

Министерство науки и высшего образования РФ
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра информационных систем и сетей
Направление 09.03.01 «Информатика и вычислительная техника»

Выпускная квалификационная работа
(дипломная работа)

**«Разработка нейронной сети для поиска фотоматериалов пригодных для
разметки набора данных»**

студента группы ОБ-09.03.01.01-41
Бурановой Алины Андреевны

(подпись)

Научный руководитель:
Трусов Алексей Сергеевич,
Ст.преподаватель _____

(подпись)

Заведующий кафедрой:
Дюгуров Денис Владимирович,
к.т.н, доцент _____

(решение о допуске)

«____» _____ 2024 г.

Ижевск, 2024

Содержание

Введение.....	3
Глава I. Предметная область.....	4
§ 1. Цель и задачи.....	4
§ 2. Условия пригодности изображений.....	4
§ 3. Нейронная сеть и ее типы.....	6
§ 4. Выбор инструментов разработки.....	7
Глава II. Реализация.....	9
§ 1. Описание.....	9
§ 2. Архитектура.....	9
§ 3. Разработка.....	9
§ 3.1. Парсинг изображений.....	10
§ 3.2. Обучение нейросети.....	11
§ 3.3. Веб-интерфейс.....	17
§ 4. Оценка обучения нейросети.....	19
§ 5. Результат.....	21
Литература.....	22
Приложение.....	24
Приложение 1. Компонент parser.py.....	24
Приложение 2. Компонент ai.py.....	26
Приложение 3. Компонент app.py.....	30
Приложение 4. Компонент index.html.....	34
Приложение 5. Компонент images.html.....	34
Приложение 6. Компонент model_params.html.....	34
Приложение 7. Компонент results.html.....	35
Приложение 8. Компонент requirements.txt.....	35

Введение

В современном мире нейронные сети играют важнейшую роль во многих областях, от обработки информации до создания искусственного интеллекта. В нашей работе мы покажем, как можно использовать нейросети для автоматизации процесса сбора изображений.

Данный проект имеет огромное значение и необходимость по ряду причин:

1. Он способен существенно ускорить и автоматизировать процесс сбора изображений для разметки данных. Это особо важно в контексте машинного обучения и искусственного интеллекта, где качественные данные для обучения моделей являются ключом к успешному проекту.
2. Благодаря веб-интерфейсу проект доступен для широкого круга пользователей, не требуя от них специализированных навыков или знаний.
3. Такой проект может стать полезным инструментом для исследователей, студентов, разработчиков и других специалистов, работающих в области машинного обучения и искусственного интеллекта.

Глава I. Предметная область

§ 1. Цель и задачи

Цель:

Разработать веб-ориентированное приложение на основе искусственного интеллекта для автоматической классификации и сортировки изображений, собранных с помощью парсера¹, согласно запросу пользователя.

Задачи:

1. Разработать веб-интерфейс для ввода пользовательских запросов, параметров обучения модели и ручной сортировки изображений.
2. Создать парсер для сбора изображений по запросу пользователя.
3. Обучить модель на основе классифицированных изображений.
4. Реализовать функцию автоматической классификации оставшихся неразмеченных изображений моделью.
5. Предоставить пользователю набор пригодных для разметки данных изображений.

§ 2. Условия пригодности изображений

Пригодными будем считать те изображения, которые удовлетворяют условиям:

1. Искомый объект соответствует запросу;
2. Искомый объект виден полностью;
3. Качество изображения должно быть достаточно высоким, чтобы можно было легко идентифицировать объект.
4. Искомый объект является центральным элементом изображения, а не

¹ Парсер — это программа, сервис или скрипт, который собирает данные с указанных веб-ресурсов, анализирует их и выдает в нужном формате.

находится на периферии.

5. Изображение не должно содержать неприемлемого или нежелательного контента.









Пригодные	Непригодные
	
	
	
	

Таблица 1. Примеры пригодных и непригодных изображений по запросу

§ 3. Нейронная сеть и ее типы

Нейронная сеть — математическая модель, а также ее программное или аппаратное воплощение, построенная по принципу организации биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. Нейронные сети не программируются в привычном смысле этого слова, они обучаются.

Рассмотрим типы нейросетей:

- **Персептрон:** Персептрон является одним из самых простых типов нейросетей. Он состоит из одного или нескольких слоев нейронов, каждый из которых связан с предыдущим и следующим слоями. Персептроны широко применяются в задачах классификации и регрессии, например, для распознавания образов или прогнозирования цен на акции.
- **Сверточные нейросети:** Сверточные нейросети (Convolutional Neural Networks, CNN) широко используются в области компьютерного зрения. Они эффективно обрабатывают изображения и видео, распознают объекты, выполняют сегментацию и классификацию. Сверточные нейросети обладают специальными слоями, такими как сверточные слои и пулинг, которые позволяют автоматически извлекать признаки из входных данных.
- **Рекуррентные нейросети:** Рекуррентные нейросети (Recurrent Neural Networks, RNN) применяются для обработки последовательных данных, таких как временные ряды, тексты или речь. Они имеют специальные рекуррентные связи, которые позволяют учитывать

контекст и зависимости между последовательными элементами. Рекуррентные нейросети широко используются в задачах машинного перевода, генерации текста и анализа сентимента.

- **Другие типы нейросетей:** Существует множество других типов нейросетей, включая глубокие нейронные сети (Deep Neural Networks, DNN), автокодировщики (Autoencoders), генеративные сети (Generative Adversarial Networks, GAN) и многое другое. Каждый из этих типов имеет свои уникальные особенности и применение в разных областях.

В данном проекте мы будем использовать сверточную нейросеть, так как она лучше всего подходит для задач классификации изображений.

