

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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
По лабораторной работе №3
Дисциплины «Основы нейронных сетей»

Выполнил:

Евдаков Евгений Владимирович

3 курс, группа ИВТ-б-о-22-1,

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

(подпись)

Руководитель практики:

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

(подпись)

Ставрополь, 2024 г.

Тема: Сверточные нейронные сети.

Цель: изучить архитектуру и принципы работы сверточных нейронных сетей.

Ход работы:

Задание 1. Сверточные нейронные сети. Распознавание марок машин

Рассмотрим применение сверточных слоев на примере задачи классификации машин по маркам: "Renault", "Mercedes", "Ferrari".

Сначала загрузим датасет из облака в colab:

```
[1] # Подключение модуля для загрузки данных из облака
import gdown

# Загрузка zip-архива с датасетом из облака на диск виртуальной машины colab
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/15/middle_fmz.zip', None, quiet=True)
```

'middle_fmz.zip'

Рисунок 1. Загрузка датасета

Далее распакуем данные и зададим имя папки с ними:

```
# Разархивация датасета в директорию 'content/cars'
!unzip -qo "middle_fmz.zip" -d /content/cars

# Папка с папками картинок, рассортированных по категориям
IMAGE_PATH = '/content/cars/'
```

Рисунок 2. Распаковка данных

Теперь можно увидеть, что находится в загруженной базе. Для этого воспользуемся функцией `listdir()` из модуля `os` и получим список папок по адресу `IMAGE_PATH`:

```
# Для работы с файлами
import os

os.listdir(IMAGE_PATH)
```

['Ferrari', 'Renault', 'Mercedes']

Рисунок 3. Функция `listdir()`

Функция `listdir()` возвращает список папок в неопределенном порядке, поэтому список классов желательно отсортировать, чтобы имена классов всегда шли в одном порядке. Метками классов будут индексы имен классов в списке классов. Количество классов определим как длину списка классов.

```
# Определение списка имен классов
CLASS_LIST = sorted(os.listdir(IMAGE_PATH))

# Определение количества классов
CLASS_COUNT = len(CLASS_LIST)

# Проверка результата
print(f'Количество классов: {CLASS_COUNT}, метки классов: {CLASS_LIST}')

Количество классов: 3, метки классов: ['Ferrari', 'Mercedes', 'Renault']
```

Рисунок 4. Определение списка имен и классов

Аналогично можно обратиться уже к каждой папке, чтобы получить имена файлов в них. Для этого соберем вместе путь до папки и имя папки:

```
i = 1

# Формирование пути к выборке одной марки авто
f'{IMAGE_PATH}{CLASS_LIST[i]}/'

'/content/cars/Mercedes/'
```

Рисунок 5. Обращение к папке

Теперь можно получить списки файлов для всех классов:

```
for cls in CLASS_LIST:
    print(cls, ': ', os.listdir(f'{IMAGE_PATH}{cls}/'))

Ferrari : ['car_Ferrari_1192.png', 'car_Ferrari_521.png', 'car_Ferrari_979.png', 'car_Ferrari_804.png',
Mercedes : ['car_626.png', 'car_473.png', 'car_735.png', 'car_293.png', 'car_8.png', 'car_247.png',
Renault : ['car_626.png', 'car_473.png', 'car_735.png', 'car_293.png', 'car_8.png', 'car_247.png',
```

Рисунок 6. Получение списков файлов

Далее в цикле переберем все классы. Сформируем путь к классу, выберем из него один случайный экземпляр (при помощи функции `random.choice()`) и отобразим его в ячейке (получим содержимое картинки при помощи функции `open()` из модуля `Image`).

```
from PIL import Image
import random
import matplotlib.pyplot as plt

# Создание заготовки для изображений всех классов
fig, axs = plt.subplots(1, CLASS_COUNT, figsize=(25, 5))

# Для всех номеров классов
for i in range(CLASS_COUNT):
    # Формирование пути к папке содержимого класса
    car_path = f'{IMAGE_PATH}{CLASS_LIST[i]}/'
    # Выбор случайного файла из 1-го файла
    img_path = car_path + random.choice(os.listdir(car_path))
    # Отображение изображения (подробнее будет объяснено далее)
    axs[i].set_title(CLASS_LIST[i])
    axs[i].imshow(Image.open(img_path))
    axs[i].axis('off')

# Отображение всего полотна
plt.show()
```




Рисунок 7. Перебор классов

Далее выполним создание списков файлов и их меток класса. По аналогии с тем, как выше осматривалось содержимое папок, можно получить имена файлов фотографий, собрать их в список `data_files`, а в список `data_labels` - собрать номера (метки) классов один за другим:

```
data_files = [] # Список путей к файлам картинок
data_labels = [] # Список меток классов, соответствующих файлам

for class_label in range(CLASS_COUNT): # Для всех классов по порядку номеров (их меток)
    class_name = CLASS_LIST[class_label] # Выборка имени класса из списка имен
    class_path = IMAGE_PATH + class_name # Формирование полного пути к папке с изображениями класса
    class_files = os.listdir(class_path) # Получение списка имен файлов с изображениями текущего класса
    print(f'Размер класса {class_name} составляет {len(class_files)} машин')

    # Добавление к общему списку всех файлов класса с добавлением родительского пути
    data_files += [f'{class_path}/{file_name}' for file_name in class_files]

    # Добавление к общему списку меток текущего класса - их ровно столько, сколько файлов в классе
    data_labels += [class_label] * len(class_files)

print('Общий размер базы для обучения:', len(data_labels))
```

Размер класса Ferrari составляет 1088 машин
Размер класса Mercedes составляет 1161 машин
Размер класса Renault составляет 1178 машин
Общий размер базы для обучения: 3427

Рисунок 8. Создание списков и их меток класса

Теперь в списках находятся пути к файлам и соответствующие им номера классов:

```
print('Пути к файлам: ', data_files[1085:1090])
print('Их метки классов:', data_labels[1085:1090])
```

Пути к файлам: ['/content/cars/Ferrari/car_Ferrari_938.png', '/content/cars/Ferrari/car_Ferrari_486.png',
Их метки классов: [0, 0, 0, 1, 1]

Рисунок 9. Путь к файлам

Далее выполним формирование набора данных из имеющейся базы. Вначале зададим размеры изображения.

- перебираем в цикле пути к файлам изображений;
- открываем каждое изображение;
- приводим изображение к заданному размеру;
- переводим изображение в числовой формат;
- присоединяем полученный массив к общему списку;
- переводим общий список изображений в numpy-массив;
- переводим общий список меток классов в numpy-массив.

```
[11] # Задание единых размеров изображений
IMG_WIDTH = 128 # Ширина изображения
IMG_HEIGHT = 64 # Высота изображения

[12] import numpy as np # Библиотека работы с массивами

data_images = [] # Пустой список для данных изображений

for file_name in data_files:
    # Открытие и смена размера изображения
    img = Image.open(file_name).resize((IMG_WIDTH, IMG_HEIGHT))
    img_np = np.array(img) # Перевод в numpy-массив
    data_images.append(img_np) # Добавление изображения в виде numpy-массива к общему списку

x_data = np.array(data_images) # Перевод общего списка изображений в numpy-массив
y_data = np.array(data_labels) # Перевод общего списка меток класса в numpy-массив

print(f'В массив собрано {len(data_images)} фотографий следующей формы: {img_np.shape}')
print(f'Общий массив данных изображений следующей формы: {x_data.shape}')
print(f'Общий массив меток классов следующей формы: {y_data.shape}')
```

В массив собрано 3427 фотографий следующей формы: (64, 128, 3)
Общий массив данных изображений следующей формы: (3427, 64, 128, 3)
Общий массив меток классов следующей формы: (3427,)

Рисунок 10. Задание размеров изображения

Изображения переведены в тензоры, посмотрим, в каком виде они сейчас хранятся в памяти, обратившись к первому из них по индексу:



Рисунок 11. Обращение к изображению

Далее выполним нормирование массива изображений.

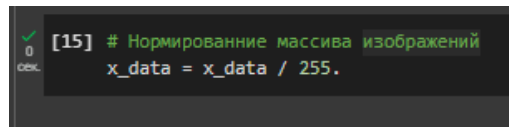


Рисунок 12. Нормирование массива изображений

Далее выполним создание сверточной сети. Для начала выполним подключение нужных слоев из модуля tensorflow.keras.layers.

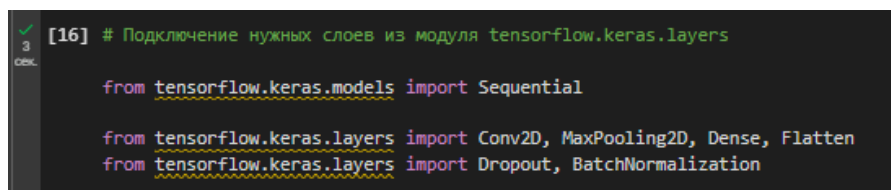


Рисунок 13. Подключение слоев из модуля tensorflow.keras.layers

Далее скомпонуем рассмотренные в теоретической части слои, в комментариях детально распишем, каким образом меняется форма данных при прохождении через слои.

