1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. **Кафедра «Высшая школа кибербезопасности»**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. **Использование Python-библиотек для работы с ИИ**
2. по дисциплине «Цифровая культура»
3. Выполнили Мальцев М.А.
4. студенты гр. 5131001/30003 Шевчук Н.Е.

<*подпись*>

Руководитель

ассистент ВШК ИКНК Иванов М.С.

<*подпись*>

1. Санкт-Петербург
2. 2025

**Оглавление**

[1 Цель работы 3](#_Toc196837588)

[2 Задачи 3](#_Toc196837589)

[2.1 Реализация классификатора 3](#_Toc196837590)

[2.2 Реализация Telegram-бота 4](#_Toc196837591)

[2.3 Реализация объединения 5](#_Toc196837592)

[3 Ход работы 5](#_Toc196837593)

[3.1 Реализация классификатора 5](#_Toc196837594)

[3.1.1 Изучение исходного кода lab3.py 5](#_Toc196837595)

[3.1.2 Формирование датасетов 8](#_Toc196837596)

[3.1.3 Модификация кода под соответствующий вариант 8](#_Toc196837597)

[3.1.4 Тестирование программы 9](#_Toc196837598)

[3.2 Реализация Telegram-бота 11](#_Toc196837599)

[3.2.1 Описание решения 11](#_Toc196837600)

[3.2.2 Тестирование итогового функционала 13](#_Toc196837601)

[4 Ответы на контрольные вопросы 14](#_Toc196837602)

[5 Вывод 17](#_Toc196837603)

[Приложение 1 18](#_Toc196837604)

[Приложение 2 21](#_Toc196837605)

[Приложение 3 27](#_Toc196837606)

# Цель работы

Получить навыки работы с методами ИИ для решения задачи классификации данных с использованием языка программирования Python.

# Задачи

## Реализация классификатора

1. Изучить исходный код выданного скрипта lab3.py. Изучить используемые в скрипте библиотеки.
2. Скачать предоставленные в задании наборы данных для проверки работоспособности скрипта.
3. Сформировать датасеты для обучения и тестирования (валидации). Набор данных должен представлять собой набор изображений. Первый набор изображений – люди, второй – животное в зависимости от варианта (вариант соответствует порядковому номеру в списке группы). Для формирования датасетов может быть использован сервис «Hugging Face», либо любые другие открытые источники.
4. Если в обучающей выборке недостаточно изображений – уменьшите значение batch\_size при вызове функции train\_datagen.flow\_from\_directory.
5. Модифицировать скрипт в соответствии с требованиями:
   * на вход скрипту подаются параметры: путь до тренировочного датасета, путь до датасета для тестирования (валидации), путь до файла, который необходимо классифицировать;
   * вывод скрипта – результат бинарной классификации (что изображено на картинке: человек или животное).
6. Протестировать работу программы на изображениях:
7. Из обучающего набора данных.
8. Из тестового набора данных.
9. Из произвольного источника.
10. Провести эксперименты по изменению параметров модели классификации для повышения эффективности. Сделать соответствующие выводы о работе скрипта.
11. Сохранить обученную модель с наилучшим результатом классификации в отдельный файл.

## Реализация Telegram-бота

1. Установить необходимые для работы скрипта модули. Установить модуль telebot для работы с Telegram-ботом.
2. Изучить функционал модуля telebot.
3. Получить токен для работы с ботом путем обращения к боту @botfather.
4. Реализовать собственный Telegram-бот, который поддерживает следующие команды:

* /register – команда для регистрации пользователя; после отправки данной команды пользователю необходимо ввести пароль для успешной регистрации.
* /login – команда для прохождения аутентификации; после отправки данной команды пользователю необходимо ввести пароль, введенный при регистрации, для дальнейшего использования команды /predict;
* /predict – команда для проведения бинарной классификации картинки.
* /logout – команда для выхода из системы; после отправки данной команды пользователь не может пользоваться командой /predict до тех пор, пока заново не пройдет аутентификацию с помощью /login.

1. Важно, чтобы только зарегистрированный пользователь, прошедший аутентификацию, мог использовать команду /predict.
2. Один из вариантов реализации отслеживания состояний пользователей – это хранение следующих параметров: ID чата, пароль, флаг (осуществлен вход или нет).

## Реализация объединения

Необходимо чтобы отправки данной команды /predict пользователь отправил боту картинку; в ответ бот присылает ответ – на картинке изображен человек или животное; для классификации следует использовать файл модели, полученной на шаге 8 из реализации классификатора (при запуске команды predict не должно происходить повторное обучение модели).

# Ход работы

## Реализация классификатора

### Изучение исходного кода lab3.py

Скрипт предназначен для классификации изображений (Трава/Одуванчики) с использованием нейронной сети, построенной на TensorFlow/Keras, с предварительной обработкой данных и визуализацией.

Скрипт выполняет следующие задачи:

* Загрузка и организация данных: Изображения из папок train/Dandelion, train/Grass, valid/Dandelion, valid/Grass.
* Визуализация образцов: Показывает несколько изображений из обучающей выборки.
* Предобработка данных: Использует ImageDataGenerator для подготовки изображений.
* Создание и обучение модели: Простая нейронная сеть для бинарной классификации.
* Тестирование модели: Прогнозирование класса (Одуванчик или Трава) для одного тестового изображения (test.jpg).

