,000
spatial_dropout1d (SpatialDropout1D)	(None, 4000, 400)	0
flatten (Flatten)	(None, 1600000)	0
dense (Dense)	(None, 64)	102,400,064
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 20)	1,300

Total params: 112,401,364 (428.78 MB)
Trainable params: 112,401,364 (428.78 MB)
Non-trainable params: 0 (0.00 B)

Рисунок 89. Создание модели Затем выполним обучение модели

```python
history_emb = model_emb.fit(
    x_train,
    y_train,
    epochs=15,
    batch_size=256,
    validation_data=(x_test, y_test),
    class_weight=class_weight_dict,
    verbose=1,
)
```

```
Epoch 1/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 24s 317ms/step - accuracy: 0.1168 - loss: 2.8283 - val_accuracy: 0.4622 - val_loss: 1.8850
Epoch 2/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 122ms/step - accuracy: 0.8684 - loss: 0.4831 - val_accuracy: 0.7773 - val_loss: 0.9205
Epoch 3/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 6s 118ms/step - accuracy: 0.9907 - loss: 0.0613 - val_accuracy: 0.7992 - val_loss: 0.7741
Epoch 4/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 119ms/step - accuracy: 0.9952 - loss: 0.0326 - val_accuracy: 0.8166 - val_loss: 0.7079
Epoch 5/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 123ms/step - accuracy: 0.9971 - loss: 0.0207 - val_accuracy: 0.8076 - val_loss: 0.6795
Epoch 6/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 123ms/step - accuracy: 0.9982 - loss: 0.0157 - val_accuracy: 0.8124 - val_loss: 0.6565
Epoch 7/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 124ms/step - accuracy: 0.9985 - loss: 0.0116 - val_accuracy: 0.8148 - val_loss: 0.6413
Epoch 8/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 121ms/step - accuracy: 0.9987 - loss: 0.0100 - val_accuracy: 0.8115 - val_loss: 0.6226
Epoch 9/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 122ms/step - accuracy: 0.9990 - loss: 0.0084 - val_accuracy: 0.8073 - val_loss: 0.6474
Epoch 10/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 123ms/step - accuracy: 0.9994 - loss: 0.0068 - val_accuracy: 0.7995 - val_loss: 0.6681
Epoch 11/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 123ms/step - accuracy: 0.9999 - loss: 0.0058 - val_accuracy: 0.8148 - val_loss: 0.6043
Epoch 12/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 124ms/step - accuracy: 0.9997 - loss: 0.0054 - val_accuracy: 0.8343 - val_loss: 0.5495
Epoch 13/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 7s 125ms/step - accuracy: 0.9993 - loss: 0.0052 - val_accuracy: 0.8280 - val_loss: 0.5567
Epoch 14/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 129ms/step - accuracy: 0.9997 - loss: 0.0032 - val_accuracy: 0.8370 - val_loss: 0.5765
Epoch 15/15
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 125ms/step - accuracy: 0.9996 - loss: 0.0044 - val_accuracy: 0.8358 - val_loss: 0.5392
```

Рисунок 90. Обучение модели

Далее выполним построение графиков процесса обучения модели

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
fig.suptitle("График процесса обучения модели с Embedding")
ax1.plot(history_emb.history["accuracy"], label="Доля верных ответов на обучающем наборе")
ax1.plot(
    history_emb.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_emb.history["loss"], label="Ошибка на обучающем наборе")
ax2.plot(history_emb.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()
```

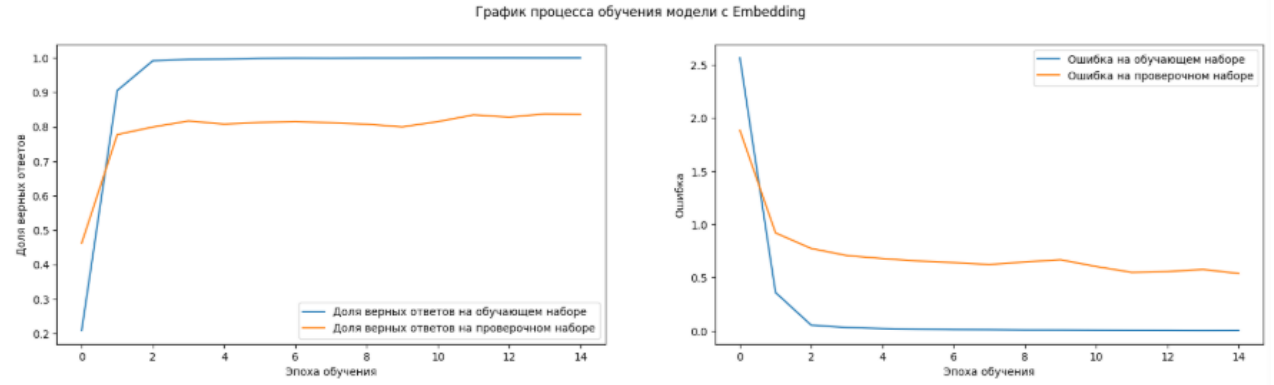


Рисунок 91. Графики процесса обучения модели После выполним построение матрицы ошибок