§ 4. Выбор инструментов разработки

- **Python** - это высокоуровневый язык программирования общего назначения. Его используют для разработки веб-версий программ, приложений, игр, а также для создания нейросетей, тестирования и научных исследований. Python динамически типизируется и собирает мусор. Он поддерживает несколько парадигм программирования, включая структурированное (в частности, процедурное), объектно-ориентированное и функциональное программирование.
- **HTML** - стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. С помощью HTML можно создавать разные конструкции, изображения и другие объекты, такие как интерактивная веб-форма, встраивать их в отображаемую страницу.
- **SQL** - декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами

данных.

- **Flask** - фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков - минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Глава II. Реализация

§ 1. Описание

Процесс начинается с ввода пользователем запроса, после которого активируется парсер для сбора максимального количества изображений, соответствующих данному запросу. Затем пользователь проводит визуальный анализ и сортировку изображений на два класса: приемлемые и неприемлемые. Изображения сохраняются локально на компьютер пользователя. На этих классах происходит обучение модели. Обученная модель автоматически сортирует оставшиеся неотмеченные изображения. В конечном итоге пользователь получает набор пригодных для разметки данных изображений.

§ 2. Архитектура

```
myproject
├── static
│   └── images
├── templates
│   ├── ai.html
│   ├── images.html
│   └── index.html
├── venv
├── ai.py
├── app.py
├── parser.py
├── database.db
└── equipments.txt
```

§ 3. Разработка

Перед началом работы перейдем в папку нашего будущего проекта, создадим и активируем там наше новое виртуальное окружение²:

```
python -m venv venv
.\venv\Scripts\activate
```

² Виртуальное окружение - это изолированная среда, которая позволяет установить и работать с определенными версиями программного обеспечения без вмешательства в основную систему.

Это позволит нам устанавливать пакеты и зависимости, не влияя на другие проекты или системную установку Python.

§ 3.1. Парсинг изображений

Для сбора изображений напомним функцию `parse_with_selenium`, принимающую в качестве входного параметра запрос.

Изображения будем извлекать из открытых источников, а именно поисковых систем Google и Yandex. Для этого внутри функции мы создадим список `browsers`, который хранит информацию о каждом из этих поисковиков:

```
browsers = [
    {
        'name': 'Chrome',
        'url': f"https://google.com/search?tbm=isch&q={query}",
        'img_class': 'YQ4gaf',
        'button_selector': '.LZ4I'
    },
    {
        'name': 'Yandex',
        'url': f"https://yandex.ru/images/search?text={query}",
        'img_class': 'ContentImage-Image
ContentImage-Image_clickable',
        'button_selector':
        '.Button2.Button2_width_max.Button2_size_l.Button2_view_action.F
etchListButton-Button'
    }
]
```

Здесь `name` - это имя поисковика, `url` - адрес запроса, `img_class` - класс тега `img`, который необходим для отсеивания лишних изображений, а `button_selector` - класс кнопки, который используется для автоматического нажатия кнопки в процессе работы программы. Позже рассмотрим, для чего нам необходимо каждое из этих значений.

Пройдемся по каждому элементу списка `browsers`. В начале цикла создаем экземпляр веб-драйвера `Chrome`. Затем, с помощью метода `.get()`, этот веб-драйвер переходит на URL, который задан в словаре `browser` под

ключом `url`. Этот URL представляет собой адрес поискового запроса в определенной поисковой системе:

```
driver = webdriver.Chrome()
driver.get(browser['url'])
```

Далее с помощью веб-драйвера пролистываем страницу до самого низа, параллельно нажимая кнопку для показа большего количества изображений, если она существует. Ищем кнопку по ключу `button_selector` в словаре `browser`. Когда мы дошли до конца страницы, выполняем следующий код:

```
html = driver.page_source
soup = BeautifulSoup(html, 'html.parser')

tags = soup.find_all('img', {'class': browser['img_class']})
tags = [tag for tag in tags if ' '.join(tag.get('class')) ==
browser['img_class']]

srcs.extend([tag.get('src') for tag in tags if not
tag.get('src').startswith('data:image')])
```

Здесь мы получили HTML и вытащили из него все изображения с классом, указанным в ключе `img_class` словаря `browser`. По завершении цикла исключим из полученного списка изображений дублирующиеся:

```
unique_srcs = list(set(srcs))
```

Функция `parse_with_selenium` возвращает список изображений `unique_srcs`.

Полный код модуля представлен в приложении 1.

§ 3.2. Обучение нейросети

Для написания нейронной сети будем использовать библиотеку TensorFlow. Рассмотрим самые необходимые функции:

```
def load_and_preprocess_images(path, batch_size):
    # Указываем путь до папки с датасетом
    dataset_dir = pathlib.Path(path)

    train_ds = tf.keras.utils.image_dataset_from_directory(
```

```

        dataset_dir,
        validation_split=0.2,
        subset="training",
        seed=123,
        image_size=(IMG_WIDTH, IMG_HEIGHT),
        batch_size=batch_size
    )

    val_ds = tf.keras.utils.image_dataset_from_directory(
        dataset_dir,
        validation_split=0.2,
        subset="validation",
        seed=42,
        image_size=(IMG_WIDTH, IMG_HEIGHT),
        batch_size=batch_size
    )

    class_names = train_ds.class_names
    print(f"Class names: {class_names}")
    return train_ds, val_ds, class_names

```

Функция `load_and_preprocess_images` принимает на вход путь до датасета³ и размер батча⁴. Она разбивает датасет на 2 выборки: тренировочную и валидационную⁵. Параметр `validation_split` со значением `0.2` означает, что валидационная выборка будет составлять 20% от общего набора данных. Такой процент обеспечивает достаточное количество данных для обучения, а также достаточное количество данных для проверки обобщающей способности модели. Это значение мы оставим лишь для оценки обучения нашей модели, а далее изменим его на `0`, тем самым обеспечив больший размер тренировочной выборки. Параметр `seed` используется для инициализации генератора случайных чисел. `image_size` определяет размер изображений, а `batch_size` - размер батча.

```

# Кэшируем изображения
def cash_images(train_ds, val_ds):
    autotune = tf.data.AUTOTUNE

```

³ Датасет, или набор данных – это совокупность данных, систематизированных в определенном формате, представляющих собой базовый элемент для работы с данными во многих отраслях.