В скрипте используются следующие библиотеки:

* **os** - предоставляет функции для работы с операционной системой, такие как доступ к файлам и папкам. В скрипте используется для работы с директориями и файлами, например, os.listdir (список файлов) и os.path.join (создание путей).
* **numpy as np** - библиотека для работы с многомерными массивами и математическими операциями. В скрипте используется для обработки массивов данных изображений, например, np.expand\_dims и np.vstack
* **tensorflow as tf -** фреймворк для машинного обучения, особенно для создания и обучения нейронных сетей. В скрипте используется как основная библиотека для создания, компиляции и обучения модели tf.keras.
* **matplotlib.pyplot as plt** - библиотека для создания графиков и визуализаций. В скрипте используется для отображения изображений из датасета и тестового изображения.
* **matplotlib.image as mpimg** - модуль Matplotlib для чтения и обработки изображений. В скрипте используется для загрузки изображений (mpimg.imread) для визуализации.
* **scipy.interp -** функция для интерполяции данных.
* **itertools.cycle** - создает итераторы для циклического перебора элементов.
* Модули из **sklearn**:

1. sklearn.svm: Алгоритмы метода опорных векторов (не используются);
2. sklearn.datasets: Наборы данных для тестирования (не используются);
3. sklearn.metrics.roc\_auc\_score, roc\_curve, auc: Метрики для оценки качества классификации (ROC-кривые и площадь под кривой);
4. sklearn.preprocessing.label\_binarize: Преобразование меток в бинарный формат для многоклассовой классификации;
5. sklearn.multiclass.OneVsRestClassifier: Стратегия для многоклассовой классификации (не используется);
6. sklearn.model\_selection.train\_test\_split: Разделение данных на обучающую и тестовую выборки (не используется);

* **tensorflow.keras.preprocessing.image -** инструменты для предобработки изображений (загрузка, преобразование в массивы). В скрипте используется для загрузки тестового изображения (image.load\_img, image.img\_to\_array).
* **tensorflow.keras.preprocessing.image.ImageDataGenerator** - генерация батчей (набор, обрабатывающийся одновременно) изображений с предобработкой и аугментацией. В скрипте используется для подготовки обучающих и валидационных данных (flow\_from\_directory).

С листингом исходного кода с подробными комментариями можно ознакомиться в приложении 1. На рисунке 1 представлено изображение, использующаяся при тестировании исходного кода. На рисунке 2представлен конечный вывод программы.



Рисунок 1 - Изображение для тестового запуска исходного кода

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 - Результат тестового запуска

### Формирование датасетов

Были сформированы датасеты с jpg-файлами: люди и жирафы, в соответствии с вариантом. Датасеты были помещены в папки train и valid. На рисунке 3 представлена статистика по количеству картинок в датасетах.

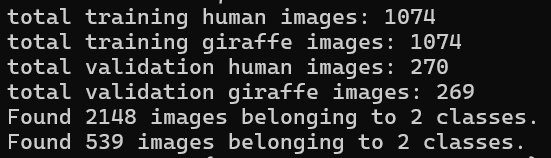


Рисунок 3 - Количество изображений в датасетах

### Модификация кода под соответствующий вариант

Далее, исходный код был модифицирован следующим образом:

* + на вход скрипту подаются параметры: путь до тренировочного датасета, путь до датасета для тестирования (валидации), путь до файла, который необходимо классифицировать;
  + вывод скрипта – график зависимости точности распознавания от эпохи и график зависимости ошибок модели при обучении от эпохи; результат бинарной классификации (что изображено на картинке: человек или животное).
  + Для повышения точности были использованы сверточные слои для извлечения визуальных особенностей. Параметр batch\_size был изменён на 32, параметры steps\_por\_epoch и validation\_steps используют половину генератора, чтобы избежать переобучения.

На рисунке 4 представлен график зависимости точности распознавания от эпохи и график зависимости ошибок модели при обучении от эпохи.

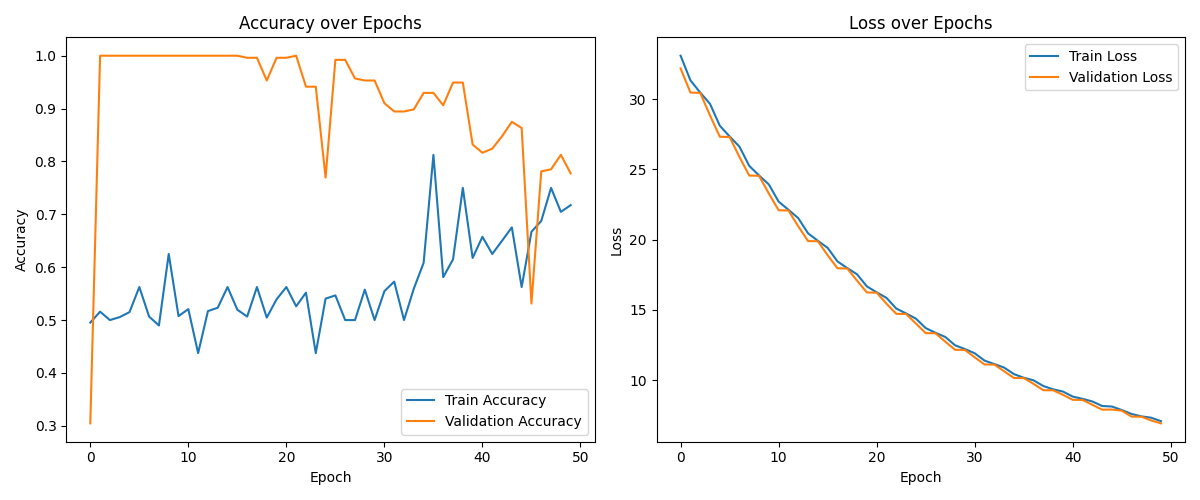


Рисунок 4 - Графики, демонстрирующие процесс обучения модели

Accuracy (точность) – это метрика, которая характеризует качество модели. Accuracy показывает долю правильно предсказанных классов ко всем предсказаниям и вычисляется по следующей формуле:

где TP (True Positive) – ожидаемо правильные предсказания, TN (True Negative) – ожидаемо неправильные предсказания, FP (False Positive) – предсказания, являющиеся ошибками второго рода, FN (False Negative) – предсказания, являющиеся ошибками первого рода.