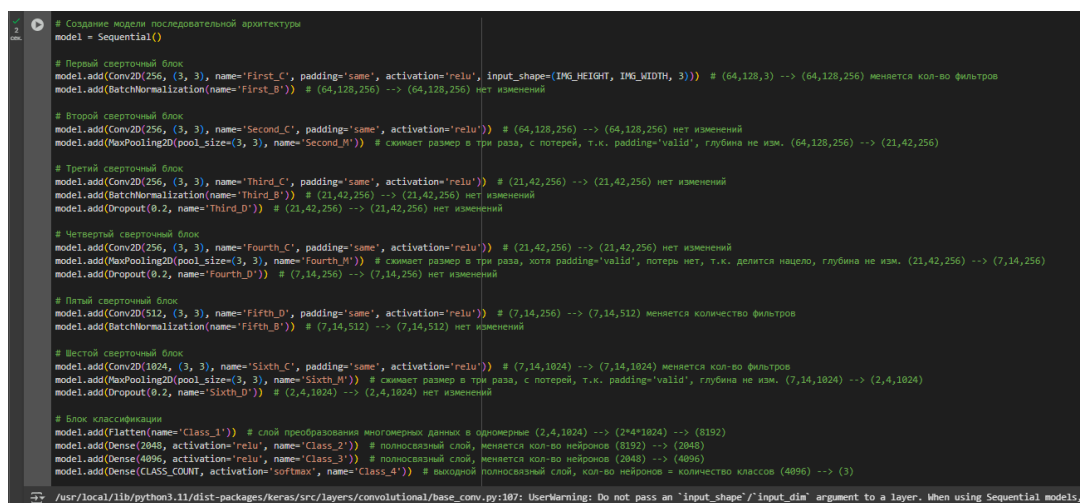


Рисунок 14. Создание модели

Далее запросим сводку по архитектуре сети и проверим, верны ли расчеты форм тензоров на выходе слоев:



Layer (type)	Output Shape	Param #
First_C (Conv2D)	(None, 64, 128, 256)	7,168
First_B (BatchNormalization)	(None, 64, 128, 256)	1,024
Second_C (Conv2D)	(None, 64, 128, 256)	590,080
Second_M (MaxPooling2D)	(None, 31, 63, 256)	0
Third_C (Conv2D)	(None, 31, 63, 256)	590,080
Third_B (BatchNormalization)	(None, 31, 63, 256)	1,024
Third_D (Dropout)	(None, 31, 63, 256)	0
Fourth_C (Conv2D)	(None, 31, 63, 256)	590,080
Fourth_M (MaxPooling2D)	(None, 7, 14, 256)	0
Fourth_D (Dropout)	(None, 7, 14, 256)	0
Fifth_D (Conv2D)	(None, 7, 14, 512)	1,100,160
Fifth_B (BatchNormalization)	(None, 7, 14, 512)	2,048
Sixth_C (Conv2D)	(None, 7, 14, 1024)	4,719,616
Sixth_M (MaxPooling2D)	(None, 7, 7, 1024)	0
Sixth_D (Dropout)	(None, 7, 7, 1024)	0
Class_1 (Flatten)	(None, 4913)	0
Class_2 (Dense)	(None, 4096)	16,778,264
Class_3 (Dense)	(None, 4096)	8,392,704
Class_4 (Dense)	(None, 1)	12,293

Total params: 17,865,539 (125.37 MB)
 Trainable params: 17,863,491 (125.36 MB)
 Non-trainable params: 2,048 (8.00 KB)

Рисунок 15. Архитектура сети

Затем нужно скомпилировать нейронную сеть. Для начала укажем метод подсчета ошибки сети – loss, по значению которой указанный оптимизатор (optimizer) пересчитает веса модели.

```
[19] # Подключение оптимизатора Adam
      from tensorflow.keras.optimizers import Adam

      # Компиляция модели
      model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(learning_rate=0.0001), metrics=['accuracy'])
```

Рисунок 16. Подключение оптимизатора

Далее выполним обучение модели сверточной нейронной сети подготовленных данных.

```
[ ] # Обучение модели сверточной нейронной сети подготовленных данных

store_learning = model.fit(x_data, # ----- x_train, примеры набора данных
                           y_data, # ----- y_train, метки примеров набора данных
                           validation_split=0.2, # --- 0.2 - доли данных для валидационной (проверочной) выборки, 1-0.2=0.8 останется в обучающей
                           shuffle=True, # ----- перемешивание данных для равномерного обучения, соответствие экземпляра и метки сохранятся
                           batch_size=25, # ----- размер пакета, который обрабатывает нейронка перед одним изменением весов
                           epochs=35, # ----- epochs - количество эпох обучения
                           verbose=1) # ----- 0 - не визуализировать ход обучения, 1 - визуализировать
```

```
Epoch 7/35 [=====] - 34s 318ms/step - loss: 0.5396 - accuracy: 0.7643 - val_loss: 1.1355 - val_accuracy: 0.4694
Epoch 8/35 [=====] - 34s 318ms/step - loss: 0.4774 - accuracy: 0.8015 - val_loss: 1.0392 - val_accuracy: 0.5539
Epoch 9/35 [=====] - 34s 309ms/step - loss: 0.4153 - accuracy: 0.8439 - val_loss: 1.2873 - val_accuracy: 0.4577
Epoch 10/35 [=====] - 34s 318ms/step - loss: 0.3824 - accuracy: 0.8515 - val_loss: 2.2179 - val_accuracy: 0.2755
Epoch 11/35 [=====] - 34s 318ms/step - loss: 0.3288 - accuracy: 0.8789 - val_loss: 1.6834 - val_accuracy: 0.3571
Epoch 12/35 [=====] - 34s 318ms/step - loss: 0.2647 - accuracy: 0.9004 - val_loss: 1.8601 - val_accuracy: 0.5918
Epoch 13/35 [=====] - 34s 318ms/step - loss: 0.2278 - accuracy: 0.9135 - val_loss: 1.4081 - val_accuracy: 0.5087
Epoch 14/35 [=====] - 34s 318ms/step - loss: 0.1988 - accuracy: 0.9278 - val_loss: 1.3612 - val_accuracy: 0.6312
Epoch 15/35 [=====] - 34s 318ms/step - loss: 0.1487 - accuracy: 0.9445 - val_loss: 2.0809 - val_accuracy: 0.4796
Epoch 16/35 [=====] - 34s 318ms/step - loss: 0.1441 - accuracy: 0.9467 - val_loss: 1.2826 - val_accuracy: 0.6399
Epoch 17/35 [=====] - 34s 309ms/step - loss: 0.1259 - accuracy: 0.9548 - val_loss: 3.5723 - val_accuracy: 0.3178
Epoch 18/35 [=====] - 34s 309ms/step - loss: 0.0962 - accuracy: 0.9658 - val_loss: 2.2241 - val_accuracy: 0.4985
Epoch 19/35 [=====] - 34s 309ms/step - loss: 0.0665 - accuracy: 0.9759 - val_loss: 2.6560 - val_accuracy: 0.4985
Epoch 20/35 [=====] - 34s 318ms/step - loss: 0.0816 - accuracy: 0.9683 - val_loss: 2.1716 - val_accuracy: 0.5525
Epoch 21/35 [=====] - 34s 318ms/step - loss: 0.0547 - accuracy: 0.9821 - val_loss: 2.6161 - val_accuracy: 0.4927
Epoch 22/35 [=====] - 34s 318ms/step - loss: 0.0554 - accuracy: 0.9818 - val_loss: 3.4627 - val_accuracy: 0.4380
Epoch 23/35 [=====] - 34s 318ms/step - loss: 0.0496 - accuracy: 0.9825 - val_loss: 3.3460 - val_accuracy: 0.4788
Epoch 24/35 [=====] - 34s 311ms/step - loss: 0.0526 - accuracy: 0.9818 - val_loss: 3.3891 - val_accuracy: 0.4388
Epoch 25/35 [=====] - 34s 309ms/step - loss: 0.0584 - accuracy: 0.9807 - val_loss: 2.7636 - val_accuracy: 0.5233
Epoch 26/35 [=====] - 34s 309ms/step - loss: 0.0697 - accuracy: 0.9763 - val_loss: 1.1544 - val_accuracy: 0.7187
Epoch 27/35 [=====] - 34s 318ms/step - loss: 0.0469 - accuracy: 0.9858 - val_loss: 4.2437 - val_accuracy: 0.3746
Epoch 28/35 [=====] - 34s 318ms/step - loss: 0.0479 - accuracy: 0.9861 - val_loss: 2.1042 - val_accuracy: 0.5802
Epoch 29/35 [=====] - 34s 309ms/step - loss: 0.0304 - accuracy: 0.9912 - val_loss: 2.1902 - val_accuracy: 0.6283
Epoch 30/35 [=====] - 34s 318ms/step - loss: 0.0163 - accuracy: 0.9967 - val_loss: 2.3958 - val_accuracy: 0.6086
Epoch 31/35 [=====] - 34s 309ms/step - loss: 0.0369 - accuracy: 0.9861 - val_loss: 1.1671 - val_accuracy: 0.7362
Epoch 32/35 [=====] - 34s 318ms/step - loss: 0.0556 - accuracy: 0.9818 - val_loss: 1.8612 - val_accuracy: 0.5991
Epoch 33/35 [=====] - 34s 318ms/step - loss: 0.0364 - accuracy: 0.9891 - val_loss: 3.3833 - val_accuracy: 0.4067
Epoch 34/35 [=====] - 34s 309ms/step - loss: 0.0209 - accuracy: 0.9934 - val_loss: 2.5528 - val_accuracy: 0.5598
Epoch 35/35 [=====] - 34s 309ms/step - loss: 0.0563 - accuracy: 0.9792 - val_loss: 3.4610 - val_accuracy: 0.4475
```

Рисунок 17. Обучение модели

Далее выполним построение графиков.