⁴ Размер батча - это количество обучающих примеров, используемых за одну итерацию алгоритма машинного обучения.

⁵ Валидационная выборка — набор данных, который используется в процессе разработки модели машинного обучения для подбора оптимального набора гиперпараметров (для проверки того, как работает модель).

```

train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=autotune)
val_ds = val_ds.cache().prefetch(buffer_size=autotune)
return train_ds, val_ds

```

Функция `cash_images` принимает на вход датасеты изображений и кэширует⁶ их. Это позволяет ускорить обучение нейросети и полезно при аугментации⁷ данных.

```

# Создаем модель
def create_model(class_names):
    num_classes = len(class_names)

    model = Sequential([
        layers.Input(shape=(IMG_WIDTH, IMG_HEIGHT, 3)),
        layers.Rescaling(1. / 255),

        # аугментация
        layers.RandomFlip('horizontal'),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
        layers.RandomContrast(0.1),

        layers.Conv2D(16, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),

        layers.Conv2D(32, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),

        layers.Conv2D(64, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),

        # регуляризация
        layers.Dropout(0.2),

        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes)
    ])

    return model

```

Функция `create_model` принимает на вход список `class_names`, содержащий имена классов, на основе которого определяет их количество.

⁶ Кэширование — это процесс, при котором элементы сайта или страница целиком сохраняются в кэше (хранилищах быстрого доступа).

⁷ Аугментация данных – это процесс искусственного генерирования новых данных на основе уже существующих.

Затем она создает модель нейронной сети с использованием технологии Sequential из библиотеки Keras:

1. Сначала методом `Input` вводится форма входных данных, которая ожидает изображения указанного размера и 3 цветовых канала;
2. `Rescaling` масштабирует пиксели изображения в диапазон от 0 до 1;
3. Производится аугментация данных с помощью случайного отражения изображения по горизонтали, случайного вращения, приближения и изменения контраста на 10%;
4. Добавляются три блока сверточных слоев, каждый из которых состоит из сверточного слоя с нелинейной функцией активации ReLU и слоя максимального пулинга⁸;
5. Добавляется слой `Dropout` для регуляризации⁹, что помогает предотвратить переобучение;
6. Добавляется слой `Flatten`, который преобразует многомерные данные в одномерный вектор;
7. С помощью метода `Dense` добавляются два полносвязных слоя: первый с 128 узлами и функцией активации ReLU, второй с количеством узлов, равным количеству классов.

Функция возвращает построенную модель.

```
# Компилируем модель
def compile_model(model):
    model.compile(
        optimizer='adam',
```

⁸ Слой пулинга представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размера 2×2) уплотняется до одного пикселя, проходя нелинейное преобразование.

⁹ Регуляризация — это техника, используемая для предотвращения переобучения модели на обучающих данных. Суть регуляризации заключается в введении дополнительных ограничений или штрафов на величину и/или сложность модели.

```

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )
    return

```

Функция `compile_model` компилирует модель с использованием оптимизатора Adam, функции потерь и метрики точности. Это базовая конфигурация для обучения моделей классификации в TensorFlow.

```

# Выводим вычисления
def print_model_summary(model):
    return model.summary()

```

Функция `print_model_summary` принимает в качестве параметра модель и возвращает её сводку: количество слоев в модели, количество параметров на каждом слое и общее количество параметров.

```

# Тренируем модель
def train_model(model, train_ds, val_ds, epochs):
    history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=epochs
    )

    # Отображаем процесс обучения
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')

```

```
plt.title('Training and Validation Loss')
plt.show()

return
```

Функция `train_model` выполняет обучение модели на тренировочном наборе данных и проверяет ее на валидационном наборе данных в течение определенного количества эпох. Она также отслеживает историю обучения, включая точность и потери обучения и валидации, по которым, после обучения, строит графики. Эти графики помогают оценить эффективность обучения модели, позволяя наблюдать за изменениями точности и потерь по мере прохождения эпох.

```
# Загружаем тестовое изображение и возвращаем его в виде массива NumPy
def load_test_image(image_path):
    img = keras.preprocessing.image.load_img(image_path,
target_size=(IMG_HEIGHT, IMG_WIDTH))
    img_array = keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    return img, img_array
```

Функция `load_test_image` загружает тестовое изображение по указанному пути. Она приводит изображение к необходимому размеру, преобразует его в массив и расширяет размерность. На выходе получаем оригинальное изображение и его массивное представление.

```
# Вычисляем вероятность
def predict_model(model, img_array):
    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])
    return score
```

Функция `predict_model` применяет модель к массиву изображения, получая в качестве результата прогнозы. Затем преобразовывает выходные данные модели в вероятности. Возвращается вектор вероятностей, каждый элемент которого соответствует вероятности принадлежности изображения к определенному классу.

Полный код модуля представлен в приложении 2.