По данным графиков видно, что вероятность ошибки значительно снижается по мере обучения модели и во время последних эпох принимает пренебрежимо малое значение.

С полным листингом реализованного кода можно ознакомиться в приложении 2.

### Тестирование программы

Пример команды запуска программы представлен на рисунке 5.

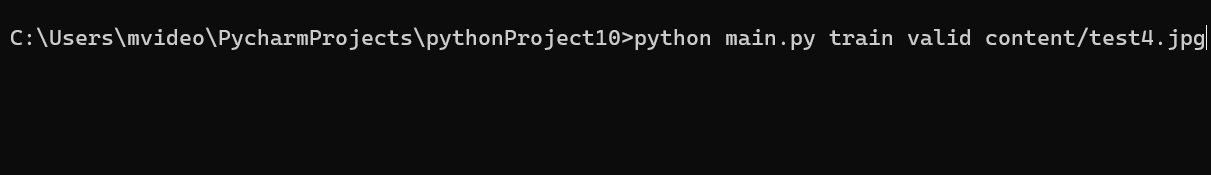


Рисунок 5 - Пример команды запуска программы

По результатам тестирования на различных входных тестовых файлах – из обучающего набора данных, из тестового набора данных и из произвольного источника, - был сделан вывод, что программа корректно классифицирует человек или жираф представлен на тестовом изображении. На рисунке 6 представлено тестовое изображение, подаваемое программе на вход, при одном из тестовых запусков. На рисунке 7 можно ознакомиться с результатом работы программы. Также по графикам, приведенным в пункте выше, видно, что модель корректно обучается, вследствие чего вероятность ошибки сводится к минимуму.



Рисунок 6 - Пример тестового изображения

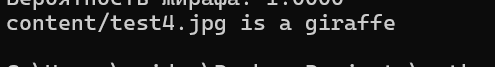


Рисунок 7 - Результат работы классификатора

## Реализация Telegram-бота

### Описание решения

В начале работы над реализацией Telegram-бота был установлен модуль telebot, необходимый для работы бота.

Также были изучены основы работы с библиотекой для создания Telegram-ботов pyTelegramBotAPI.

Был реализован и протестирован простейший код, выводящий приветствие, представленный в листинге 1.

Листинг - Код простейшего бота

|  |
| --- |
| import telebot  bot = telebot.TeleBot("TOKEN", parse\_mode=None)  @bot.message\_handler(commands=['start'])  def send\_welcome(message):  bot.reply\_to(message, "Hello!")  bot.infinity\_polling() |

Затем для реализации необходимых команд были добавлены функции:

1. register\_user, process\_password\_step – программа запрашивает у пользователя желаемый пароль и в случае соответствия вносит его данные в список registered\_users в формате [ID пользователя, пароль, флаг входа].
2. login\_user, process\_login\_step – запрашивает у пользователя пароль и в случае правильности введённого пароля, помечает пользователя как вошедшего, устанавливая флагу входа значение 1.
3. predict\_image, process\_image\_step –запрашивает у пользователя картинку и затем, определив содержание с помощью классификатора, отправляет пользователю полученный результат (человек или жираф).
4. logout\_user –помечает пользователя, запросившего выход, как не залогиненного, устанавливая флагу входа значение 0.
5. help – программа предоставляет краткое описание команд.

Вышеописанный функционал был протестирован и были устранены некоторые неучтённые детали. Например, ситуация, когда зарегистрированный пользователь повторно вводит команду /register и т. д.. Пример регистрации пользователя представлен на рисунке 8.



Рисунок 8 - Пример регистрации нового пользователя

Впоследствии место хранения данных пользователей было изменено с массива на файл users.csv, в котором данные имеют вид {user\_id},{hashed\_password}. Был создан список signed\_in\_users для хранения идентификаторов залогиненных пользователей. Таким образом, необходимо осуществлять вход при каждом перезапуске бота.

### Тестирование итогового функционала

Для реализации команды /predict в код Telegram-бота был объединён с кодом классификатора из данной работы. Примеры распознавания объекта на изображении с помощью команды /predict представлены на рисунках 9 и 10.



Рисунок 9 - Пример распознавания жирафа



Рисунок 10 - Пример распознавания человека

# Ответы на контрольные вопросы

1. *Какие классы задач могут быть решены с помощью методов искусственного интеллекта?*

С помощью методов искусственного интеллекта могут решаться такие классы задач, как:

1) распознавание образов (компьютерное зрение, обработка изображений и видео);

2) обработка естественного языка (NLP – чат-боты, перевод, анализ тональности);

3) прогнозирование и анализ данных (машинное обучение в финансах, медицине, логистике);

4) принятие решений (экспертные системы, рекомендательные алгоритмы). Автоматизация и робототехника (управление автономными системами);

5) генерация контента (тексты, изображения, музыка – GPT, DALL-E, Stable Diffusion). Кибербезопасность (анализ угроз, обнаружение аномалий);

6) оптимизация процессов (управление ресурсами, маршрутизация).

1. *Чем отличается обучающий набор данных от тестового?*

Обучающий набор используется для обучения модели (настройка её параметров), а тестовый — для проверки её качества на новых данных, которые модель не видела. Обучающий набор обычно больше, а тестовый — независимый, чтобы оценить обобщающую способность модели.

1. *Что такое признак в контексте методов искусственного интеллекта? Что такое метка в контексте методов искусственного интеллекта? В чем их разница?*

Признак — это характеристика данных, используемая для обучения (например, размер изображения, цвет пикселя). Метка — это целевое значение, которое модель должна предсказать (например, класс "человек" или "жираф"). Признаки — это входные данные, а метка — ожидаемый выход.