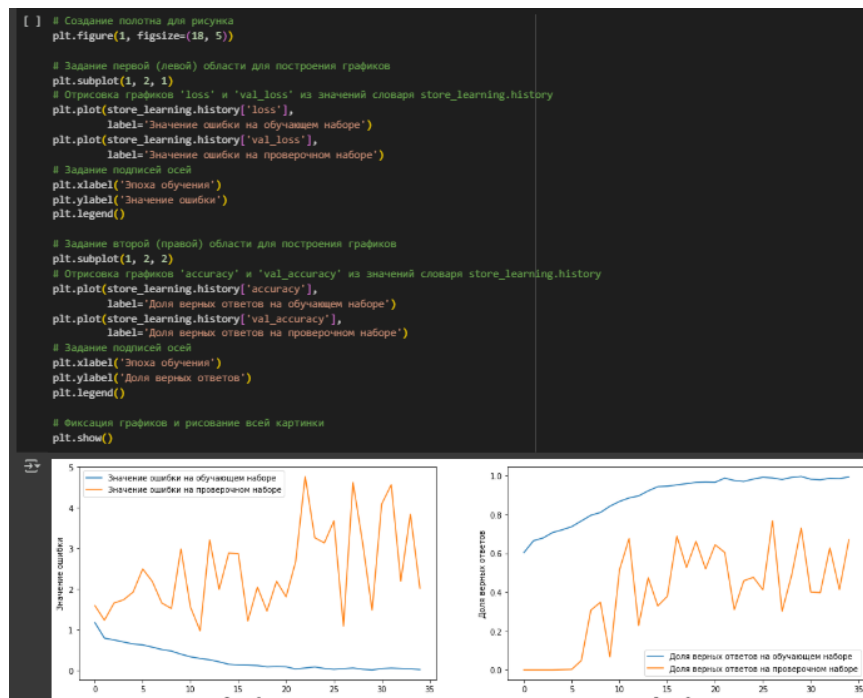


Рисунок 18. Построение графиков

Далее проведем аугментацию обучающих данных. Для начала Подключим все необходимые библиотеки:

```

✓ 0 [22] from PIL import Image, ImageEnhance      # Инструменты для работы с изображениями
сек. import matplotlib.pyplot as plt              # Отрисовка графиков
import numpy as np                               # Работа с массивами
import random                                    # Генерация случайных чисел
import math                                       # Математические функции

```

Рисунок 19. Подключение библиотек

Рассмотрим методы на примере работы с одним изображением. Создадим экземпляр класса Image библиотеки PIL, загрузив изображение из файла при помощи функции Image.open(). В скобках укажем путь к изображению из списка data_files по номеру i. Поместим содержимое изображения в переменную img, что откроет доступ к методам (функциям) объекта-картинки:

```

✓ 0 i = 100
сек. img = Image.open(data_files[i])             # Открытие i-го изображения из датасета
print('Размер исходного изображения:', img.size)

```

Размер исходного изображения: (192, 108)

Рисунок 20. Экземпляр класса Image

Создадим сервисную функцию для вывода картинки с помощью инструмента .imshow() из модуля plt:

```

✓ 0 def show_image(img):
сек. plt.figure(figsize=(8, 5))                  # Создание полотна для рисования
plt.imshow(img)                                # Отрисовка изображения
plt.axis('off')                                # Отключение ненужных осей
plt.show()                                     # Вывод результата

```

Рисунок 21. Сервисная функция

Также напишем сервисную функцию для визуального сравнения исходного изображения с измененным, и выведем прочитанную из файла картинку на экран:

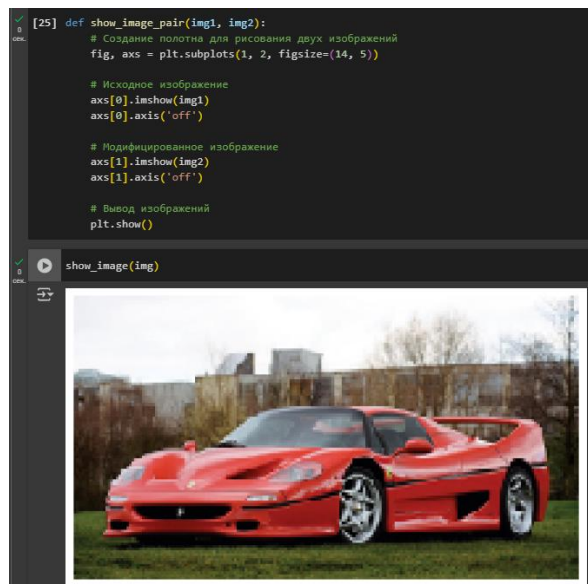


Рисунок 22. Вывод картинки

После создадим изображение с новыми размерами функцией-методом `.resize()`.



Рисунок 23. Изменение размеров

Далее с помощью функции-метода `.crop()` обрезаем изображение, выделяя прямоугольную область по заданным координатам.



Рисунок 24. Обрезка картинки

Далее напишем функцию случайной обрезки изображения `random_crop()`. Задавая разумные значения пределов обрезки (скажем, в пределах 0.25) и перезапуская ячейку, получим новые фрагменты из исходной фотографии.

```
[30] def random_crop(x,                # Подаваемое изображение
                    f_x,                # Предел обрезки справа и слева (в масштабе ширины)
                    f_y):              # Предел обрезки сверху и снизу (в масштабе высоты)

    # Получение левой и правой границ обрезки
    left = x.width * random.random() * f_x
    right = x.width * (1. - random.random()) * f_x - 1.

    # Получение верхней и нижней границ обрезки
    upper = x.height * random.random() * f_y
    lower = x.height * (1. - random.random()) * f_y - 1.

    return x.crop((left, upper, right, lower))

[31] img_crop = random_crop(img_, 0.2, 0.2)

# Вывод картинок
show_image_pair(img_, img_crop)

# Вывод размеров результата
img_crop.size
```




Рисунок 25. Случайная обрезка картинки

Далее с помощью функции-метода `.rotate()` будем вращать картинку внутри ее границ.

```
[32] angle = 10
img_rot = img_.rotate(angle, expand=True)

# Вывод картинки
show_image(img_rot)
```



Рисунок 26. Вращение картинки

Затем для создания новых обучающих изображений зададим небольшой угол и подрежем края. Угол должен быть случайным. Центр поворота сдвигать нет необходимости из-за обрезки краев.

```
[33] # Функция нахождения ширины и высоты прямоугольника наибольшей площади
# после поворота заданного прямоугольника на угол в градусах

def rotated_rect(w,          # Ширина изображения
                h,          # Высота изображения
                angle        # Угол поворота в градусах
                ):
    angle = math.radians(angle)
    width_is_longer = w >= h
    side_long, side_short = (w,h) if width_is_longer else (h,w)

    sin_a, cos_a = abs(math.sin(angle)), abs(math.cos(angle))

    if side_short <= 2.*sin_a*cos_a*side_long or abs(sin_a-cos_a) < 1e-10:
        x = 0.5 * side_short
        wr, hr = (x/sin_a, x/cos_a) if width_is_longer else (x/cos_a, x/sin_a)
    else:
        cos_2a = cos_a*cos_a - sin_a*sin_a
        wr, hr = (w*cos_a - h*sin_a)/cos_2a, (h*cos_a - w*sin_a)/cos_2a

    return wr, hr
```

Рисунок 27. Функция нахождения ширины и высоты прямоугольника

Далее воспользуемся функцией `rotated_rect()` для вычисления размеров прямоугольника обрезки, а затем обрежем исходную картинку методом `.crop()`, располагая рамку обрезки по центру исходной картинки:

```
[34] # Вычисление размеров прямоугольника обрезки максимальной площади
crop_w, crop_h = rotated_rect(img_.width, img_.height, angle)

# Обрезка повернутого изображения
w, h = img_rot.size
img_rot_crop = img_rot.crop(((w - crop_w)*0.5, (h - crop_h)*0.5,
                             (w + crop_w)*0.5, (h + crop_h)*0.5))

# Вывод картинки
show_image(img_rot_crop)
```




Рисунок 28. Функция `rotated_rect()`

Теперь напомним функцию поворота на случайный угол (в градусах) с учетом необходимой обрезки:

```
[35] def random_rot(x,          # Подаваемое изображение
                  ang         # Максимальный угол поворота
                  ):
    # Случайное значение угла в диапазоне [-ang, ang]
    a = random.uniform(-1., 1.) * ang

    # Вращение картинки с расширением рамки
    r = x.rotate(a, expand=True)

    # Вычисление размеров прямоугольника обрезки максимальной площади
    # для размеров исходной картинки и угла поворота в градусах
    crop_w, crop_h = rotated_rect(x.width, x.height, a)

    # Обрезка повернутого изображения и возврат результата
    w, h = r.size
    return r.crop(((w - crop_w)*0.5, (h - crop_h)*0.5,
                    (w + crop_w)*0.5, (h + crop_h)*0.5))

img_rot = random_rot(img_, 15)

# Вывод картинок
show_image_pair(img_, img_rot)

# Вывод размеров результата
img_rot.size
```