§ 3.3. Веб-интерфейс

Теперь создадим основной файл Python, который будет использовать фреймворк Flask для создания веб-интерфейса. Это позволит нам связать ранее созданные модули Python и включить их в HTML. В файле создадим различные функции обработки маршрута. Опишем каждый маршрут.

Маршрут `/`: Когда пользователь отправляет GET запрос, функция возвращает главную страницу с полем ввода запроса. Если же был отправлен POST запрос, функция берет введенный пользователем запрос и передает его в функцию `parse_with_selenium`, которая парсит изображения. Изображения затем сохраняются в базу данных. После этого пользователь перенаправляется на маршрут `/images`.

Главная

Рис. 1. Страница `index.html`

Маршрут `/images`: Для GET запросов, функция возвращает страницу с изображениями, каждое из которых находится внутри чекбокса¹⁰, а также показывает, сколько еще осталось набрать изображений для каждого класса. Если изображений достаточно, они загружаются на компьютер так, чтобы отмеченные и неотмеченные изображения находились в разных папках. Затем пользователь перенаправляется на маршрут `/model_params`. Для POST запросов, функция считывает отмеченные и неотмеченные изображения, в зависимости от чего обновляет значение булевой переменной, отвечающей за

¹⁰ Чекбокс (флажок, флаговая кнопка) — это элемент графического пользовательского интерфейса, который позволяет пользователю управлять параметром с двумя состояниями: включено и отключено.

их приемлемость в базе данных, после чего перенаправляет пользователя на тот же маршрут /images, но с значением `page=page + 1`.

Результаты поиска по запросу: Машина вид сбоку

Выберите непригодные изображения:

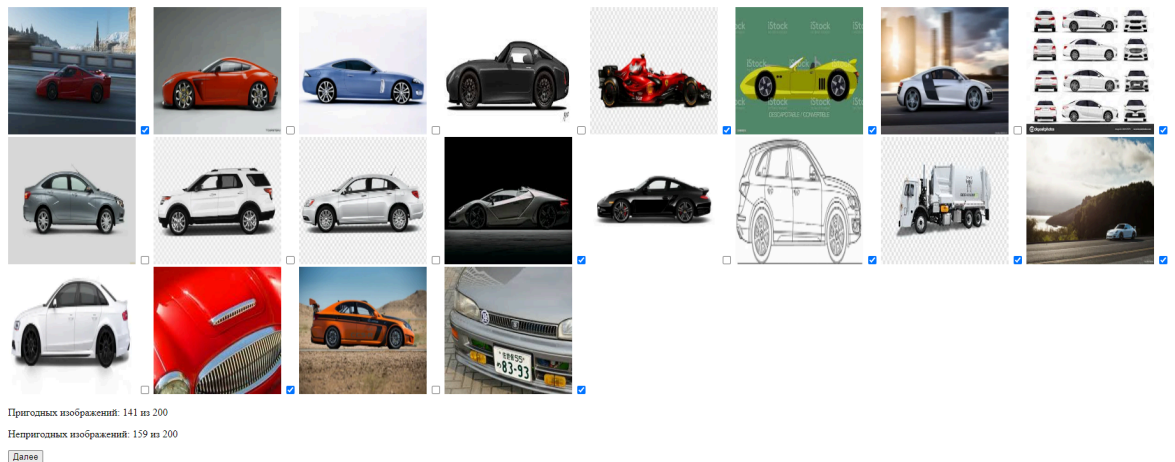


Рис. 2. Страница images.html

Маршрут /model_params: Если был отправлен GET запрос, код просто возвращает страницу /model_params с полями ввода `epochs` и `batch_size`. Когда пользователь отправляет POST запрос, этот код берет параметры, введенные пользователем, и сохраняет их в сессии. После этого, пользователь перенаправляется на страницу /results.

Настройка модели

Количество эпох:

Размер пакета: ▼

Рис. 3. Страница model_params.html

Маршрут /results: Когда этот маршрут активируется, функция извлекает путь к изображениям и параметры обучения, которые были сохранены в сессии, затем тренирует модель на этих изображениях. Для каждого изображения она вычисляет прогнозируемый класс и вероятность

этого класса, затем перемещает изображение в соответствующую директорию в зависимости от его предсказанного класса. Наконец, функция возвращает страницу результатов.

Парсинг завершен!

Рис. 4. Страница results.html

Полный код можно посмотреть в приложении 3

§ 4. Оценка обучения нейросети

Рассмотрим графики, полученные в результате обучения:

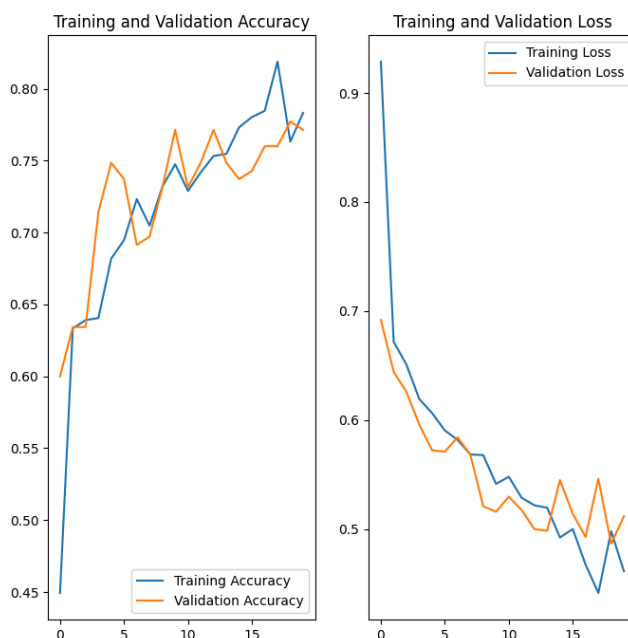


Рис. 5. Графики тренировочных и валидационных точности и потерь

Анализируя их, мы наблюдаем стабильное увеличение точности и снижение потерь. Это свидетельствует о том, что модель успешно обучается: способность к предсказанию улучшается, количество ошибок уменьшается. Наблюдаемые незначительные колебания значений связаны с процессом обучения, в ходе которого модель постоянно обновляет и корректирует свои параметры.

