ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Московский институт электроники и математики

Моя первая нейронная сеть. Обитатели гидросферы: от датасета до классификации

Отчет по проектной работе студента образовательной программы бакалавриата
«Прикладная математика»
по направлению подготовки 01.03.04 ПРИКЛАДНАЯ МАТЕМАТИКА

Студент
БПМ225
Лихачева Рада Дмитриевна

Руководитель проекта
Профессор
Попов Виктор Юрьевич

Москва 2023г.

Содержание

1 Аннотация

Нейронные сети - наиболее актуальный в настоящее время инструмент научного исследования практически в любой области. При работе над данным проектом нас в первую очередь заинтересовал аспект применения нейросетей для классификации объектов. Особенный интерес для научно-практических целей представляет, на наш взгляд, классификация естественных объектов, поэтому для составления базового рабочего прототипа классификационной нейросети были выбраны объекты биологические - рыбы и медузы.

Цель проекта: Создание нейронной сети с помощью библиотеки Keras для классификации изображений оригинального набора данных.

Этапы проекта:

- Создание и обработка датасета. Написание программу на Python для решения этой задачи.

- Создание нейронной сети: подбор слоев, активационных функций, функции ошибки, оптимайзера.

- Обучение нейронной сети.

- Тестирование обученной модели нейронной сети.

Результат проекта: Работающая нейросеть, которая с точность 90% классифицирует фотографии двух классов объектов: рыбки и медузы.

2 Подготовка набора данных для обучения

2.1 Этапы создания собственного датасета

Качественно подобранный датасет сильно упрощает обучение нейросети, а также увеличивает точность модели. В ходе выполнения проекта этот факт не раз подтверждался. В некоторых случаях, несколько подредактировав или увеличив один только датасет, получалось повысить точность почти на 20%. Поэтому очень важно правильно собирать и обрабатывать исходные данные. Одним из основных критериев является то, что набор данных, на которых обучается нейросеть, должен как можно больше соответствовать тому, для чего она будет использоваться. В данном проекте речь идет о правильно подобранных картинках. Как на обучающих, так и на тестовых выборках должен быть примерно одинаковый ракурс, освещение, фокус на распознаваемом объекте, его размер и так далее.

В Keras есть уже подготовленные датасеты, которые можно легко загрузить в код. Например, Cifar10 - набор из 10 классов изображений: самолеты, птицы, коты и другие.

Для того, чтобы составить собственный датасет, для начала необходимо найти достаточное количество изображений для каждого класса. Делать это вручную очень долго и трудоемко, поэтому можно ускорить этот процесс. Для этого была написана программа, которая будет искать картинки в Яндексе по конкретному запросу и сохранять их в нужную папку под нужными названиями.

Чтобы датасет, созданный для данного проекта, был более разнообразным - он был составлен не просто из запросов “аквариумная рыбка” и “медуза”, а из разных видов и медуз, и рыб. Для этого в интернете были найдены наиболее распространенные виды, которые затем были записаны в список titles в файле titles.py.

В titles эти разновидности записываются как фразы для поисковых запросов, а в список urls - под каким названием сохраняется фотография под таким же индексом в titles. Затем, уже в основном файле, указывается значение для переменной j, которая будет отвечать за индексы (с какого номера начать сохранять фотографии). Это удобно, если фотографии загружать в несколько подходов. Таким образом, не придётся скачивать все с начала, а можно просто добавлять новые в

конец.

Пример работы кода: если titles = ['гуппи', 'золотая рыбка', 'медуза аурелия'], тогда urls = ['fish_', 'fish_', 'jellyfish_']. Последовательно в указанные папки загружаются фотки по запросам "гуппи" и "золотая рыбка", которые будут будут записать под именами "fish_(номер)", а по запросу "медуза аурелия" - "jellyfish_(номер)". При желании можно делать более детальные разбивки (например, разбить рыбок на виды или добавить каких-то морских обитателей).