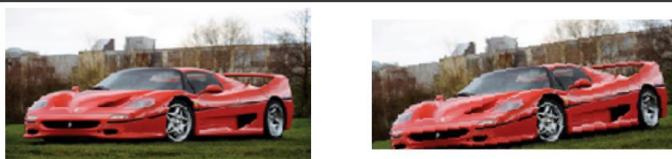


Рисунок 29. Функция поворота

Далее выполним отражение или поворот на угол, кратный 90 градусам: `.transpose()`. Затем обернем код по аналогии в функцию:

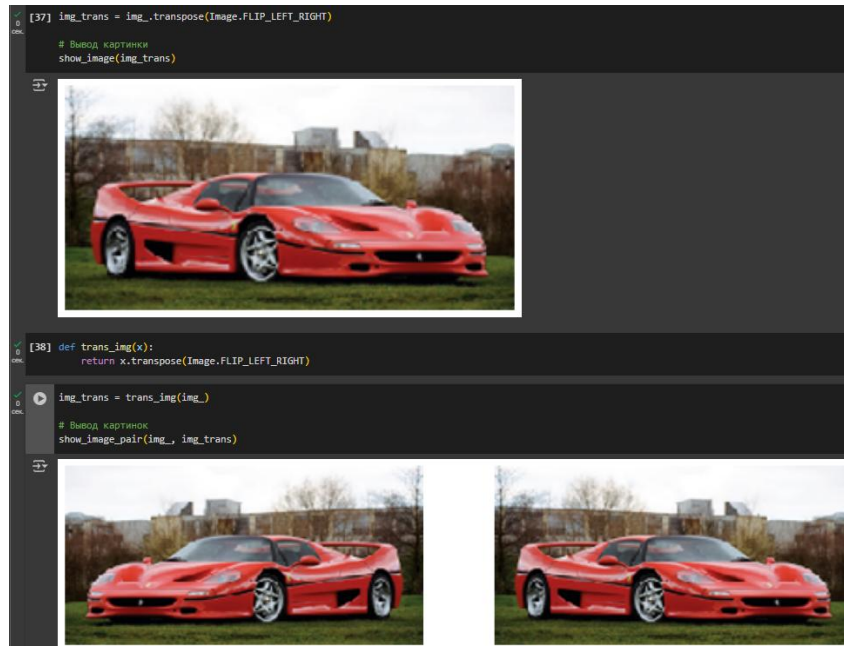


Рисунок 30. Отражение картинки

Далее выполним изменение контрастности: `Contrast`. И обернем код по аналогии в функцию.

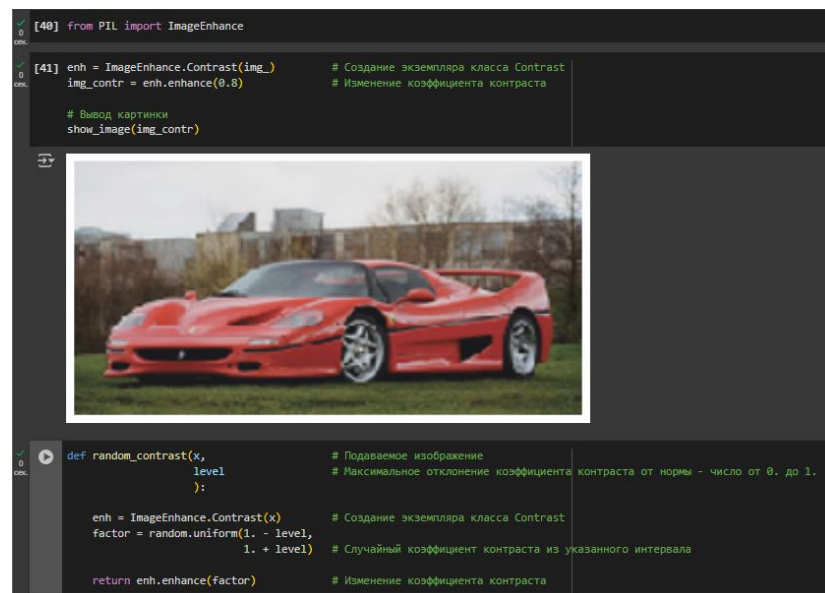


Рисунок 31. Изменение контрастности

После выполнения функции контрастности и выполним изменение яркости: `Brightness`.

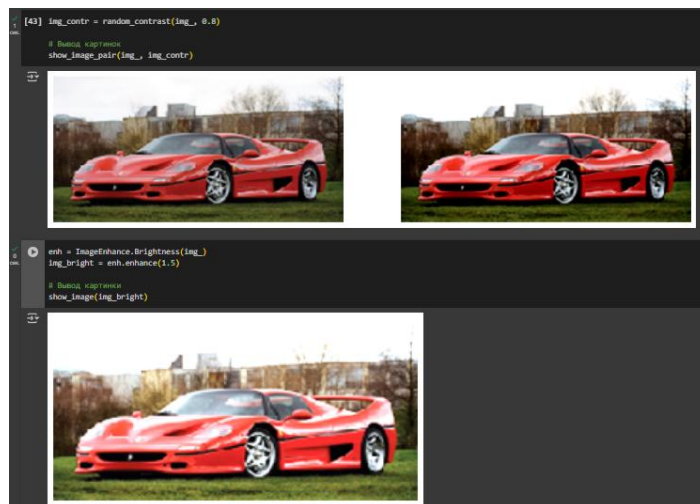


Рисунок 32. Изменение яркости

Далее обернем код по аналогии в функцию. И опробуем работу функции несколько раз:

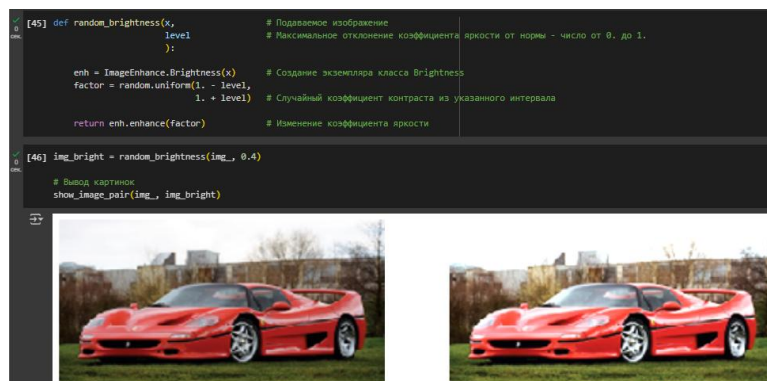


Рисунок 33. Функция изменения яркости

Далее выполним случайное применение изменений изображения. Примените выбранное изменение картинке по случайному индексу: `random.randrange(len(mod_oper))` из списка `mod_oper`.

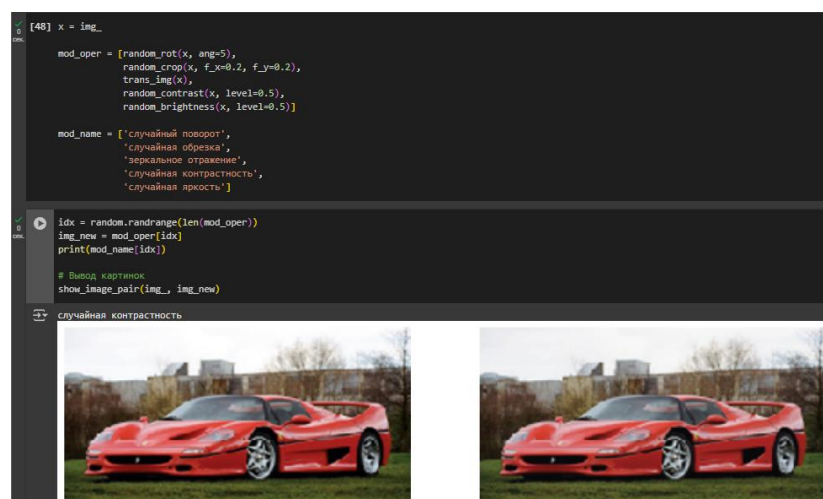


Рисунок 34. Случайное применение изменений изображения

Далее добавим цикл, который применит случайно отобранные изменения в случайном количестве.

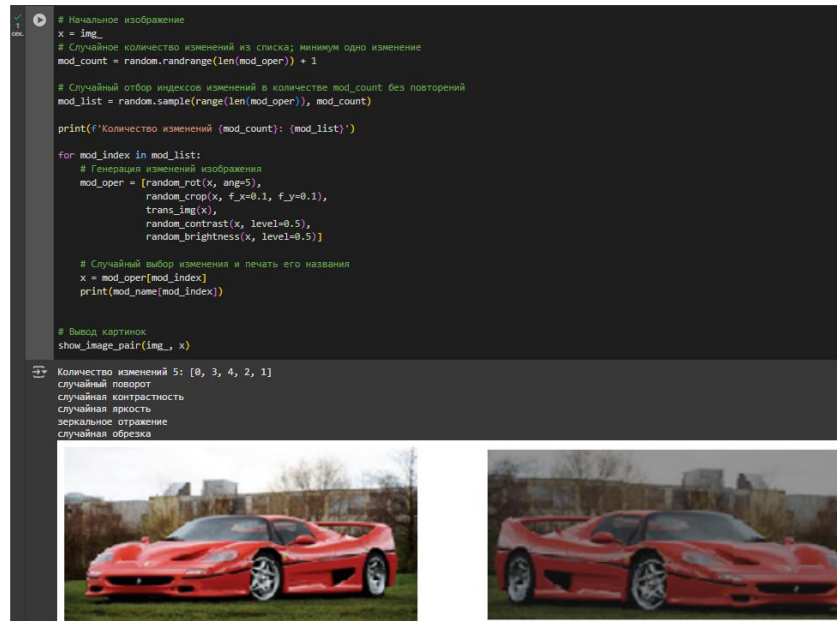


Рисунок 35. Выполнение цикла

Теперь осталось собрать отдельные функции изменений в одну общую функцию случайной аугментации изображений.