Restored model, accuracy: 83.45%

Рис. 6. Точность модели

Численное значение точности в результате обучения равно 83,45% (рис. 6), что можно считать достаточно хорошим результатом, учитывая небольшой объем использованных данных.

Теперь необходимо проверить работу обученной нейронной сети на практике. Для этого выберем пригодное изображение, которое модель не встречала ранее, приведем его к необходимому размеру и представим модели:

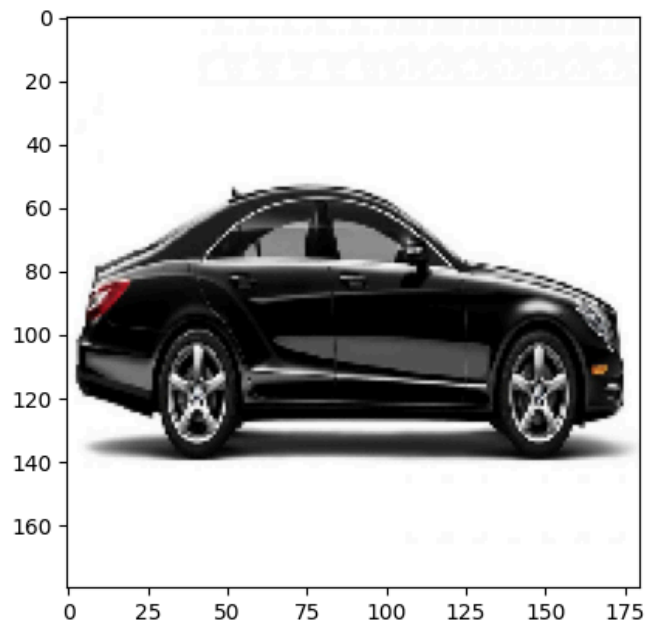


Рис. 7. Тестовое изображение

Predicted: accept 93.67%

Рис. 8. Предсказание на тестовом изображении

По результатам, представленным выше можно сделать вывод, что обучение нашей модели прошло успешно и она готова к дальнейшему использованию.

§ 5. Результат

В результате работы на компьютере пользователя должна появиться папка с пригодными изображениями.

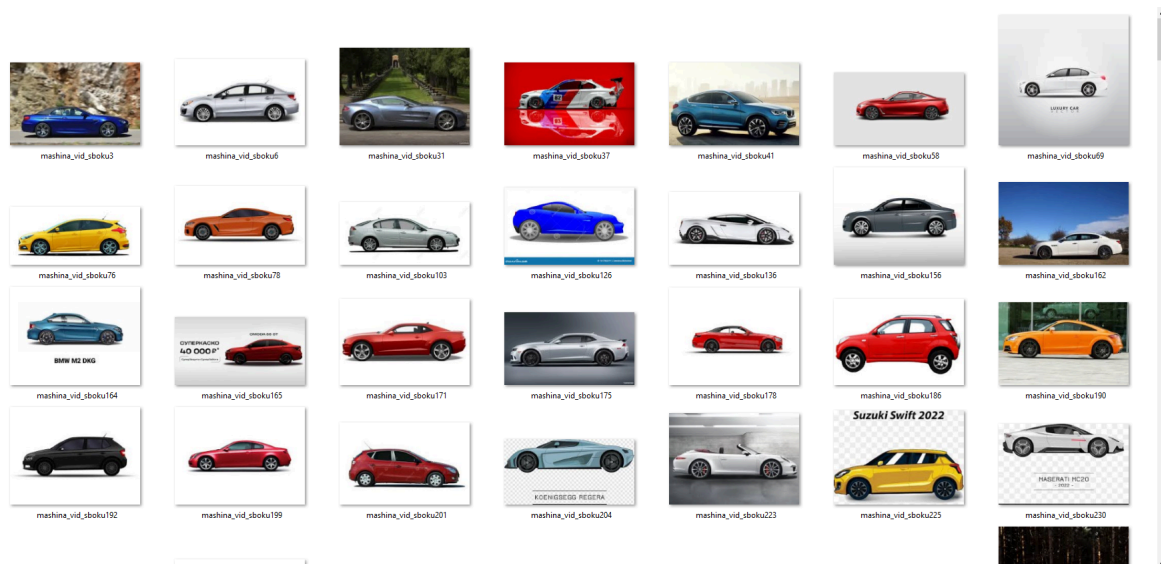


Рис. 9. Папка с датасетом

Литература

1. Python. Python Software Foundation : офиц. сайт. URL: <https://www.python.org> (дата обращения: 17.05.2024).
2. Selenium : офиц. сайт. URL: <https://www.selenium.dev> (дата обращения: 17.05.2024).
3. Доусон, М. Программируем на Python / Майкл Доусон. — Пер. с англ. — Санкт-Петербург : Питер, 2022. — 416 с.
4. Жерон, О. Руководство по машинному обучению с помощью Scikit-Learn и TensorFlow: Концепции, инструменты и техники для создания интеллектуальных систем / Орельен Жерон. — Пер. с фр. — Москва : Вильямс, 2020. — 1040 с.
5. Лекун, Я. Как учится машина: Революция в области нейронных сетей и глубокого обучения / Ян Лекун. — Пер. с фр. — Москва : Альпина ПРО, 2021. — 335 с.
6. Вьюгин, В. В. Математические основы машинного обучения и прогнозирования / Владимир Вячеславович Вьюгин. — Москва : МЦНМО, 2022. — 399 с.
7. Рашка, С. Python и машинное обучение. Машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2 / Рашка Себастьян, Мирджалили Вахид. — Киев: Диалектика-Вильямс, 2021. — 848 с.
8. Шилдс, У. SQL: быстрое погружение / SQL: быстрое погружение. — Пер. с англ. Павлов А. — Санкт-Петербург : Питер, 2022. — 223 с.