В итоге в папке fishes окажутся фотографии fish_0, fish_1, fish_2 и тд, а в jellyfish будут jellyfish _0, jellyfish _1 и тд

После скачивания достаточно большого количества изображений - необходимо проверить, насколько каждое из них подходит к конкретной задаче. Все некачественные фотографии, которые могут сильно помешать обучению модели необходимо удалить или заменить качественными. Таким образом, из папок fishes и jellyfishes были удалены фоторафии, на которых объекты не очень хорошо видны или на которых много лишних вещей, например, людей. Также были удалены фотографии, на которых изображены рисунки или на которых рыбки и медузы находились в нестандартных условиях, например, выброшенные на сушу. После этого изображений становится на порядок меньше, однако датасет можно считать хорошо подготовленным и можно начать пробовать обучать на нем нейронную сеть.

2.2 Реализация кода поиска изображений

Код написан на языке программирования Python в среде разработки IDLE. Перед запуском нужно установить в среду разработки библиотеки selenium и requests. Также необходимо иметь на устройстве браузер Google Chrome и установить chromedriver с такой же версией, как у браузера [1].

Первым делом импортируем библиотеки и файл titles.py, в котором находятся только 2 списка: titles - с нужными поисковыми запросами, urls - с названиями, под которыми будут сохраняться фотографии по этим поисковым запросам. Затем загружаем webdriver, указывая путь до сохраненного chromedriver. Потом задаем значение для переменной j - с какого номера начинать записывать картинки. После этого проходим по всем запросам, получаем ссылки на картинки и скачиваем

изображения в формате png в указанную папку. Конструкция try/except применена для того, чтобы в случаях, когда вдруг возникает ошибка, код продолжал работать, а не прерывался.

```
from selenium import webdriver
import requests
from titles import *

browser = webdriver.Chrome(path to the driver)
j = 0
for i in range(len(titles)):
    print(i, " ", urls[i], titles[i])
    browser.get("https://yandex.ru/images/search?from=tabbar&
       text=" + titles[i])
    data = browser.page_source

    try:
        data = data.split('img_href&quot;:&quot;')
        data.pop(0)
        data = data[:-1]

        for kartinka in data:
            #print (j)
            try:
                kartinka = kartinka.split('&quot;')
                kartinka = kartinka[0]
                p = requests.get(kartinka)
                out = open("fishes\\"+urls[i]+str(j)+".png", "
                   wb")
                out.write(p.content)
                out.close()
            except:
                print('картинка не скачалась')
            j += 1
```

```
except Exception as e:
    error = tr.TracebackException(exc_type =type(e),
        exc_traceback = e.__traceback__ , exc_value =e).
        stack[-1]
    print('{} in {} row:{} | arguments:{}'.format(e, error
        .lineno, error.line))
```

В результате мы получаем на устройстве 2 папки: fishes и jellyfishes. В каждой папке есть достаточно количество отобранных фотографий рыбок и медуз соответственно. Таким образом, у нас получилось создать уникальный датасет.

3 Написание нейросети на Python

3.1 Создание модели нейросети

После того, как набор изображений готов, можно перейти к этапу создания нейронной сети. Для этой задачи будет использована среда Google Colab, поскольку она хорошо подходит для работы с нейросетями, а также у нее очень удобный интерфейс.

Сначала подключаем библиотеки, которые будем использовать [3], а также загружаем на Google диск в отдельную папку наши папки с фотографиями. После загрузки получаем папку project_ns, в которой находятся 2 другие папки: “fishes vs jellyfish”, в которой находятся папки “fishes” и “jellyfishes” с изображениями аквариумных рыбок и медуз соответственно, и test, с тестовыми картинками рыб и медуз.

После импортирования всех библиотек, перед этим установив их, если они еще не установлены, необходимо также подключиться к Google Disk, чтобы получить доступ к датасету. Все эти процессы и результат их работы можно увидеть на рисунке 1

```
!pip install keras
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.12.0)
```

Иморт библиотек

```
[15] from tensorflow.keras.models import Sequential, load_model
     #Базовые слои для свёрточных сетей
     from tensorflow.keras.preprocessing.image import ImageDataGenerator # работа с изображениями
     from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization
     from tensorflow.keras.optimizers import Adam # оптимизаторы

     from tensorflow.keras.preprocessing import image #Для отрисовки изображений
     from google.colab import files #Для загрузки своей картинки

     import numpy as np #Библиотека работы с массивами
     import matplotlib.pyplot as plt #Для отрисовки графиков
     from PIL import Image #Для отрисовки изображений

     import os #Для работы с файлами


     %matplotlib inline
```