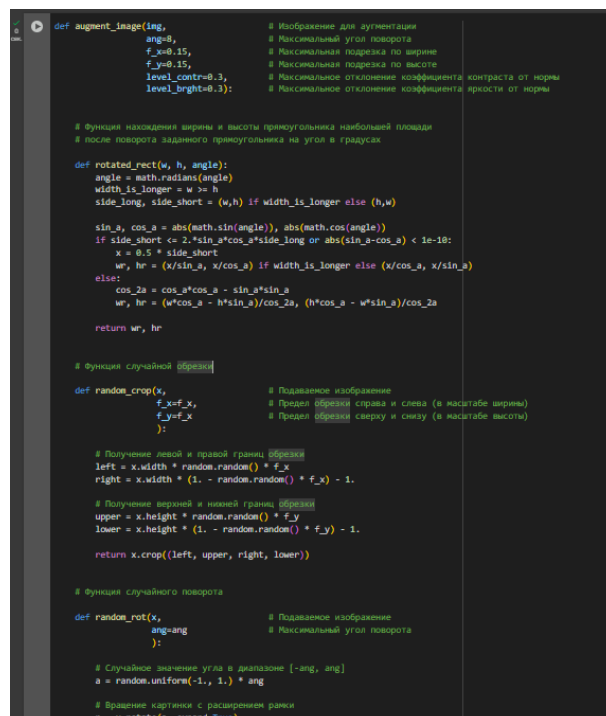


Рисунок 36. Функция случайной аугментации изображений

При запуске будет происходить применение случайного количества методов аугментации в случайном порядке. Посмотрим на несколько вариантов аугментации одной картинке, запуская вычисления в цикле:

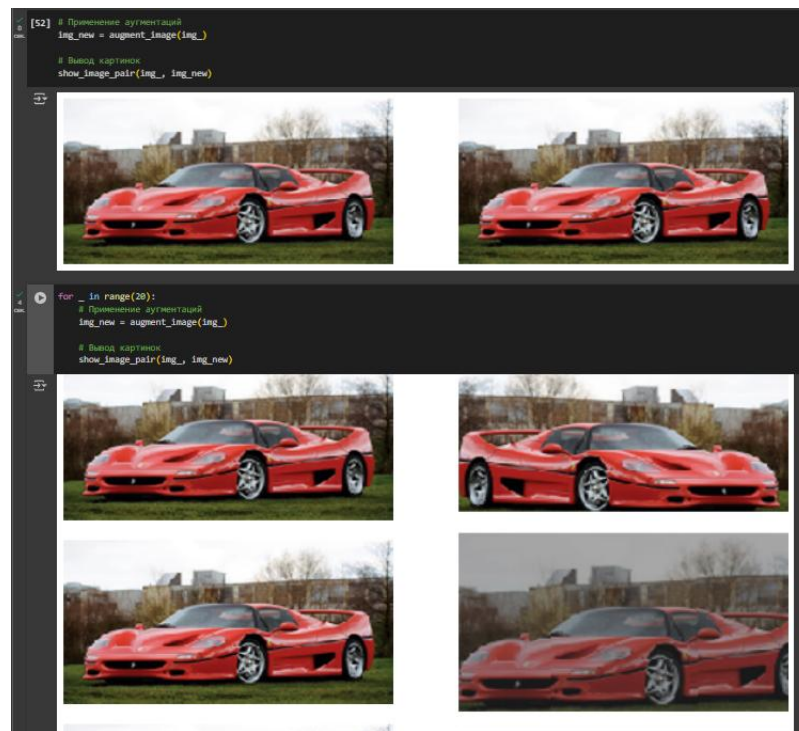


Рисунок 37. Запуск цикла

Задание 2. Сверточные нейронные сети (Практика 2).

Для начала выполним подключение библиотек.

```

# Работа с массивами
import numpy as np

# Генератор аугментированных изображений
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Основа для создания последовательной модели
from tensorflow.keras.models import Sequential

# Основные слои
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization

# Оптимизатор
from tensorflow.keras.optimizers import Adam

# Матрица ошибок классификатора
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Подключение модуля для загрузки данных из облака
import gdown

# Инструменты для работы с файлами
import os

# Отрисовка графиков
import matplotlib.pyplot as plt

# Рисование графиков в ячейках Colab
%matplotlib inline

```

Рисунок 38. Загрузка библиотек

Задание гиперпараметров модели. Для начала необходимо правильно сгруппировать и организовать значения параметров, по которым будут производиться аугментация и обучение модели. Сперва зададим гиперпараметры для сети, которые включают директории для данных, значения для разделения выборок и входного размера изображений. А также выполним загрузку датасета и подготовку данных.

```
2 # Задание гиперпараметров

TRAIN_PATH = '/content/cars' # Папка для обучающего набора данных
TEST_PATH = '/content/cars_test' # Папка для тестового набора данных

TEST_SPLIT = 0.1 # Доля тестовых данных в общем наборе
VAL_SPLIT = 0.2 # Доля проверочной выборки в обучающем наборе

IMG_WIDTH = 128 # Ширина изображения для нейросети
IMG_HEIGHT = 64 # Высота изображения для нейросети
IMG_CHANNELS = 3 # Количество каналов (для RGB равно 3, для Grey равно 1)

# Параметры аугментации
ROTATION_RANGE = 8 # Пределы поворота
WIDTH_SHIFT_RANGE = 0.15 # Пределы сдвига по горизонтали
HEIGHT_SHIFT_RANGE = 0.15 # Пределы сдвига по вертикали
ZOOM_RANGE = 0.15 # Пределы увеличения/уменьшения
BRIGHTNESS_RANGE = (0.7, 1.3) # Пределы изменения яркости
HORIZONTAL_FLIP = True # Горизонтальное отражение разрешено

EPOCHS = 60 # Число эпох обучения
BATCH_SIZE = 24 # Размер батча для обучения модели
OPTIMIZER = Adam(0.0001) # Оптимизатор

3 # Загрузка zip-архива с датасетом из облака на диск виртуальной машины colab
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/15/middle_fmr.zip', None, quiet=True)

'middle_fmr.zip'
```

Рисунок 39. Загрузка датасета

Далее определим список классов и их число.

```
# Определение списка имен классов
CLASS_LIST = sorted(os.listdir(TRAIN_PATH))

# Определение количества классов
CLASS_COUNT = len(CLASS_LIST)

# Проверка результата
print(f'Количество классов: {CLASS_COUNT}, метки классов: {CLASS_LIST}')

Количество классов: 3, метки классов: ['Ferrari', 'Mercedes', 'Renault']

[6] int(1088 * 0.1)

108

[7] photo_1 = '/content/cars/Ferrari/car_Ferrari_0.png'
photo_1 = '/content/cars_test/Ferrari/car_Ferrari_0.png'
```

Рисунок 40. Определение списка классов

Для разделения данных требуется не только создать папку и указать путь к ней; также нужно определить количество изображений в каждом из трёх классов, выделить некоторую их долю (заданную в гиперпараметрах) в каждом классе, и уже потом переместить файлы по указанному пути в папку.


```
# Перенос файлов для теста в отдельное дерево папок, расчет размеров наборов данных

try:
    os.mkdir(TEST_PATH)
except:
    pass

train_count = 0
test_count = 0

for class_name in CLASS_LIST:
    class_path = f'{TRAIN_PATH}/{class_name}'
    test_path = f'{TEST_PATH}/{class_name}'
    class_files = os.listdir(class_path)
    class_file_count = len(class_files)

    try:
        os.mkdir(test_path)
    except:
        pass

    test_file_count = int(class_file_count * TEST_SPLIT)
    test_files = class_files[-test_file_count:]
    for f in test_files:
        os.rename(f'{class_path}/{f}', f'{test_path}/{f}')
    train_count += class_file_count
    test_count += test_file_count

    print(f'Размер класса {class_name}: {class_file_count} машин, для теста выделено файлов: {test_file_count}')

print(f'Общий размер базы: {train_count}, выделено для обучения: {train_count - test_count}, для теста: {test_count}')
```

Размер класса Ferrari: 1088 машин, для теста выделено файлов: 108
Размер класса Mercedes: 1161 машин, для теста выделено файлов: 116
Размер класса Renault: 1178 машин, для теста выделено файлов: 117
Общий размер базы: 3427, выделено для обучения: 3086, для теста: 341