1. *Чем методы глубокого обучения отличаются от других методов искусственного интеллекта?*

Глубокое обучение использует многослойные нейронные сети для автоматического извлечения признаков из данных, тогда как другие методы ИИ (например, деревья решений) обычно требуют предварительной обработки данных, где признаки либо выбираются вручную человеком (например, длина объекта, средняя яркость изображения), либо извлекаются с помощью заранее заданных алгоритмов. Это означает, что в традиционных методах ИИ процесс извлечения признаков часто зависит от экспертных знаний или заранее определённых правил, а не происходит автоматически, как в глубоком обучении. Метод эффективен для больших данных (изображения, текст), но требует больше вычислительных ресурсов.

1. *Из чего состоит слой в нейронной сети? Какие слои бывают? Что такое нейрон?*

Нейрон — это базовая вычислительная единица нейронной сети, имитирующая работу биологического нейрона.

Слой — это группа нейронов, обрабатывающих данные одновременно.

Существуют следующие виды слоёв:

1) входной слой (Input Layer) Получает исходные данные (например, пиксели изображения или вектор признаков). Не имеет весов и вычислений — просто передаёт данные дальше;

2) полносвязный слой (Dense / Fully Connected, FC) Каждый нейрон слоя связан со всеми входами. Используется в классических нейросетях и финальных этапах современных архитектур;

3) свёрточный слой (Convolutional, Conv) Применяет фильтры (ядра) для поиска локальных признаков (например, границ на изображении). Основа архитектур CNN (ResNet, VGG);

4) рекуррентный слой (RNN, LSTM, GRU) Обрабатывает последовательности (текст, временные ряды), сохраняя "память" о предыдущих данных;

5) слой пулинга (Pooling: Max, Average) Уменьшает размерность данных, сохраняя ключевую информацию (например, оставляет максимальное значение в области);

6) слой дропаута (Dropout) Случайно "отключает" часть нейронов во время обучения для борьбы с переобучением;

7) слой нормализации (BatchNorm, LayerNorm) Стандартизирует данные внутри слоя для ускорения обучения и стабилизации сети;

8) слой активации (ReLU, Sigmoid, Softmax, Tanh) Добавляет нелинейность (например, ReLU: f(x) = max(0, x)). Softmax часто используется в выходном слое для классификации;

9) слой внимания (Attention / Self-Attention) Определяет важность элементов входных данных (основа Transformers, BERT, GPT);

10) выходной слой (Output Layer) Формирует итоговый результат

1. *Что такое аутентификация?*

Аутентификация – это своеобразный процесс установления личности субъекта, а также подтверждение подлинности субъекта в рамках какой-либо программы, сайта или операционной системы.

1. *Что такое авторизация?*

Авторизация — это процесс определения прав доступа пользователя после аутентификации (например, может ли пользователь редактировать данные).

1. *Чем аутентификация отличается от авторизации?*

Аутентификация подтверждает, кто ты (идентификация), а авторизация определяет, что ты можешь делать (доступ). Аутентификация происходит первой, авторизация — после.

1. *Для чего нужен токен Telegram-бота?*

Токен Telegram-бота — это уникальный ключ, который выдаётся при создании бота в BotFather. Он используется для идентификации бота и взаимодействия с Telegram API (отправка сообщений, получение обновлений). Токен подтверждает, что запросы к API отправляются от твоего бота.

# Вывод

В ходе лабораторной работы были получены навыки работы с методами ИИ для решения задачи классификации данных с использованием языка программирования Python, а также библиотеками для создания Telegram-бота на языке Python. Была реализована и протестирована программа обучения нейросети для бинарной классификации изображений. Был реализован Telegram-бот, поддерживающий необходимый набор команд от пользователя и определяющий к какому классу принадлежит изображение на основе программы-классификатора.

# Приложение 1

Комментированный листинг исходного кода lab3.py:

import os  
import numpy as np  
import tensorflow as tf  
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
  
from scipy.interpolate import interp1d  
from itertools import cycle  
from sklearn import svm, datasets  
from sklearn.metrics import roc\_auc\_score  
from sklearn.metrics import roc\_curve, auc  
from sklearn.preprocessing import label\_binarize  
from tensorflow.keras.preprocessing import image  
from sklearn.multiclass import OneVsRestClassifier  
from sklearn.model\_selection import train\_test\_split  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
*#Настройка путей к данным*train\_dandelion\_dir = os.path.join('train/dandelion') *#создает пути к папкам, объединяя компоненты*train\_grass\_dir = os.path.join('train/grass')  
valid\_dandelion\_dir = os.path.join('valid/dandelion')  
valid\_grass\_dir = os.path.join('valid/grass')  
  
*#Получение списка файлов*train\_dandelion\_names = os.listdir(train\_dandelion\_dir) *#возвращает список имен файлов в указанной папке*train\_grass\_names = os.listdir(train\_grass\_dir)  
validation\_grass\_names = os.listdir(valid\_grass\_dir)  
  
*#Вывод количества изображений*print('total training dandelion images:', len(os.listdir(train\_dandelion\_dir)))*#Выводит количество файлов в каждой папке, чтобы проверить размер датасета*print('total training grass images:', len(os.listdir(train\_grass\_dir)))  
print('total validation dandelion images:', len(os.listdir(valid\_dandelion\_dir)))  
print('total validation grass images:', len(os.listdir(valid\_grass\_dir)))  
  
*#Визуализация образцов изображений*nrows = 4 *#задают сетку 4x4 для отображения 16 изображений*ncols = 4  
pic\_index = 0 *#инициализирует индекс для выбора изображений*fig = plt.gcf() *#возвращает текущую фигуру Matplotlib*fig.set\_size\_inches(ncols \* 4, nrows \* 4) *#задает размер фигуры (16x16 дюймов).*pic\_index += 8 *#увеличивает индекс на 8 для выбора следующих 8 изображений*next\_dandelion\_pic = [os.path.join(train\_dandelion\_dir, fname)  
 for fname in train\_dandelion\_names[pic\_index-8:pic\_index]] *#список путей к 8 изображениям кошек (из train/dandelion), используя срез [0:8].*next\_grass\_pic = [os.path.join(train\_grass\_dir, fname)  
 for fname in train\_grass\_names[pic\_index-8:pic\_index]]  
for i, img\_path in enumerate(next\_dandelion\_pic + next\_grass\_pic): *#объединяет списки путей (16 изображений: 8 кошек + 8 собак)* sp = plt.subplot(nrows, ncols, i + 1) *#создает подграфик в сетке 4x4 (индекс начинается с 1).* sp.axis('Off') *#отключает оси для чистого отображения изображений.* img = mpimg.imread(img\_path) *#загружает изображение в виде массива NumPy* plt.imshow(img) *#отображает изображение в подграфике.*plt.show() *#выводит всю фигуру с 16 изображениями.  
  