Подключаем гугл диск

```
[2]  # подключем диск
     from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Рис. 1: Подключения, необходимые для работы

После этого задаем путь до папки с папками, разбитыми по классам, и основные константы, которые часто менялись в процессе поиска наилучшего результата работы нейросети - рисунок 2.

batch_size - количество фотографий для одной итерации, лучше всего задать кратным общему количеству фотографий. img_width, img_height - ширина и высота изображения соответственно.

Задаем основные константы

```
[7]  # путь к данным (загружаем из гугл диска)
     train_path = '/content/drive/My Drive/project_ns/fishes vs jellyfish/'

     batch_size = 32
     img_width = 230
     img_height = 200
```

Рис. 2: Задаем основные константы

Следующий шаг - загрузка и подготовка датасета к обучению на нем модели. ImageDataGenerator - удобный в использовании инструмент для увеличения количества фотографий для нейросети путем изменения уже существующих различными методами [2]. Например, у horizontal_flip установлено значение True, чтобы переворачивать изображения по горизонтали. Также указывается соотношение обучающей и проверочной выборки. Чаще всего это 0,2 или 0,1. На рисунке 3 можно увидеть другие параметры и что они означают.

Затем datagen используется для создания обучающих и тестовых наборов для модели. Метод flow_from_directory вызывается для создания пакетов данных изображения из каталога, при этом параметр target_size устанавливает размер изображений, class_mode имеет значение “binary”, поскольку имеется только 2 класса. Если классов больше - нужно указывать “categorical”. Для параметра shuffle установлено значение True, чтобы перемешивать изображения, а для параметра subset установлено значение «traning» для train_generator и «validation» для validation_generator, чтобы указать, как изображения будут разделены на наборы для обучающей и проверочной выборок.

В конце выводим количество фотографий в каждой выборке, а также названия классов, которые соответствуют названиям папок, в которых эти изображения находятся.

Работа с набором данных

```
[10] # Генератор изображений
datagen = ImageDataGenerator(
    rescale=1. / 255, # Значения цвета меняем на дробные показания
    rotation_range=15, # Поворачиваем изображения при генерации выборки
    width_shift_range=0.1, # Двигаем изображения по ширине при генерации выборки
    height_shift_range=0.1, # Двигаем изображения по высоте при генерации выборки
    zoom_range=0.1, # Зумируем изображения при генерации выборки
    horizontal_flip=True, # Отзеркаливание изображений
    fill_mode='nearest', # Заполнение пикселей вне границ ввода
    validation_split=0.2 # Указываем разделение изображений на обучающую и тестовую выборку
)
```

```
[11] print("Обучающая выборка:")
train_generator = datagen.flow_from_directory(
    train_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True,
    subset='training'
)

#
print("\nПроверочная выборка:")
validation_generator = datagen.flow_from_directory(
    train_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True,
    subset='validation'
)

print('\nClasses:')
print(*os.listdir(train_path), sep=', ')
```

```
Обучающая выборка:
Found 2484 images belonging to 2 classes.

Проверочная выборка:
Found 620 images belonging to 2 classes.

Classes:
fishes, jellyfishes
```

Рис. 3: Работа с набором изображений

Затем приступаем к созданию модели. Это модель сверточной нейронной сети, которая состоит из нескольких слоев, которые последовательно обрабатывают входные данные. Полностью итоговую модель, полученную в результате многочисленных экспериментов с различными параметрами, можно увидеть на рисунке 4.

Conv2D - сверточный слой. Первый слой имеет 32 ядра (канала), каждое из которых имеет форму 3x3, использует функцию активации ReLU, а также принимает на вход данные с фиксированными значениями img_height - высота в пикселях, img_width - ширину в пикселях, и 3 цветовых канала (красный, зеленый и синий). В последующих изменяется количество каналов. MaxPooling2D - уменьшает карту признаков, оставляя максимальные значения. Flatten - вытягивает тензор в единый вектор.

Dense - полносвязный слой. Последний - выходной. В качестве функции активации использована функция сигмоида, котрая выражается формулой 1. Поскольку эта функция имеет ограничение выходного значения на промежутке от 0 до 1, то она отлично подходит для задач бинарной классификации, поскольку выходные значения можно легко интерпретировать как вероятность. BatchNormalization - для ускорения, Dropout - для предотвращения переобучения.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

В конце компилируем модель. Несмотря на то, что классификация бинарная, большей точности получилось достичь, используя функцию ошибки "categorical". Оптимизатор - Адам, в ходе эксперименотов с параметрами получилось, что лучшие точность и время при скорости обучения learning_rate=0.0001.