9. Grinberg, M. SQLAlchemy 2 In Practice: Learn to program relational databases in Python step-by-step / Miguel Grinberg. — Kindle Edition, 2023. — 274 p.
10. Grinberg, M. The Flask Mega-Tutorial, Part I: Hello, World! In How LLMs Work, Explained Without Math. Retrieved on May 25, 2024, from <https://www.novtex.ru/>. Originally published on December 3, 2023.

Приложение

Приложение 1. Компонент parser.py

```
import concurrent.futures # Библиотека для потоков
import os # Библиотека для работы с файлами
import random # Библиотека для генерации рандомных чисел
import time # Библиотека для работы с задержками
import requests # Библиотека для отправки запросов к сайту
from bs4 import BeautifulSoup # Библиотека для парсинга

# Библиотека для веб-драйвера
from selenium import webdriver
from selenium.common import TimeoutException,
ElementNotInteractableException
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as ec

# Парсим изображения с помощью Selenium
def parse_with_selenium(query):
    query = query.replace(' ', '+')
    length = 0
    srcs = []
    browsers = [
        {
            'name': 'Chrome',
            'url':
f"https://google.com/search?tbm=isch&q={query}",
            'img_class': 'YQ4gaf',
            'button_selector': '.LZ4I'
        },
        {
            'name': 'Yandex',
            'url':
f"https://yandex.ru/images/search?text={query}",
            'img_class': 'ContentImage-Image
ContentImage-Image_clickable',
            'button_selector':
'.Button2.Button2_width_max.Button2_size_1.Button2_view_action.F
etchListButton-Button'
        }
    ]

    for browser in browsers:
        driver = webdriver.Chrome()
        driver.get(browser['url'])

        while True:
```



```

        last_count = length
        length = driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);"
                                "var
lenOfPage=document.body.scrollHeight;"
                                "return lenOfPage;")
        time.sleep(0.8)
        if last_count == length:
            try:
                button = WebDriverWait(driver, 0.5).until(
ec.presence_of_element_located((By.CSS_SELECTOR,
browser['button_selector'])))
                button.click()
            except TimeoutException:
                print(f"{browser['name']} scrolling is
complete.")
                break
            except ElementNotInteractableException:
                print(f"{browser['name']} scrolling is
complete.")
                break

        # Получаем HTML и преобразовываем его в объект
BeautifulSoup
        html = driver.page_source
        soup = BeautifulSoup(html, 'html.parser')

        # Вытаскиваем все теги <img>
        tags = soup.find_all("img", {'class':
browser['img_class']})
        tags = [tag for tag in tags if ' '.join(tag.get('class'))
== browser['img_class']]

        # Из тегов вытаскиваем атрибуты "src", пропуская формат
изображений base64
        srcs.extend([tag.get('src') for tag in tags if not
tag.get('src').startswith('data:image')])

        # Убираем дублирующиеся ссылки
        unique_srcs = list(set(srcs))
        random.shuffle(srcs)

        print(f"Images found: {len(unique_srcs)}")
        return unique_srcs

# Транслитерация и замена СИМВОЛОВ
def transliterate(query):
    rus_to_eng = {

```

```

        'a': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е':
'e', 'ё': 'yo',
        'ж': 'zh', 'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л':
'l', 'м': 'm',
        'н': 'n', 'о': 'o', 'п': 'p', 'р': 'r', 'с': 's', 'т':
't', 'у': 'u',
        'ф': 'f', 'х': 'h', 'ц': 'ts', 'ч': 'ch', 'ш': 'sh', 'щ':
'sch', 'ъ': '',
        'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya', ' ':
' ',
    }
    title = ''.join(rus_to_eng.get(c, c) for c in query.lower())
    return title

```

```

# Загрузка изображений локально, используя потоки
def streaming_download(path, title, srcs, numbers=None):
    def download_image(url, i):
        response = requests.get(url)
        with open(f'{path}/{title}{i}.jpg', 'wb') as f:
            f.write(response.content)

    if not os.path.exists(path):
        os.makedirs(path)

    if numbers is None:
        numbers = range(len(srcs))

    with concurrent.futures.ThreadPoolExecutor() as executor:
        for number, src in zip(numbers, srcs):
            executor.submit(download_image, src, number)
    return

```

Приложение 2. Компонент ai.py

```

import pathlib # Библиотека для работы с директориями
from matplotlib import pyplot as plt # Библиотека для графиков
import numpy as np # Библиотека для вычислений

# Библиотека для нейронной сети
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential

IMG_WIDTH, IMG_HEIGHT = 180, 180

def load_and_preprocess_images(path, batch_size):
    # Указываем путь до папки с датасетом
    dataset_dir = pathlib.Path(path)