#Настройка генераторов данных*train\_datagen = ImageDataGenerator(rescale=1/255) *#Создаются два генератора: для обучения (train\_datagen) и валидации (validation\_datagen). Аугментация (например, повороты) не используется, только нормализация.*validation\_datagen = ImageDataGenerator(rescale=1/255)  
train\_generator = train\_datagen.flow\_from\_directory( *#загружает изображения из папки train/, ожидая подкаталоги dandelion и grass.* 'train/',  
 classes = ['dandelion', 'grass'], *#задает имена классов (0 для dandelion, 1 для grass).* target\_size=(200, 200), *# изменяет размер всех изображений до 200x200 пикселей.* batch\_size=5, *#задает размер батча (5 изображений за раз)* class\_mode='binary') *#указывает на бинарную классификацию (0 или 1).*validation\_generator = validation\_datagen.flow\_from\_directory(  
 'valid/',  
 classes = ['dandelion', 'grass'],  
 target\_size=(200, 200),  
 batch\_size=19,  
 class\_mode='binary',  
 shuffle=False) *#отключает случайное перемешивание данных  
  
#Создание нейронной сети*model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input\_shape = (200,200,3)), *#создает модель, где слои добавляются последовательно. Слой 1 - Преобразует изображение 200x200x3 (RGB) в одномерный вектор (120,000 элементов).* tf.keras.layers.Dense(128, activation=tf.nn.relu), *#слой 2 - полносвязный слой с 128 нейронами и активацией ReLU (вводит нелинейность)* tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)]) *#слой 3 - выходной слой с 1 нейроном и сигмоидной активацией, возвращающий вероятность (0–1) для бинарной классификации.*model.summary() *#выводит структуру модели: типы слоев, размеры выходов и количество параметров  
  
#Компидяция модели*model.compile(optimizer = tf.keras.optimizers.Adam(), *#использует оптимизатор Adam (эффективен для нейронных сетей)* loss = 'binary\_crossentropy', *#функция потерь для бинарной классификации* metrics=['accuracy']) *#отслеживает точность классификации во время обучения и валидации  
  
#Обучение модели*history = model.fit(train\_generator, *#обучает модель на данных из train\_generator* steps\_per\_epoch=8, *#означает, что за эпоху обрабатывается 8 батчей* epochs=15, *#обучение длится 15 эпох* verbose=1, *#показывает прогресс-бар* validation\_data = validation\_generator, *#оценивает модель на валидационных данных после каждой эпохи* validation\_steps=8) *#обрабатывает 8 батчей для валидации  
  
#Тестирование на одном изображении*uploaded = ['test.jpg'] *#список с одним тестовым изображением*for fn in uploaded:  
 path = 'content/' + fn  
 img = image.load\_img(path, target\_size=(200, 200)) *#загружает и изменяет размер изображения.* x = image.img\_to\_array(img) *#преобразует изображение в массив NumPy (форма: (200, 200, 3)).* plt.imshow(x / 255.) *#отображает изображение (нормализуя пиксели)* x = np.expand\_dims(x, axis=0) *#добавляет размерность батча (форма: (1, 200, 200, 3))* images = np.vstack([x]) *#объединяет массивы* classes = model.predict(images, batch\_size=10) *#предсказывает вероятность класса* print(classes[0])  
 if classes[0] < 0.5:  
 print(fn + " is a dandelion")  
 else:  
 print(fn + " is a grass")

# Приложение 2

Листинг программы-классификатора (для варианта человек/жираф):

import os *# Для работы с файлами и папками*import sys *# Для получения аргументов командной строки*import numpy as np *# Для работы с массивами*import tensorflow as tf *# Основная библиотека для создания и обучения нейронных сетей*import matplotlib.pyplot as plt *# Для построения графиков*import matplotlib.image as mpimg *# Для чтения и отображения изображений*from tensorflow.keras.preprocessing import image *# Для загрузки и обработки изображений*from tensorflow.keras.preprocessing.image import ImageDataGenerator *# Для генерации и аугментации данных*if len(sys.argv) != 4:  
 print("Use: python main.py <train\_dir> <valid\_dir> <test\_file>")  
 sys.exit(1)  
train\_dir = sys.argv[1] *#train*valid\_dir = sys.argv[2] *# valid*test\_file = sys.argv[3] *#content/test.jpg*train\_human\_dir = os.path.join(train\_dir, 'human')  
train\_giraffe\_dir = os.path.join(train\_dir, 'giraffe')  
valid\_human\_dir = os.path.join(valid\_dir, 'human')  
valid\_giraffe\_dir = os.path.join(valid\_dir, 'giraffe')  
  
for dir\_path in [train\_human\_dir, train\_giraffe\_dir, valid\_human\_dir, valid\_giraffe\_dir]:  
 if not os.path.exists(dir\_path):  
 raise FileNotFoundError(f"Directory not found: {dir\_path}")  
if not os.path.exists(test\_file):  
 raise FileNotFoundError(f"Test file not found: {test\_file}")  
  
print('total training human images:', len(os.listdir(train\_human\_dir)))  
print('total training giraffe images:', len(os.listdir(train\_giraffe\_dir)))  
print('total validation human images:', len(os.listdir(valid\_human\_dir)))  
print('total validation giraffe images:', len(os.listdir(valid\_giraffe\_dir)))  
  