Рисунок 41. Перенос файлов для теста в отдельное дерево папок
Далее рассмотрим генераторы изображений и выборки.

```
[9] # Генераторы изображений

# Изображения для обучающего набора нормализуются и аугментируются согласно заданным гиперпараметрам
# Далее набор будет разделен на обучающую и проверочную выборку в соотношении VAL_SPLIT
train_datagen = ImageDataGenerator(
    rescale=1. / 255.,
    rotation_range=ROTATION_RANGE,
    width_shift_range=WIDTH_SHIFT_RANGE,
    height_shift_range=HEIGHT_SHIFT_RANGE,
    zoom_range=ZOOM_RANGE,
    brightness_range=BRIGHTNESS_RANGE,
    horizontal_flip=HORIZONTAL_FLIP,
    validation_split=VAL_SPLIT
)

# Изображения для тестового набора только нормализуются
test_datagen = ImageDataGenerator(
    rescale=1. / 255.
)
```

Рисунок 42. Генераторы изображения и выборки

Существует удобный метод генератора - `.flow_from_directory()`, который помогает извлечь из папок изображения для генерации, посчитать классы и автоматически вычислить метки классов для изображений.

```
# Обучающая выборка генерируется из папки обучающего набора
train_generator = train_datagen.flow_from_directory(
    # Путь к обучающим изображениям
    TRAIN_PATH,
    # Параметры требуемого размера изображения
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    # Размер батча
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True,
    # Указание сгенерировать обучающую выборку
    subset='training'
)

# Проверочная выборка также генерируется из папки обучающего набора
validation_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True,
    # Указание сгенерировать проверочную выборку
    subset='validation'
)

# Тестовая выборка генерируется из папки тестового набора
test_generator = test_datagen.flow_from_directory(
    TEST_PATH,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=test_count,
    class_mode='categorical',
    shuffle=True,
)

Found 2469 images belonging to 3 classes.
Found 617 images belonging to 3 classes.
Found 341 images belonging to 3 classes.
```

Рисунок 43. Метод генератора - .flow_from_directory()

Элемент train_generator содержит 24 элемента. Выведем список из 24 пар.

```
[11] len(train_generator[0]) == 24
False

train_generator[0]
# (список пар изображений-меток)

...,
[[0.3372549, 0.32156864, 0.29883923],
 [0.32941177, 0.31764707, 0.2901961 ],
 [0.32941177, 0.3137255, 0.28627452],
 ...,
 [0.3803922, 0.43137258, 0.45882356],
 [0.3803922, 0.43137258, 0.45882356],
 [0.3803922, 0.43137258, 0.45882356]],
[[0.34117648, 0.32156864, 0.3019608 ],
 [0.34117648, 0.32156864, 0.3019608 ],
 [0.34117648, 0.32156864, 0.3019608 ],
 ...,
 [0.37254903, 0.42352945, 0.43921572],
 [0.37647063, 0.427451, 0.4431373 ],
 [0.37647063, 0.427451, 0.4431373 ]],
[[0.34509805, 0.32156864, 0.30980393],
 [0.34509805, 0.32156864, 0.30980393],
 [0.34509805, 0.32156864, 0.30980393],
 ...]]
```

Рисунок 44. Элемент train_generator

Далее выведем первую пару (изображение, метка) в первом батче.

```
[13] train_generator[0][0]
# (изображение, метка)

[[0.7725491, 0.7686275, 0.78823537],
 [0.7725491, 0.7686275, 0.78823537],
 [0.77647066, 0.7725491, 0.79215693],
 ...,
 [0.14901961, 0.04705883, 0.04313726],
 [0.03921569, 0.01568628, 0.01568628],
 [0.04313726, 0.02745098, 0.02745098]]]
```

Рисунок 45. Первая пара в первом батче

Далее выведем изображение из первой пары первого батча.

```
[14] train_generator[0][0][0]
# изображение

array([[0.6313726 , 0.6784314 , 0.7058824 ],
       [0.60784316, 0.654902  , 0.6862745 ],
       [0.61960787, 0.65882355, 0.6901961 ],
       ...,
       [0.5254902 , 0.56078434, 0.6039216 ],
       [0.5176471 , 0.5568628 , 0.6039216 ],
       [0.5176471 , 0.5568628 , 0.6          ]],

       [[0.68235296, 0.7137255 , 0.7372549 ],
       [0.654902  , 0.6862745 , 0.7137255 ],
       [0.6509804 , 0.68235296, 0.70980394],
```

Рисунок 46. Изображение из первой пары первого батча

После выведем первое изображение в первом батче.

```
train_generator[0][0][1]
# метка

array([[0.6862745 , 0.69803923, 0.70980394],
       [0.6901961 , 0.7019608 , 0.70980394],
       [0.69411767, 0.7058824 , 0.7137255 ],
       ...,
       [0.7411765 , 0.74509805, 0.7411765 ],
       [0.7411765 , 0.74509805, 0.7411765 ],
       [0.7411765 , 0.74509805, 0.7411765 ]],

       [[0.6862745 , 0.69803923, 0.70980394],
       [0.6901961 , 0.7019608 , 0.70980394],
       [0.69411767, 0.7058824 , 0.7137255 ],
```

Рисунок 47. Первое изображение в первом батче

Далее выполним создание экземпляра класса с массивом [1,2,3,4,5].

После выполним метод `run`, который всегда возвращает строку '123'.

```
[16] class Some:
      def __init__(self, array):
          self.array = array
      def run(self):
          return '123'

      def __len__(self):
          return len(self.array)

[17] sss = Some([1,2,3,4,5])

[18] sss.run()
'123'

[19] len(sss.array)
5

len(sss)
5
```

Рисунок 48. Использование встроенных функций

После выполним проверку формы данных и проверку назначения меток классов.

```
[21] # Проверка формы данных
print(f'Формы данных тренировочной выборки: {train_generator[0][0].shape}, {train_generator[0][1].shape}, батчей: {len(train_generator)}')
print(f'Формы данных проверочной выборки: {validation_generator[0][0].shape}, {validation_generator[0][1].shape}, батчей: {len(validation_generator)}')
print(f'Формы данных тестовой выборки: {test_generator[0][0].shape}, {test_generator[0][1].shape}, батчей: {len(test_generator)}')

print()

# Проверка назначения меток классов
print(f'Метки классов тренировочной выборки: {train_generator.class_indices}')
print(f'Метки классов проверочной выборки: {validation_generator.class_indices}')
print(f'Метки классов тестовой выборки: {test_generator.class_indices}')

Формы данных тренировочной выборки: (24, 64, 128, 3), (24, 3), батчей: 103
Формы данных проверочной выборки: (24, 64, 128, 3), (24, 3), батчей: 26
Формы данных тестовой выборки: (341, 64, 128, 3), (341, 3), батчей: 1

Метки классов тренировочной выборки: {'Ferrari': 0, 'Mercedes': 1, 'Renault': 2}
Метки классов проверочной выборки: {'Ferrari': 0, 'Mercedes': 1, 'Renault': 2}
Метки классов тестовой выборки: {'Ferrari': 0, 'Mercedes': 1, 'Renault': 2}

[22] len(train_generator)
103

[23] len(train_generator[0])
2

x_train, y_train = train_generator[0]
```

Рисунок 49. Проверка формы данных

Далее выполним проверку работы генераторов выборок.

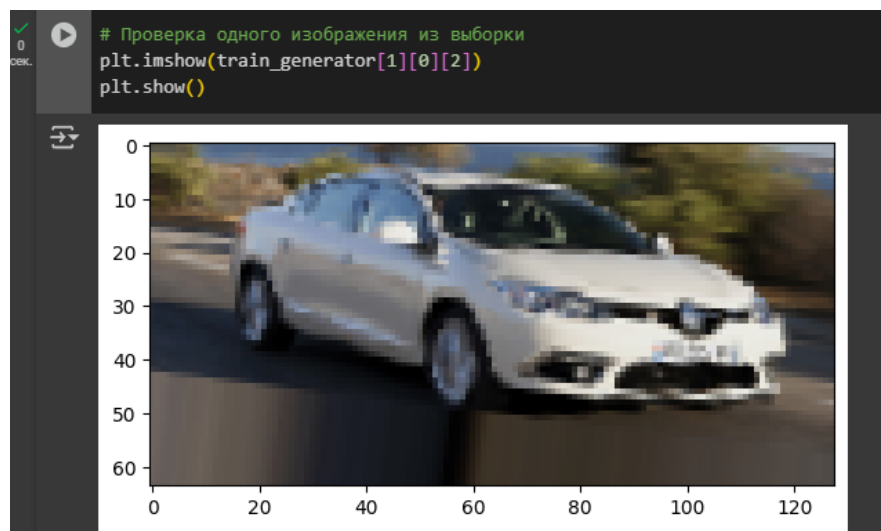


Рисунок 50. Проверка работы генераторов выборок

Сама же картинка представлена в виде трехмерного массива нормализованных пикселей (от 0 до 1):

```
[29] train_generator[1][0][2]

array([[0.6039216, 0.6431373, 0.68235296],
       [0.6039216, 0.6431373, 0.68235296],
       [0.6039216, 0.6431373, 0.68235296],
       ...,
       [0.57254905, 0.6, 0.63529414],
       [0.57254905, 0.6, 0.63529414],
       [0.57254905, 0.6, 0.63529414],
       ...,
       [0.6039216, 0.6431373, 0.68235296],
       [0.6039216, 0.6431373, 0.68235296],
       [0.6039216, 0.6431373, 0.68235296],
       ...,
       [0.5686275, 0.6, 0.6313726 ],
       [0.5686275, 0.6, 0.6313726 ],
       [0.5686275, 0.6, 0.6313726 ]],
```