```
[12] #Создаем последовательную модель
     model = Sequential()

     #Первый сверточный слой
     model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)))
     model.add(MaxPooling2D(pool_size=(2, 2)))

     #Второй сверточный слой
     model.add(Conv2D(64, (3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))

     #Третий сверточный слой
     model.add(Conv2D(64, (3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))

     #Четвертый сверточный слой
     model.add(Conv2D(128, (3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))

     #Слой преобразования двумерных данных в одномерные
     model.add(Flatten())

     #Полносвязный слой
     model.add(Dense(512, activation='relu'))
     model.add(BatchNormalization())
     #Полносвязный слой
     model.add(Dense(512, activation='relu'))
     model.add(Dropout(0.1))
     model.add(BatchNormalization())
     #Полносвязный слой
     model.add(Dense(512, activation='relu'))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())

     # Выходной полносвязный слой
     model.add(Dense(1, activation='sigmoid'))

     # на всякий случай проверка количества классов и количества картинок в выборках
     print(len(train_generator.class_indices))
     print(train_generator.samples, validation_generator.samples)


     model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.0001), metrics=['accuracy'])

     2
     2484 620
```

Рис. 4: Создание и компиляция модели

3.2 Обучение модели

Обучение нейронной сети - самый долгий процесс. Обычно занимает от 1 до 3 часов (пример на рисунке 6), в зависимости от количества эпох, данных и размера batch_size. Именно из-за времени обучения подбор идеальных параметров для конкретного датасета тяжелый и длительный процесс. Для того, чтобы обучить сеть используем метод fit (рисунок 5), в котором указываем пути и шаги для тренировочной и проверочной выборок, также количество эпох - сколько раз датасет проходит через нейросеть, verbose=1 - для отображения процесса.

Чтобы была возможность через время опять воспользоваться данной обученной моделью и не ждать заново 2 часа, - используется метод model.save(путь до папки, в которую сохраняем).

```
history = model.fit(
    train_generator,
    steps_per_epoch = train_generator.samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // batch_size,
    epochs=35,
    verbose=1
)

model.save('/content/drive/My Drive/project_ns_models/fish_or_jellyfish_3.h5')
```

Рис. 5: Обучение модели

Рис. 6: Время обучение модели

Epoch	Точность на обучающей	Точность на проверочной
1	0.8340	0.7368
5	0.9172	0.8421
10	0.9282	0.8618
15	0.9470	0.8717
20	0.9494	0.8980
25	0.9580	0.8816
30	0.9686	0.8832
35	0.9659	0.8931

Таблица 1: Точности выборок

В таблице 1 представлены результаты точности модели на протяжении обучения.

На рисунке 7 показано, как отображать графики изменения точности.

```
[16] #Отбражаем график точности обучения
     plt.plot(history.history['accuracy'],  label='Доля верных ответов на обучающем наборе')
     plt.plot(history.history['val_accuracy'],  label='Доля верных ответов на проверочном наборе')
     plt.xlabel('Эпоха обучения')
     plt.ylabel('Доля верных ответов')
     plt.legend()
     plt.show()
```

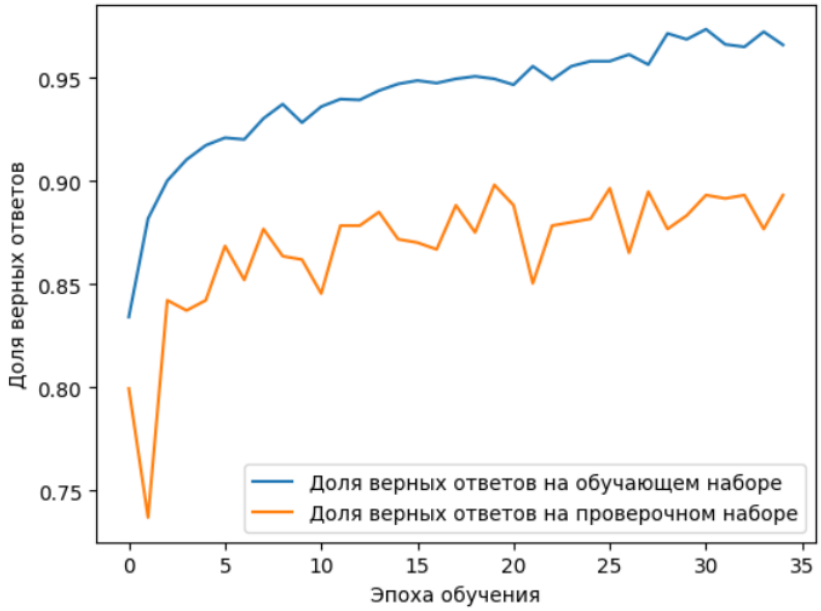

Рис. 7: Графики точности

Чтобы быстро протестировать нейросеть, можно воспользоваться той моделью, которая уже была обучена раньше, просто загрузив ее с диска, как показано на рисунке 8:

Можно запустить ячейку с кодом ниже, вместо того, чтобы полностью обучать модель заново, и быстрее протестировать нейросеть.