```

```

train_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(IMG_WIDTH, IMG_HEIGHT),
    batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(IMG_WIDTH, IMG_HEIGHT),
    batch_size=batch_size
)

class_names = train_ds.class_names
print(f"Class names: {class_names}")
return train_ds, val_ds, class_names

# Кэшируем изображения
def cash_images(train_ds, val_ds):
    autotune = tf.data.AUTOTUNE
    train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=autotune)
    val_ds = val_ds.cache().prefetch(buffer_size=autotune)
    return train_ds, val_ds

# Создаем модель
def create_model(class_names):
    num_classes = len(class_names)

    model = Sequential([
        layers.Input(shape=(IMG_WIDTH, IMG_HEIGHT, 3)),
        layers.Rescaling(1. / 255),

        # аугментация
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
        layers.RandomContrast(0.1),

        layers.Conv2D(16, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),

        layers.Conv2D(32, 3, padding='same', activation='relu'),

```

```

        layers.MaxPooling2D(),

        layers.Conv2D(64, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),

        # регуляризация
        layers.Dropout(0.2),

        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes)
    ])

    return model

# Компилируем модель
def compile_model(model):
    model.compile(
        optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )
    return

# Выводим вычисления
def print_model_summary(model):
    return model.summary()

# Тренируем модель
def train_model(model, train_ds, val_ds, epochs):
    history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=epochs
    )

    # Отображаем процесс обучения
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))

```

```

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

return

# Сохраняем веса
def save_model_weights(model):
    model.save_weights('my_model.weights.h5')
    print(f"Saved model to disk")
    return

# Загружаем веса
def load_model_weights(model):
    model.load_weights('my_model.weights.h5')
    return

# Получаем веса
def get_model_weights(model, train_ds):
    loss, acc = model.evaluate(train_ds, verbose=2)
    print(f'Restored model, accuracy: {acc * 100:5.2f}%')
    return

# Загружаем тестовое изображение и возвращаем его в виде массива
NumPy
def load_test_image(image_path):
    img = keras.preprocessing.image.load_img(image_path,
target_size=(IMG_HEIGHT, IMG_WIDTH))
    img_array = keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    return img, img_array

# Вычисляем вероятность
def predict_model(model, img_array):
    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])
    return score

```

```

# Выводим результат
def output_result(class_names, score, img):
    print(f"Predicted: {class_names[np.argmax(score)]}")
    {np.max(score) * 100:5.2f}%")
    plt.imshow(img, cmap='Accent')
    plt.show()
    return

# Загружаем датасет, создаем модель и тренируем
def create_and_train_model(path, batch_size, epochs):
    # Загружаем и подготавливаем датасет
    train_ds, val_ds, class_names =
load_and_preprocess_images(path, batch_size)
    train_ds, val_ds = cash_images(train_ds, val_ds)

    # Создаем модель и тренируем
    model = create_model(class_names)
    compile_model(model)
    print_model_summary(model)
    train_model(model, train_ds, val_ds, epochs)

    # Получаем веса
    get_model_weights(model, train_ds)
    return model, class_names

# Загружаем изображение и получаем предсказание
def get_image_score(model, path):
    # Загружаем изображение и преобразуем его в массив
    img, image_array = load_test_image(path)

    # Вычисляем вероятность
    score = predict_model(model, image_array)
    return img, score

```

Приложение 3. Компонент app.py

```

import os
import shutil
import numpy as np
from flask import Flask, render_template, request, redirect,
url_for, session
from flask_sqlalchemy import SQLAlchemy
import ai
import parser

app = Flask(__name__)
app.secret_key = '12345'

```

```

app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:///C:/Users/SashaAlina/PythonProjects/myproject/database
.db'
db = SQLAlchemy(app)

query = ''
CLASS_LENGTH = 200

class Image(db.Model):
    id = db.Column(db.Integer, primary_key=True,
autoincrement=True)
    src = db.Column(db.String(200), nullable=False)
    acceptability = db.Column(db.Boolean)

@app.route('/', methods=['GET', 'POST'])
def index():
    global query
    db.create_all()
    if request.method == 'POST':
        query = request.form['query']

        # Парсим изображения
        srcs = parser.parse_with_selenium(query)

        # Переносим данные из класса в подкласс db.Model
        for src in srcs:
            image = Image(src=src)
            db.session.add(image)

        # Сохраняем сессию
        db.session.commit()
        return redirect(url_for('images', page=1))

    return render_template('index.html')

@app.route('/images', methods=['GET', 'POST'])
def images():
    # Получаем номер страницы из запроса
    page = request.args.get('page', type=int)

    # Получаем 20 элементов на указанной странице
    imgs = Image.query.paginate(page=page, per_page=20,
error_out=False).items

    # Обращаемся к базе данных
    unaccept_results =
db.session.query(Image).filter_by(acceptability=False).all()

```

```

    accept_results =
db.session.query(Image).filter_by(acceptability=True).all()
    other_results =
db.session.query(Image).filter_by(acceptability=None).all()

    # Получаем ссылки id изображений
    srcs_unaccept = [result.src for result in unaccept_results]
    srcs_accept = [result.src for result in accept_results]
    srcs_other = [result.src for result in other_results]

    id_unaccept = [result.id for result in unaccept_results]
    id_accept = [result.id for result in accept_results]
    id_other = [result.id for result in other_results]

    if request.method == 'POST':
        for img in imgs:
            if request.form.get(str(img.id)):
                img.acceptability = False
            else:
                img.acceptability = True
        # Сохраняем сессию
        db.session.commit()
        return redirect(url_for('images', page=page + 1))

    # Набираем нужное количество изображений для обучения
    if len(unaccept_results) >= CLASS_LENGTH and
len(accept_results) >= CLASS_LENGTH:
        # Загружаем датасет для обучения
        title = parser.transliterate(query)
        session['title'] = title

        path =
f'C:/Users/SashaAlina/PythonProjects/myproject/static/images/{ti
tle}/'
        session['path'] = path

        parser.streaming_download(path + 'unaccept', title,
srcs_unaccept, id_unaccept)
        parser.streaming_download(path + 'accept', title,
srcs_accept, id_accept)

        # Загружаем остальные изображения
        parser.streaming_download(path, title, srcs_other,
id_other)

        # Очищаем БД, она нам больше не пригодится
        Image.query.delete()
        db.session.commit()

        return redirect(url_for('params'))