*# Визуализация нескольких изображений для проверки*nrows = 4 *# Количество строк в сетке изображений (4 строки)*ncols = 4 *# Количество столбцов в сетке (4 столбца)*pic\_index = 0 *# Начальный индекс для выбора изображений*fig = plt.gcf() *# Получает текущую фигуру (окно для графиков)*fig.set\_size\_inches(ncols \* 4, nrows \* 4) *# Устанавливает размер окна (16×16 дюймов)  
  
# Получение списка файлов в папках*train\_human\_names = os.listdir(train\_human\_dir) *# Список файлов в train/human*train\_giraffe\_names = os.listdir(train\_giraffe\_dir) *# Список файлов в train/giraffe  
  
# Выбор 8 изображений людей и 8 жирафов для отображения*pic\_index += 8 *# Увеличиваем индекс на 8, чтобы взять следующие 8 файлов*next\_human\_pic = [os.path.join(train\_human\_dir, fname) for fname in train\_human\_names[pic\_index-8:pic\_index]]  
next\_giraffe\_pic = [os.path.join(train\_giraffe\_dir, fname) for fname in train\_giraffe\_names[pic\_index-8:pic\_index]]  
  
*# Отображение изображений в сетке*for i, img\_path in enumerate(next\_human\_pic + next\_giraffe\_pic): *# Объединяем списки (16 изображений)* sp = plt.subplot(nrows, ncols, i + 1) *# Создаём подграфик в сетке (позиция i+1)* sp.axis('Off') *# Отключаем оси координат для чистоты изображения* img = mpimg.imread(img\_path) *# Читаем изображение в массив* plt.imshow(img) *# Отображаем изображение в подграфике*plt.show() *# Показываем окно с изображениями*plt.close() *# Закрываем окно, чтобы освободить память  
  
# Подготовка данных с помощью ImageDataGenerator*train\_datagen = ImageDataGenerator(  
 rescale=1/255, *# Нормализация: значения пикселей делятся на 255 (из 0–255 в 0–1), чтобы упростить обучение* rotation\_range=20, *# Случайный поворот изображений на угол до ±20 градусов (аугментация)* width\_shift\_range=0.2, *# Случайный сдвиг по горизонтали на ±20% ширины (аугментация)* height\_shift\_range=0.2, *# Случайный сдвиг по вертикали на ±20% высоты (аугментация)* zoom\_range=0.2, *# Случайное увеличение/уменьшение на ±20% (аугментация)* horizontal\_flip=True, *# Случайное отражение по горизонтали (аугментация)* fill\_mode='nearest' *# Заполняет пустые области после трансформаций ближайшими пикселями*)  
  
*# Генератор для валидационных данных (без аугментации, только нормализация)*validation\_datagen = ImageDataGenerator(rescale=1/255)  
  
*# Создание генератора для обучающих данных*train\_generator = train\_datagen.flow\_from\_directory(  
 train\_dir, *# Путь к папке с обучающими данными (train)* classes=['human', 'giraffe'], *# Классы: human (0), giraffe (1)* target\_size=(200, 200), *# Изменение размера всех изображений до 200×200 пикселей* batch\_size=32, *# Размер пакета: 32 изображения обрабатываются за один шаг* class\_mode='binary' *# Бинарная классификация: 0 (human), 1 (giraffe)*)  
  
*# Создание генератора для валидационных данных*validation\_generator = validation\_datagen.flow\_from\_directory(  
 valid\_dir, *# Путь к папке с валидационными данными (valid)* classes=['human', 'giraffe'], *# Те же классы* target\_size=(200, 200), *# Тот же размер* batch\_size=32, *# Тот же размер пакета* class\_mode='binary', *# Бинарная классификация* shuffle=False *# Отключаем перемешивание, чтобы порядок данных был фиксированным*)  
  
print("Метки классов:", train\_generator.class\_indices)  
*# Проверка наличия сохранённой модели*model\_path = 'best\_human\_giraffe\_model.h5'  
use\_existing\_model = False  
if os.path.exists(model\_path):  
 print(f"Обнаружена сохранённая модель: {model\_path}")  
 response = input("Хотите использовать сохранённую модель? (y/n): ").strip().lower()  
 if response == 'y':  
 use\_existing\_model = True  
  
if use\_existing\_model:  
 model = tf.keras.models.load\_model(model\_path)  
 print("Сохранённая модель загружена.")  
else:  
 model = tf.keras.models.Sequential([ *# Создаём последовательную модель (слои добавляются один за другим)  
 # Первый сверточный слой: извлекает простые признаки (например, края, углы)* tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(200, 200, 3)),  
 *# - 32: количество фильтров (модель создаёт 32 карты признаков)  
 # - (3, 3): размер фильтра (окно 3×3 пикселя)  
 # - activation='relu': функция активации ReLU (оставляет только положительные значения, обнуляет отрицательные)  
 # - input\_shape=(200, 200, 3): входной размер (200×200 пикселей, 3 канала RGB)  
 # Результат: 32 карты признаков  
  
 # Первый пулинговый слой: уменьшает размер карт признаков, сохраняя важные детали* tf.keras.layers.MaxPooling2D(2, 2),  
 *# - (2, 2): размер окна пулинга  
 # - Берёт максимум из окна 2×2, уменьшая размер карт в 2 раза  
  
 # Второй сверточный слой: извлекает более сложные признаки (например, текстуры)* tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),  
 *# - 64: количество фильтров (больше, чтобы находить более сложные признаки)  
 # - (3, 3): тот же размер фильтра  
 # - activation='relu': снова ReLU  
 # Результат: 64 карты  
  