Рисунок 51. Трехмерный массив

Теперь создадим функцию для удобного просмотра сразу множества картинок из заданного батча. Для отрисовки нескольких изображений используем функцию `.subplots()` библиотеки `matplotlib.pyplot`.

```
[31] x_train, y_train = train_generator[0]
[32] batch = train_generator[0]
[34] # Функция рисования образцов изображений из заданной выборки
def show_batch(batch,
               img_range=range(20),
               figsize=(25, 8),
               columns=5
               ):
    for i in img_range:
        ix = i % columns
        if ix == 0:
            fig, ax = plt.subplots(1, columns, figsize=figsize)
            class_label = np.argmax(batch[1][i])
            ax[ix].set_title(CLASS_LIST[class_label])
            ax[ix].imshow(batch[0][i])
            ax[ix].axis('off')
        plt.tight_layout()
    plt.show()
```

Рисунок 52. Использование функции `.subplots()`

Далее посмотрим на примеры из обучающей выборки:

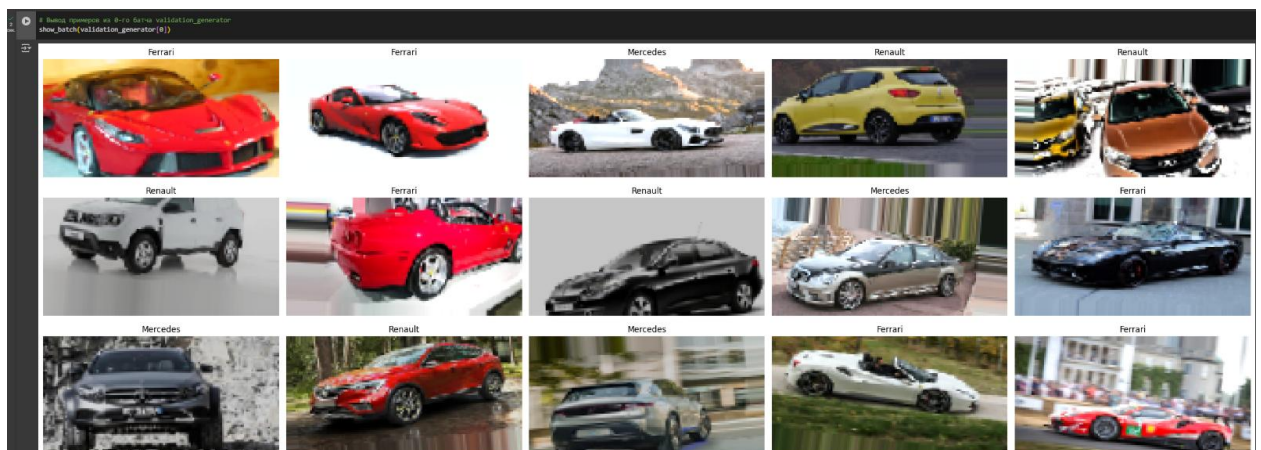


Рисунок 53. Примеры из обучающей выборки

Далее посмотрим на примеры из проверочной выборки:

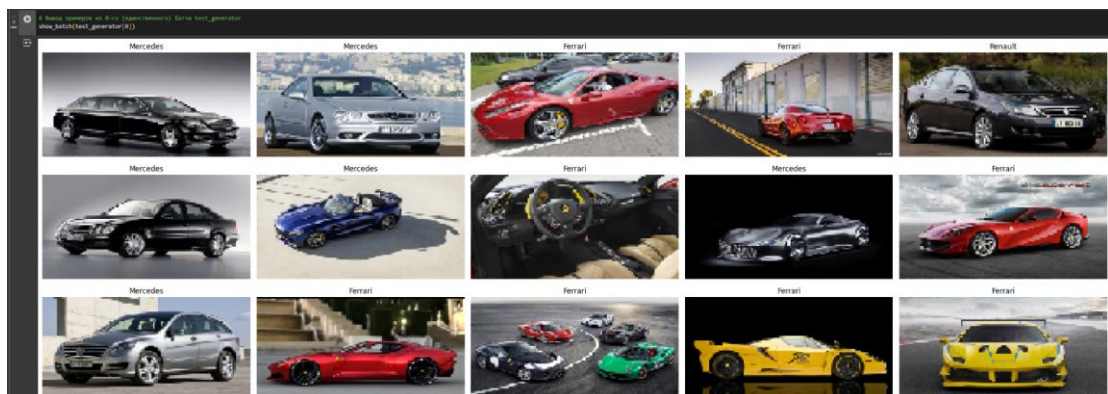


Рисунок 53. Примеры из проверочной выборки

Далее выполним создание и обучение модели нейронной сети. Для этого используем сервисные функции.

```
# Функция компиляции и обучения модели нейронной сети
# По окончании выводит графики обучения

def compile_train_model(model,          # модель нейронной сети
                        train_data,     # обучающие данные
                        val_data,       # проверочные данные
                        optimizer=OPTIMIZER, # оптимизатор
                        epochs=EPOCHS,   # количество эпох обучения
                        batch_size=BATCH_SIZE, # размер батча
                        figsize=(20, 5)): # размер полотна для графиков

    # Компиляция модели
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Вывод сводки
    model.summary()

    # Обучение модели с заданными параметрами
    history = model.fit(train_data,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=val_data)

    # Вывод графиков точности и ошибки
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=figsize)
    fig.suptitle('График процесса обучения модели')
    ax1.plot(history.history['accuracy'],
             label='Доля верных ответов на обучающем наборе')
    ax1.plot(history.history['val_accuracy'],
             label='Доля верных ответов на проверочном наборе')
    ax1.xaxis.get_major_locator().set_params(integer=True)
    ax1.set_xlabel('Эпоха обучения')
    ax1.set_ylabel('Доля верных ответов')
    ax1.legend()

    ax2.plot(history.history['loss'],
             label='Ошибка на обучающем наборе')
    ax2.plot(history.history['val_loss'],
             label='Ошибка на проверочном наборе')
    ax2.xaxis.get_major_locator().set_params(integer=True)
    ax2.set_xlabel('Эпоха обучения')
    ax2.set_ylabel('Ошибка')
    ax2.legend()
    plt.show()
```

Рисунок 54. Сервисные функции

Далее выполним функцию вывода результатов оценки модели на заданных данных.

```
# Функция вывода результатов оценки модели на заданных данных

def eval_model(model,
               x,          # данные для предсказания модели (вход)
               y_true,     # верные метки классов в формате ONE (выход)
               class_labels=[], # список меток классов
               cm_round=3,  # число знаков после запятой для матрицы ошибок
               title='',    # название модели
               figsize=(10, 10) # размер полотна для матрицы ошибок
               ):
    # Вычисление предсказания сети
    y_pred = model.predict(x)
    # Построение матрицы ошибок
    cm = confusion_matrix(np.argmax(y_true, axis=1),
                        np.argmax(y_pred, axis=1),
                        normalize='true')
    # Округление значений матрицы ошибок
    cm = np.around(cm, cm_round)

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

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

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

    # Средняя точность распознавания определяется как среднее диагональных элементов матрицы ошибок
    print('\nСредняя точность распознавания: {:3.0f}%'.format(100. * cm.diagonal().mean()))
```

Рисунок 55. Функция вывода результатов оценки модели на заданных данных

Далее напишем совместную функцию обучения и оценки модели нейронной сети.

```
# Совместная функция обучения и оценки модели нейронной сети

def compile_train_eval_model(model,          # модель нейронной сети
                             train_data,    # обучающие данные
                             val_data,      # проверочные данные
                             test_data,     # тестовые данные
                             class_labels=CLASS_LIST, # список меток классов
                             title='',      # название модели
                             optimizer=OPTIMIZER, # оптимизатор
                             epochs=EPOCHS,  # количество эпох обучения
                             batch_size=BATCH_SIZE, # размер батча
                             graph_size=(20, 5), # размер полотна для графиков обучения
                             cm_size=(10, 10) # размер полотна для матрицы ошибок
                             ):

    # Компиляция и обучение модели на заданных параметрах
    # В качестве проверочных используются тестовые данные
    compile_train_model(model,
                        train_data,
                        val_data,
                        optimizer=optimizer,
                        epochs=epochs,
                        batch_size=batch_size,
                        figsize=graph_size)

    # Вывод результатов оценки работы модели на тестовых данных
    eval_model(model, test_data[0][0], test_data[0][1],
               class_labels=class_labels,
               title=title,
               figsize=cm_size)
```