```
[ ] # загрузка уже обученной модели с диска
    model_path = '/content/drive/My Drive/project_ns_models/fish_or_jellyfish_2.h5'
    model = load_model(model_path)
```

Рис. 8: Загрузка уже обученной модели

3.3 Тестирование нейросети

После завершения обучения модели ее нужно протестировать на новом наборе изображений - рисунок 9. Для этого загружаем на диск папку с картинками в формате 1.png, 2.png и так далее. Всего 25 штук. Затем для каждой картинки по-

лучаем результат “Медуза” или “Рыбка” с помощью нейронной сети. Эти результаты выводим двумя способами: печатаем номер картинки и рядом ответ (рисунок 11), выводим картинку и подписываем ее (рисунок 10). Для удобства полотно, на котором выводятся изображения, было разбито по 5 фотографий в строке.

Проверяем модель уже на новых картинках

Чтобы загрузить свои - можно добавить в папку 'test' (находится в папке вместе с 'fishes vs jellyfish') собственные изображения медузы или аквариумной рыбки (модель также различает большую часть изображений с несколькими ТОЛЬКО рыбками, или только медузами). Или загрузить картинку сразу в среду разработки и запустить ячейк

```
def func(name):
  # загрузка изображения
  img = image.load_img('/content/drive/MyDrive/project_ns/test/' + name + '.png', target_size=(img_height, img_width))

  # выводим изображение
  axs[(int(name) - 1) % 5].imshow(img)

  img = image.img_to_array(img)
  img = np.expand_dims(img,axis=0)
  answer = model.predict(img)

  # выводим подпись класс, определенный нейросетью
  if answer >= 0.5:
    axs[(int(name) - 1) % 5].set_title('Медуза')
    print(f'{name}: Медуза')
  else:
    axs[(int(name) - 1) % 5].set_title('Рыбка')
    print(f'{name}: Рыбка')


n = 25 # количество фотографий в тестовой выборке

for i in range(1, n + 1):
  if (i - 1) % 5 == 0:
    fig, axs = plt.subplots(1, 5, figsize=(25, 5)) # Создаем полотно из 5 картинок
  name = str(i)
  func(name)


plt.show() # Показываем изображения
```

Рис. 9: Тестирование на изображениях из папки

Рис. 10: Пример вывода картинками с подписями на тестовых изображениях

```
1/1 [==============================] - 0s 28ms/step
15: Рыбка
1/1 [==============================] - 0s 32ms/step
16: Рыбка
1/1 [==============================] - 0s 29ms/step
17: Медуза
1/1 [==============================] - 0s 28ms/step
18: Медуза
```

Рис. 11: Пример вывода названия с результатом на тестовых изображениях

Также можно загрузить только одно изображение и не с диска, а сразу с компьютера или ноутбука в google colab (рисунок 12)