 # Второй пулинговый слой: уменьшает размер карт* tf.keras.layers.MaxPooling2D(2, 2),  
 *# - Уменьшает размер в 2 раза  
  
 # Третий сверточный слой: ещё более сложные признаки (например, формы)* tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),  
 *# - 128: ещё больше фильтров  
 # - Результат: 128 карт  
  
 # Третий пулинговый слой* tf.keras.layers.MaxPooling2D(2, 2),  
 *# - Уменьшает размер в 2 раза  
  
 # Преобразование карт признаков в одномерный вектор* tf.keras.layers.Flatten(),  
  
 *# Полносвязный слой: объединяет признаки для принятия решения* tf.keras.layers.Dense(128, activation='relu'),  
 *# - 128: количество нейронов  
 # - activation='relu': снова ReLU  
 # - Каждый нейрон принимает значени из массива, умножает их на веса, добавляет смещение и применяет ReLU  
  
 # Dropout: предотвращает переобучение* tf.keras.layers.Dropout(0.5),  
 *# - Случайно отключает 50% нейронов на каждом шаге обучения  
  
 # Выходной слой: делает финальное предсказание* tf.keras.layers.Dense(1, activation='sigmoid'),  
 *# - 1: один нейрон, так как это бинарная классификация  
 # - activation='sigmoid': выдаёт вероятность от 0 до 1* ])  
 *# Вывод структуры модели* model.summary()  
 *# Компиляция модели: настройка параметров обучения* model.compile(  
 optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001), *# Оптимизатор Adam с шагом обучения 0.0001* loss='binary\_crossentropy', *# Функция потерь для бинарной классификации* metrics=['accuracy'] *# Отслеживаем точность во время обучения* )  
 *# Обучение модели* history = model.fit(  
 train\_generator, *# Генератор обучающих данных* steps\_per\_epoch=len(train\_generator) // 2, *# Количество шагов на эпоху (половина всех батчей)* epochs=30, *# Количество эпох (30 полных проходов по данным)* verbose=1, *# Показывает прогресс обучения (1 — с прогресс-баром)* validation\_data=validation\_generator, *# Генератор валидационных данных* validation\_steps=len(validation\_generator) // 2, *# Количество шагов для валидации (половина батчей)* )  
 *# Сохранение финальной модели (после всех эпох)* model.save('best\_human\_giraffe\_model.h5')  
 *# Оценка модели на обучающих и валидационных данных* train\_loss, train\_acc = model.evaluate(train\_generator)  
 valid\_loss, valid\_acc = model.evaluate(validation\_generator)  
 print(f"Точность на обучающем наборе: {train\_acc:.4f}")  
 print(f"Точность на валидационном наборе: {valid\_acc:.4f}")  
  
  
 plt.plot(history.history['accuracy'], label='train\_accuracy')  
 plt.plot(history.history['val\_accuracy'], label='val\_accuracy')  
 plt.title('Model Accuracy')  
 plt.ylabel('Accuracy')  
 plt.xlabel('Epoch')  
 plt.legend()  
 plt.show()  
 plt.close()  
  
 plt.plot(history.history['loss'], label='train\_loss')  
 plt.plot(history.history['val\_loss'], label='val\_loss')  
 plt.title('Model Loss')  
 plt.ylabel('Loss')  
 plt.xlabel('Epoch')  
 plt.legend()  
 plt.show()  
 plt.close()  
  
*# Тестирование модели на тестовом изображении*test\_images = [test\_file]  
for fn in test\_images:  
 try:  
 img = image.load\_img(fn, target\_size=(200, 200)) *# Загружаем изображение и изменяем размер до 200×200* x = image.img\_to\_array(img) *# Преобразуем изображение в массив (200×200×3)* plt.imshow(x / 255.) *# Показываем изображение (нормализуем значения пикселей в 0–1)* plt.show() *# Отображаем окно* plt.close() *# Закрываем окно* x = np.expand\_dims(x, axis=0) *# Добавляем размерность батча (из (200, 200, 3) в (1, 200, 200, 3))* images = np.vstack([x]) *# Создаём массив для предсказания* classes = model.predict(images, batch\_size=10) *# Делаем предсказание (вероятность от 0 до 1)* print(f"Предсказание для {fn}: {classes[0]}") *# Выводим вероятность* if classes[0] < 0.5:  
 print(f"{fn} is a human")  
 else:  
 print(f"{fn} is a giraffe")  
 except FileNotFoundError:  
 print(f"Файл {fn} не найден.")

# Приложение 3

Листинг кода для реализации работы Telegram-бота:

import telebot  
from telebot import types  
import os  
import numpy as np  
import tensorflow as tf  
from tensorflow.keras.preprocessing import image  
import random  
from PIL import Image  
import io  
import csv  
import hashlib  
  
*# Инициализация бота*bot = telebot.TeleBot("7842869515:AAEoqR7Dwe1dNh5UjSU6bDsRcO8S0ijmX5s") *# Замените на свой токен  
  
# Список авторизованных пользователей*signed\_in\_users = []  
  
*# Загрузка модели*MODEL\_PATH = 'best\_human\_giraffe\_model.h5'  
try:  
 model = tf.keras.models.load\_model(MODEL\_PATH)  
 print("Модель успешно загружена")  
except Exception as e:  
 print(f"Ошибка загрузки модели: {e}")  
 model = None  
  
  
*# Функция для хеширования пароля*def hash\_password(password):  
 return hashlib.sha256(password.encode()).hexdigest()  
  
  
*# Функция для загрузки пользователей из файла*def load\_users():  
 users = {}  
 try:  
 with open('users.csv', mode='r') as file:  
 reader = csv.reader(file)  
 for row in reader:  
 if len(row) >= 2:  
 users[row[0]] = row[1]  
 except FileNotFoundError:  
 open('users.csv', mode='w').close()  
 return users  
  
  
*# Функция для сохранения пользователя в файл*def save\_user(user\_id, hashed\_password):  
 with open('users.csv', mode='a', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow([user\_id, hashed\_password])  
  
  
*# Обработчик команды /start*@bot.message\_handler(commands=['start'])  
def send\_welcome(message):  
 bot.reply\_to(message, "Привет! Я бот для классификации изображений. Используй /help для списка команд.")  
  