```



```

        return render_template('images.html', imgs=imgs,
                                len_accept=len(accept_results),
                                len_unaccept=len(unaccept_results), len_class=CLASS_LENGTH)

@app.route('/model_params', methods=['GET', 'POST'])
def params():
    if request.method == 'POST':
        epochs = request.form['epochs']
        batch_size = request.form['batch_size']

        session['epochs'] = epochs
        session['batch_size'] = batch_size
        return redirect(url_for('results'))

    return render_template('model_params.html')

@app.route('/results')
def results():
    path = session.get('path')
    epochs = int(session.get('epochs'))
    batch_size = int(session.get('batch_size'))

    # Тренируем модель на приемлемых и неприемлемых изображениях
    model, class_names = ai.create_and_train_model(path,
                                                    batch_size, epochs)

    for filename in os.listdir(path):
        if filename.endswith('.jpg'):
            image_path = os.path.join(path, filename)
            img, score = ai.get_image_score(model, image_path)
            print('Prediction score: {} ({:.2f}%)' .format(
                class_names[np.argmax(score)],
                100 * np.max(score)))

            if class_names[np.argmax(score)] == 'unaccept':
                new_path = path + 'unaccept/'
                print(new_path)
            else:
                new_path = path + 'accept/'
                print(new_path)
            shutil.move(image_path, new_path)

    return render_template('results.html')

if __name__ == '__main__':
    app.run(debug=True, port=8000)

```

Приложение 4. Компонент index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Главная</title>
</head>
<body>
<h1>Главная</h1>
<form method="POST">
  <label for="query"><input type="search" id="query"
name="query" placeholder="Введите запрос"><br></label>
  <button type="submit">Отправить</button>
</form>
</body>
</html>
```

Приложение 5. Компонент images.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Разметка изображений</title>
</head>
<body>
<h1>Результаты поиска по запросу: {{ query }}</h1>
<p>Выберите непригодные изображения:</p>
<form method="POST">
  {% for img in imgs %}
    <label for="{{ img.id }}"></label>
    <input type="checkbox" id="{{ img.id }}" name="{{ img.id
}}">
  {% endfor %}

  <p>Пригодных изображений: {{ len_accept }} из {{ len_class
}}</p>
  <p>Непригодных изображений: {{ len_unaccept }} из {{
len_class }}</p>

  <button type="submit" name="page" value="{{ page
}}">Далее</button>
</form>
</body>
</html>
```

Приложение 6. Компонент model_params.html

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <meta charset="UTF-8">
  <title>Настройка модели</title>
</head>
<body>
<h1>Настройка модели</h1>
<form method="POST">
  <label for="epochs">Количество эпох:</label>
  <input type="number" id="epochs" name="epochs" value="20"
min="5" max="50" step="5"><br>

  <label for="batch_size">Размер пакета:
    <select id="batch_size" name="batch_size">
      <option value="2">2</option>
      <option value="4">4</option>
      <option value="8">8</option>
      <option value="16">16</option>
      <option value="32">32</option>
      <option value="64" selected>64</option>
      <option value="128">128</option>
    </select>
  </label><br>

  <button type="submit" id="button">Отправить</button>
</form>
</body>
</html>

```

Приложение 7. Компонент results.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Результаты</title>
</head>
<body>
<h1>Парсинг завершен!</h1>
</body>
</html>

```

Приложение 8. Компонент requirements.txt

```

absl-py==2.1.0
asgiref==3.8.1
astunparse==1.6.3
beautifulsoup4==4.12.3
blinker==1.8.2
bs4==0.0.2
cachetools==5.3.3
certifi==2024.2.2

```

charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
contourpy==1.2.1
cyclor==0.12.1
Django==5.0.6
Flask==3.0.3
flatbuffers==24.3.25
fonttools==4.51.0
gast==0.4.0
google-auth==2.29.0
google-auth-oauthlib==1.0.0
google-pasta==0.2.0
grpcio==1.63.0
h5py==3.11.0
idna==3.7
itsdangerous==2.2.0
Jinja2==3.1.4
keras==2.13.1
kiwisolver==1.4.5
libclang==18.1.1
Markdown==3.6
markdown-it-py==3.0.0
MarkupSafe==2.1.5
matplotlib==3.8.4
mdurl==0.1.2
ml-dtypes==0.3.2
namex==0.0.8
numpy==1.26.4
oauthlib==3.2.2
opt-einsum==3.3.0
optree==0.11.0
packaging==24.0
pillow==10.3.0
protobuf==4.25.3
pyasn1==0.6.0
pyasn1_modules==0.4.0
Pygments==2.18.0
pyparsing==3.1.2
python-dateutil==2.9.0.post0
requests==2.31.0
requests-oauthlib==2.0.0
rich==13.7.1
rsa==4.9
six==1.16.0
soupsieve==2.5
sqlparse==0.5.0
tensorboard==2.13.0
tensorboard-data-server==0.7.2
tensorflow==2.16.1
tensorflow-estimator==2.13.0

```
tensorflow-intel==2.13.1
tensorflow-io-gcs-filesystem==0.31.0
termcolor==2.4.0
typing_extensions==4.5.0
tzdata==2024.1
urllib3==2.2.1
Werkzeug==3.0.3
wrapt==1.16.0
selenium==4.20.0
```