```
[ ]  your_image_name = 'many many f.jpg' # здесь указывается название загруженной картинки
     img = image.load_img(your_image_name, target_size=(img_height, img_width))
     plt.imshow(img)

     img = image.img_to_array(img)
     img = np.expand_dims(img,axis=0)
     answer = model.predict(img)

     if answer >= 0.5:
       plt.title('Медуза')
       print(f'{your_image_name}: Медуза')
     else:
       plt.title('Рыбка')
       print(f'{your_image_name}: Рыбка')
     plt.show()
```

Рис. 12: Алгоритм для загрузки своего изображения

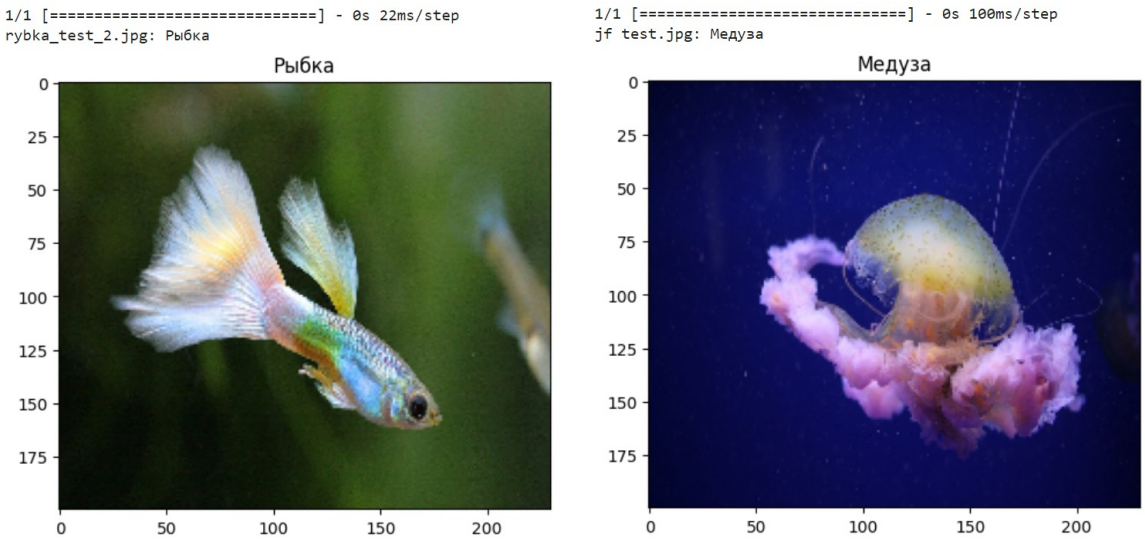

Рис. 13: Пример вывод для одного изображения с рыбкой и медузой

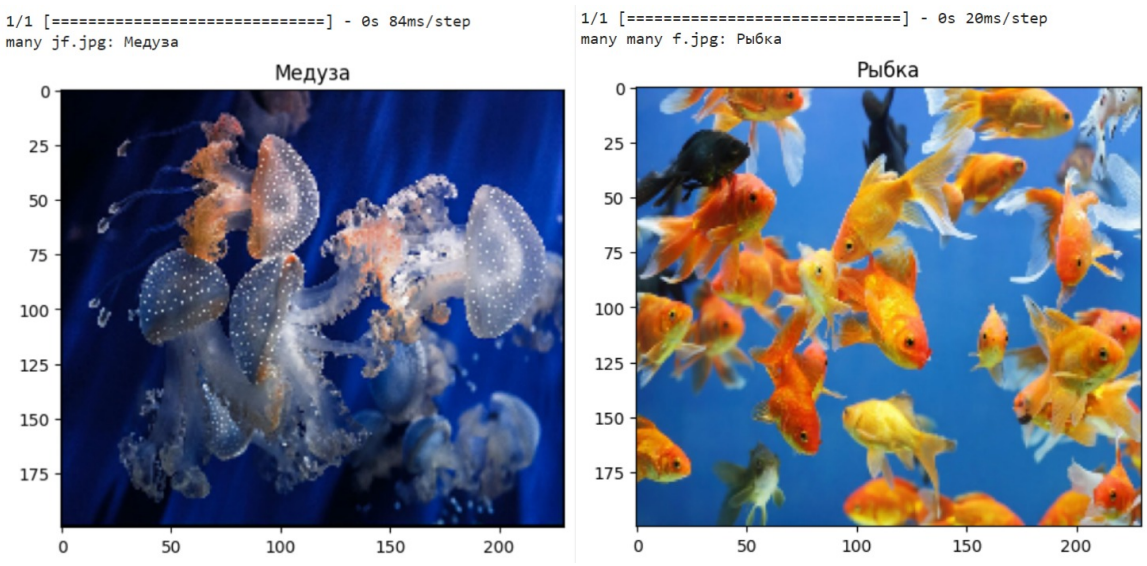

Рис. 14: Пример вывод для одного изображения с несколькими рыбками и медузами

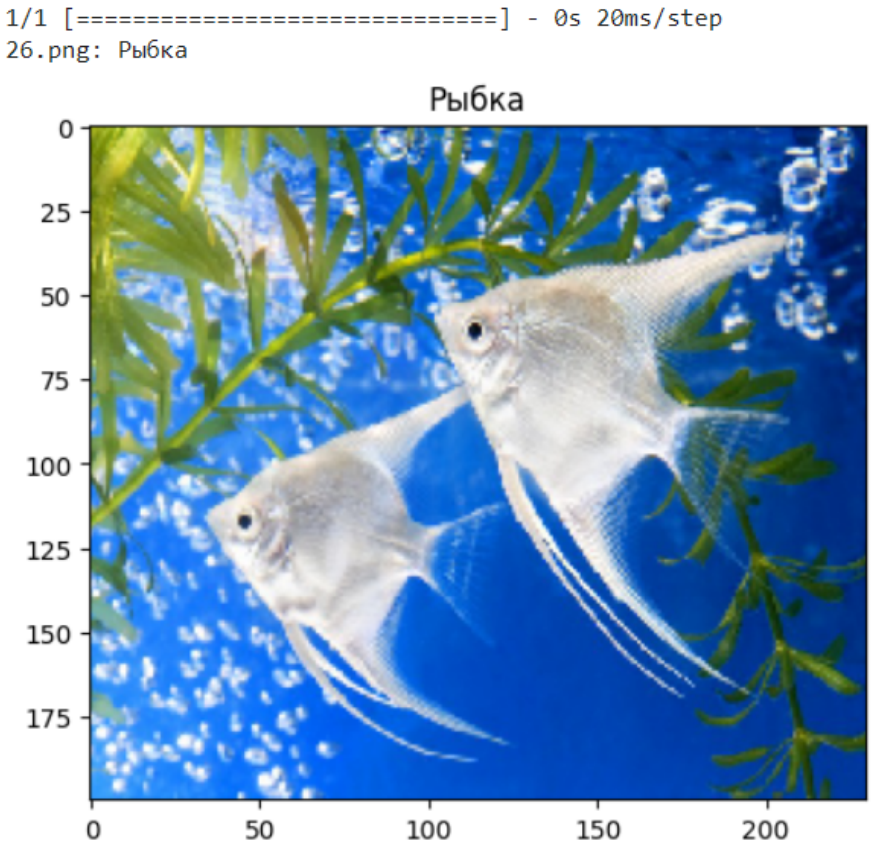

Рис. 15: Еще один красивый пример с рыбкой

4 Заключение

В ходе этого проекта были проведены обширные исследования на темы: формирование датасета, создание и обучение нейронной сети для решения задач классификации изображений.

Для набора изображений под любую задачу была написана специальная программа для скачивания нужных картинок из интернета сразу в нужные папки и под нужными названиями.