```
eval_model(model_emb, x_test, y_test, class_names)
```

```
105/105 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step
```

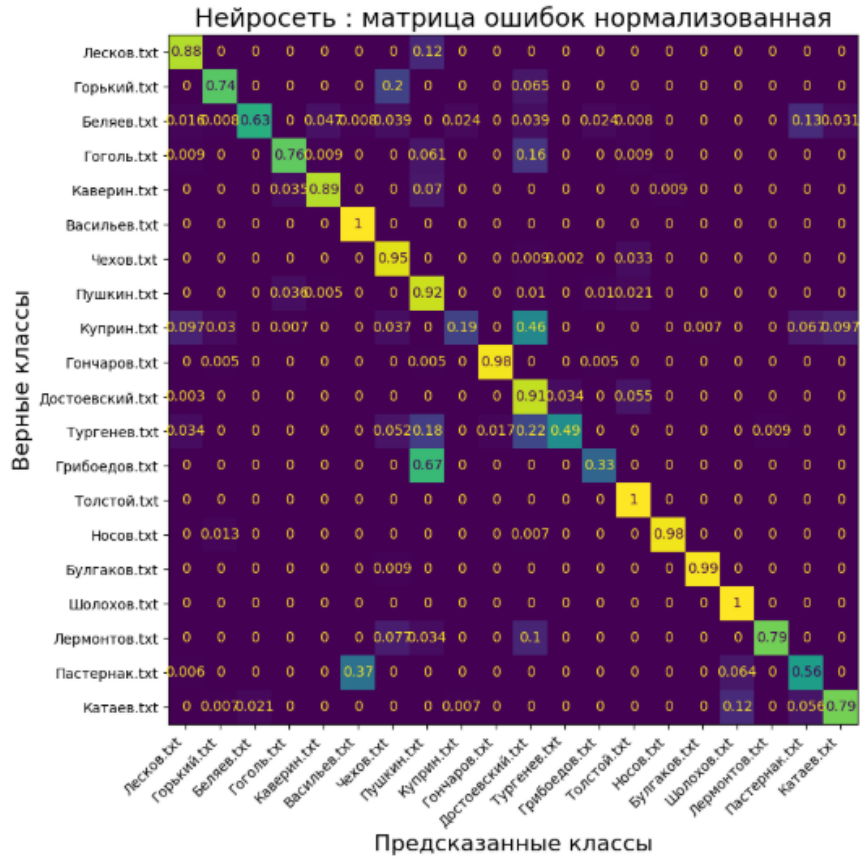


Рисунок 92. Матрица ошибок

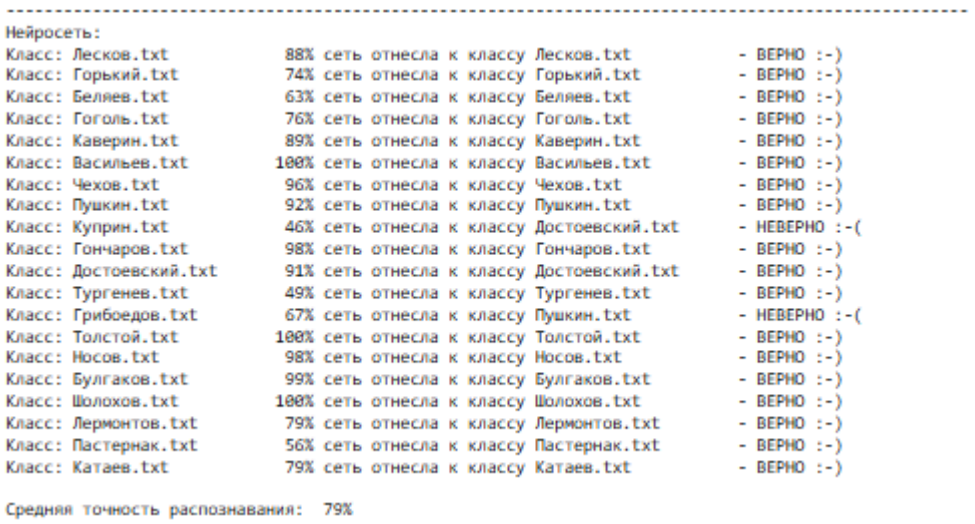


Рисунок 93. Средняя точность распознания

Отсюда видно, что точность матрицы ошибок составила 79%, что входит в диапазон выполнения задания, следовательно задание выполнено.

Вывод: в ходе выполнения данной лабораторной работы были изучены особенности построения архитектур нейронных сетей для обработки текста и особенности обучения таких НС. Были изучены способы построения словаря частотности, особенности подготовки несбалансированной базы для обучения сети, использования форматов, которые подаются в нейронную сеть, использование формата «Bag of Words», а также использование слоя «Embedding» в моделях сетей.