  
*# Обработчик команды /help*@bot.message\_handler(commands=['help'])  
def show\_help(message):  
 help\_text = """  
📌 Доступные команды:  
/register - Зарегистрироваться (установить пароль)  
/login - Войти в систему (ввести пароль)  
/logout - Выйти из системы  
/predict - Классифицировать изображение (требуется авторизация)  
/help - Показать это сообщение  
 """  
 bot.send\_message(message.chat.id, help\_text)  
  
  
*# Обработчик команды /register*@bot.message\_handler(commands=['register'])  
def register\_user(message):  
 chat\_id = str(message.chat.id)  
 users = load\_users()  
  
 if chat\_id in users:  
 bot.send\_message(chat\_id, "❌ Вы уже зарегистрированы! Используйте /login.")  
 return  
  
 msg = bot.send\_message(chat\_id, "🔑 Придумайте пароль для регистрации:")  
 bot.register\_next\_step\_handler(msg, process\_password\_step)  
  
  
def process\_password\_step(message):  
 chat\_id = str(message.chat.id)  
 password = message.text.strip()  
 hashed\_password = hash\_password(password)  
  
 save\_user(chat\_id, hashed\_password)  
 bot.send\_message(chat\_id, "✅ Регистрация успешна! Теперь войдите с помощью /login.")  
  
  
*# Обработчик команды /login*@bot.message\_handler(commands=['login'])  
def login\_user(message):  
 chat\_id = str(message.chat.id)  
 users = load\_users()  
  
 if chat\_id not in users:  
 bot.send\_message(chat\_id, "❌ Вы не зарегистрированы! Используйте /register.")  
 return  
  
 if chat\_id in signed\_in\_users:  
 bot.send\_message(chat\_id, "❌ Вы уже вошли в систему! Используйте /logout для выхода.")  
 return  
  
 msg = bot.send\_message(chat\_id, "🔑 Введите ваш пароль:")  
 bot.register\_next\_step\_handler(msg, process\_login\_step)  
  
  
def process\_login\_step(message):  
 chat\_id = str(message.chat.id)  
 input\_password = message.text.strip()  
 users = load\_users()  
  
 if chat\_id not in users:  
 bot.send\_message(chat\_id, "❌ Вы не зарегистрированы! Используйте /register.")  
 return  
  
 hashed\_input = hash\_password(input\_password)  
 if users[chat\_id] == hashed\_input:  
 signed\_in\_users.append(chat\_id)  
 bot.send\_message(chat\_id, "✅ Вход выполнен! Теперь вы можете использовать /predict.")  
 else:  
 bot.send\_message(chat\_id, "❌ Неверный пароль! Попробуйте снова /login.")  
  
  
*# Обработчик команды /logout*@bot.message\_handler(commands=['logout'])  
def logout\_user(message):  
 chat\_id = str(message.chat.id)  
  
 if chat\_id in signed\_in\_users:  
 signed\_in\_users.remove(chat\_id)  
 bot.send\_message(chat\_id, "✅ Выход выполнен! Для использования /predict войдите снова /login.")  
 else:  
 bot.send\_message(chat\_id, "❌ Вы не вошли в систему!")  
  
  
*# Обработчик команды /predict (только для авторизованных)*@bot.message\_handler(commands=['predict'])  
def predict\_image(message):  
 chat\_id = str(message.chat.id)  
  
 if chat\_id not in signed\_in\_users:  
 bot.send\_message(chat\_id, "❌ Доступ запрещен! Зарегистрируйтесь /register и войдите /login.")  
 return  
  
 msg = bot.send\_message(chat\_id, "📷 Отправьте изображение для классификации:")  
 bot.register\_next\_step\_handler(msg, process\_image\_step)  
  
  
def process\_image\_step(message):  
 chat\_id = str(message.chat.id)  
  
 if not message.photo:  
 bot.send\_message(chat\_id, "❌ Это не изображение! Попробуйте снова /predict.")  
 return  
  
 if model is None:  
 bot.send\_message(chat\_id, "❌ Модель не загружена. Пожалуйста, свяжитесь с администратором.")  
 return  
  
 try:  
 *# Получаем файл изображения* file\_info = bot.get\_file(message.photo[-1].file\_id)  
 downloaded\_file = bot.download\_file(file\_info.file\_path)  
  
 *# Конвертируем изображение в нужный формат* img = Image.open(io.BytesIO(downloaded\_file))  
 img = img.resize((200, 200)) *# Изменяем размер до 200x200* x = image.img\_to\_array(img)  
 x = x / 255.0 *# Нормализация* x = np.expand\_dims(x, axis=0)  
  
 *# Предсказание* classes = model.predict(x, batch\_size=1)  
 probability = classes[0][0]  
 *# Определяем класс* if probability < 0.5:  
 result = "human"  
 prob\_text = f"🧍‍♂️🧍‍♀️ Вероятность человека: {(1 - probability):.4f}\n🦒 Вероятность жирафа: {probability:.4f}"  
 else:  
 result = "giraffe"  
 prob\_text = f"🧍‍♂️🧍‍♀️ Вероятность человека: {(1 - probability):.4f}\n🦒 Вероятность жирафа: {probability:.4f}"  
  
 bot.send\_message(chat\_id, f"Изображение классифицировано как {result}!\n{prob\_text}")  
 except Exception as e:  
 bot.send\_message(chat\_id, f"❌ Ошибка обработки изображения: {str(e)}. Попробуйте ещё раз.")  
  
  
*# Запуск бота*if \_\_name\_\_ == "\_\_main\_\_":  
 print("Бот запущен...")  
 bot.infinity\_polling()