Для нейросети были проверены различные комбинации параметров, количества слоев, количества фоторафий в каждом классе для получения наивысшей точности модели классификации изображений на 2 класса: аквариумные рыбки и медузы. И для тех, и для других были взяты фотографии не просто на однотонном фоне, а в реальных условиях под водой. Итоговая точность на проверочной выборке - 90%.

Также были реализованы подпрограммы для проверки - на фотографиях из тестовой папки, а также для загрузки любой фотографии непосредственно в саму среду разработки. Можно заметить, что модель с высокой вероятностью определяет правильно не только единственный объект в кадре, как на рисунках 13 и 15, но и когда на изображении несколько объектов одного вида, как на рисунке 14.

Предлагаемая нейросеть может найти свое практическое применение в гидробиологии, позволяя, к примеру, при установке на подводные фотоловушки, производить автоматический подсчет плотности тех или иных представителей фауны на конкретном участке подводной местности.

Кроме того, можно отметить, что в результате работы над проектом были усовершенствованы навыки понимания работы нейросетей и их устройства, необходимые для дальнейшего написания оригинальных моделей для потенциального более широкого их применения.

Список используемой литературы

[1] Chromedriver. https://chromedriver.chromium.org/getting-started.

[2] Tensorflow. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.

[3] Документация keras. https://keras.io/api/.