Рисунок 56. Совместная функция обучения и оценки модели нейронной сети

Теперь создадим модель, обучим ее на генерируемых данных и оценим работу на тестовых:

```
# Создание последовательной модели
model_conv = Sequential()

# Первый сверточный слой
model_conv.add(Conv2D(256, (3, 3), padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)))
model_conv.add(BatchNormalization())

# Второй сверточный слой
model_conv.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model_conv.add(MaxPooling2D(pool_size=(3, 3)))

# Третий сверточный слой
model_conv.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model_conv.add(BatchNormalization())
model_conv.add(Dropout(0.2))

# Четвертый сверточный слой
model_conv.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model_conv.add(MaxPooling2D(pool_size=(3, 3)))
model_conv.add(Dropout(0.2))

# Пятый сверточный слой
model_conv.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
model_conv.add(BatchNormalization())

# Шестой сверточный слой
model_conv.add(Conv2D(1024, (3, 3), padding='same', activation='relu'))
model_conv.add(MaxPooling2D(pool_size=(3, 3)))
model_conv.add(Dropout(0.2))

# Слой преобразования многомерных данных в одномерные
model_conv.add(Flatten())

# Промежуточный полносвязный слой
model_conv.add(Dense(2048, activation='relu'))

# Промежуточный полносвязный слой
model_conv.add(Dense(4096, activation='relu'))

# Выходной полносвязный слой с количеством нейронов по количеству классов
model_conv.add(Dense(CLASS_COUNT, activation='softmax'))

# Обучение модели и вывод оценки ее работы на тестовых данных
compile_train_eval_model(model_conv,
                        train_generator,
                        validation_generator,
                        test_generator,
                        class_labels=CLASS_LIST,
                        title='Сверточный классификатор')
```

Рисунок 57. Создание модели

Результат создания модели и обучения модели:

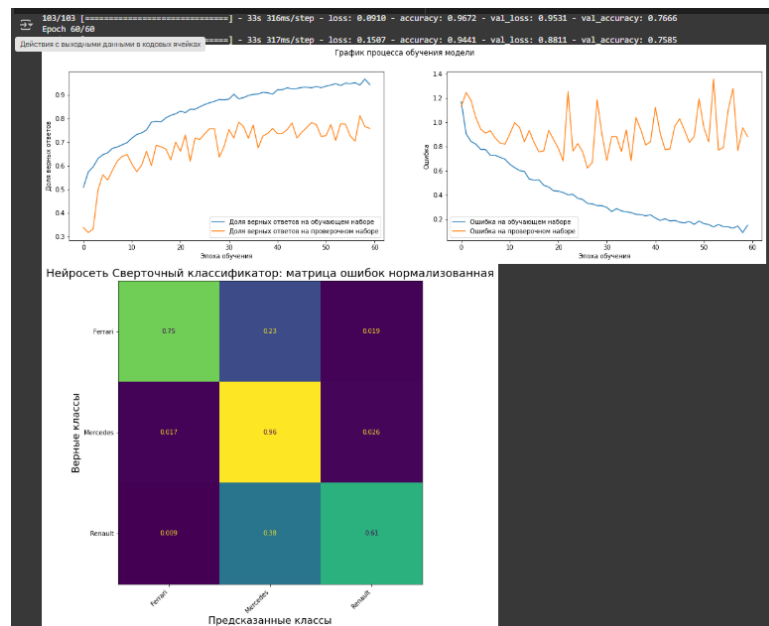


Рисунок 58. Графики процесса обучения модели

Выполнение индивидуальных заданий:

Задание 1.

Условие: необходимо создать нейронную сеть, распознающую рукописные цифры. Используя подготовленную базу и шаблон ноутбука, необходимо нормировать данные, а также создать и обучить сверточную сеть.

– Параметры модели: сеть должна содержать минимум 2 сверточных слоя; полносвязные слои; слои подвыборки, нормализации, регуляризации по 1 шт.

– Гиперпараметры обучения: функция ошибки - категориальная кроссэнтропия, оптимизатор - Adam с шагом обучения одна тысячная, размер батча - 128, количество эпох 15, детали обучения - отображать.

В конце необходимо вывести график обучения: доли верных ответов на обучающей и проверочной выборках.

Для начала выполним загрузку датасета MNIST:

```
[1] # загрузка датасета MNIST

from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 — 0s 0us/step
```

Рисунок 59. Загрузка датасета MNIST

Далее выполним загрузку необходимых библиотек:


```
[2] # Подключение утилит для to_categorical
from tensorflow.keras import utils

# Подключение библиотеки для работы с массивами
import numpy as np

# Подключение библиотек для отрисовки изображений
import matplotlib.pyplot as plt

# Подключение библиотеки для генерации случайных чисел
import random

# Подключение класса для работы с изображением
from PIL import Image

# Вывод изображения в ноутбуке, а не в консоли или файле
%matplotlib inline
```

Рисунок 60. Загрузка библиотек

Далее выполним вывод изображений каждого класса для ознакомления с датасетом:

```
[3] # вывод изображений каждого класса для ознакомления с датасетом

fig, axes = plt.subplots(1, 10, figsize=(25,3)) # создаем полотно для 10 графиков с размером 25 на 3
for i in range(10):
    label_indexes = np.where(y_train == i)[0] # получаем список из индексов положений класса i в y_train
    index = random.choice(label_indexes) # выбирает случайный индекс из списка созданного выше
    img = x_train[index] # выбираем из x_train нужное положение
    axes[i].imshow(Image.fromarray(img), cmap='gray') # выводим изображение

plt.show()
```



Рисунок 61. Вывод изображений

Далее посмотрим форматы выборок перед обучением:

```
[4] # добавляем размерность массиву mnist, чтобы сеть поняла что это чб
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], x_test.shape[2], 1)

# преобразуем выборки с ответами в ONE
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)

# посмотрим форматы выборок перед обучением
print('x_train:', x_train.shape)
print('x_test:', x_test.shape)
print()
print('y_train:', y_train.shape)
print('y_test:', y_test.shape)
```

x_train: (60000, 28, 28, 1)
x_test: (10000, 28, 28, 1)

y_train: (60000, 10)
y_test: (10000, 10)

Рисунок 62. Форматы выборок

Далее выполним создание модели и выведем ее структуру:

```
# Создание модели CNN
model = Sequential()

# Первый сверточный слой
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(BatchNormalization()) # Слой нормализации

# Второй сверточный слой
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) # Слой подвыборки (пулинг)
model.add(Dropout(0.25)) # Слой регуляризации

# Преобразование в одномерный вектор для полносвязных слоев
model.add(Flatten())

# Полносвязный слой
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) # Еще один слой регуляризации

# Выходной слой
model.add(Dense(10, activation='softmax'))

# Компиляция модели
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.001),
              metrics=['accuracy'])

# Вывод структуры модели
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization_1 (BatchNormalization)	(None, 26, 26, 32)	128
conv2d_3 (Conv2D)	(None, 24, 24, 64)	16,640
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_2 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_2 (Dense)	(None, 128)	1,179,776
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

Total params: 1,280,810 (4.58 MB)
Trainable params: 1,190,944 (4.58 MB)
Non-trainable params: 64 (256.00 B)

Рисунок 63. Создание модели

Затем выполним обучение модели:

```
[6] # Обучение модели
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=15,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

Epoch 1/15
469/469 — 154s 321ms/step - accuracy: 0.7643 - loss: 0.7590 - val_accuracy: 0.9820 - val_loss: 0.0607
Epoch 2/15
469/469 — 199s 316ms/step - accuracy: 0.9419 - loss: 0.1868 - val_accuracy: 0.9870 - val_loss: 0.0437
Epoch 3/15
469/469 — 202s 315ms/step - accuracy: 0.9541 - loss: 0.1400 - val_accuracy: 0.9871 - val_loss: 0.0399
Epoch 4/15
469/469 — 204s 320ms/step - accuracy: 0.9619 - loss: 0.1174 - val_accuracy: 0.9880 - val_loss: 0.0380
Epoch 5/15
469/469 — 148s 316ms/step - accuracy: 0.9652 - loss: 0.1028 - val_accuracy: 0.9885 - val_loss: 0.0379
Epoch 6/15
469/469 — 209s 330ms/step - accuracy: 0.9703 - loss: 0.0887 - val_accuracy: 0.9908 - val_loss: 0.0336
Epoch 7/15
469/469 — 204s 334ms/step - accuracy: 0.9719 - loss: 0.0826 - val_accuracy: 0.9910 - val_loss: 0.0318
Epoch 8/15
469/469 — 150s 321ms/step - accuracy: 0.9753 - loss: 0.0705 - val_accuracy: 0.9891 - val_loss: 0.0381
Epoch 9/15
469/469 — 156s 333ms/step - accuracy: 0.9778 - loss: 0.0656 - val_accuracy: 0.9906 - val_loss: 0.0345
Epoch 10/15
469/469 — 193s 315ms/step - accuracy: 0.9798 - loss: 0.0597 - val_accuracy: 0.9912 - val_loss: 0.0371
Epoch 11/15
469/469 — 200s 311ms/step - accuracy: 0.9801 - loss: 0.0563 - val_accuracy: 0.9912 - val_loss: 0.0313
Epoch 12/15
469/469 — 206s 321ms/step - accuracy: 0.9830 - loss: 0.0504 - val_accuracy: 0.9910 - val_loss: 0.0315
Epoch 13/15
469/469 — 202s 321ms/step - accuracy: 0.9828 - loss: 0.0502 - val_accuracy: 0.9909 - val_loss: 0.0375
Epoch 14/15
469/469 — 201s 319ms/step - accuracy: 0.9844 - loss: 0.0476 - val_accuracy: 0.9919 - val_loss: 0.0337
Epoch 15/15
469/469 — 153s 326ms/step - accuracy: 0.9845 - loss: 0.0455 - val_accuracy: 0.9910 - val_loss: 0.0379

Рисунок 64. Обучение модели

Далее выполним оценку точности на тестовых данных:

```
[ ] # Оценка точности на тестовых данных
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test loss: 0.03708580136299133
Test accuracy: 0.9896000027656555

Рисунок 65. Оценка точности

После выполним построение графиков:

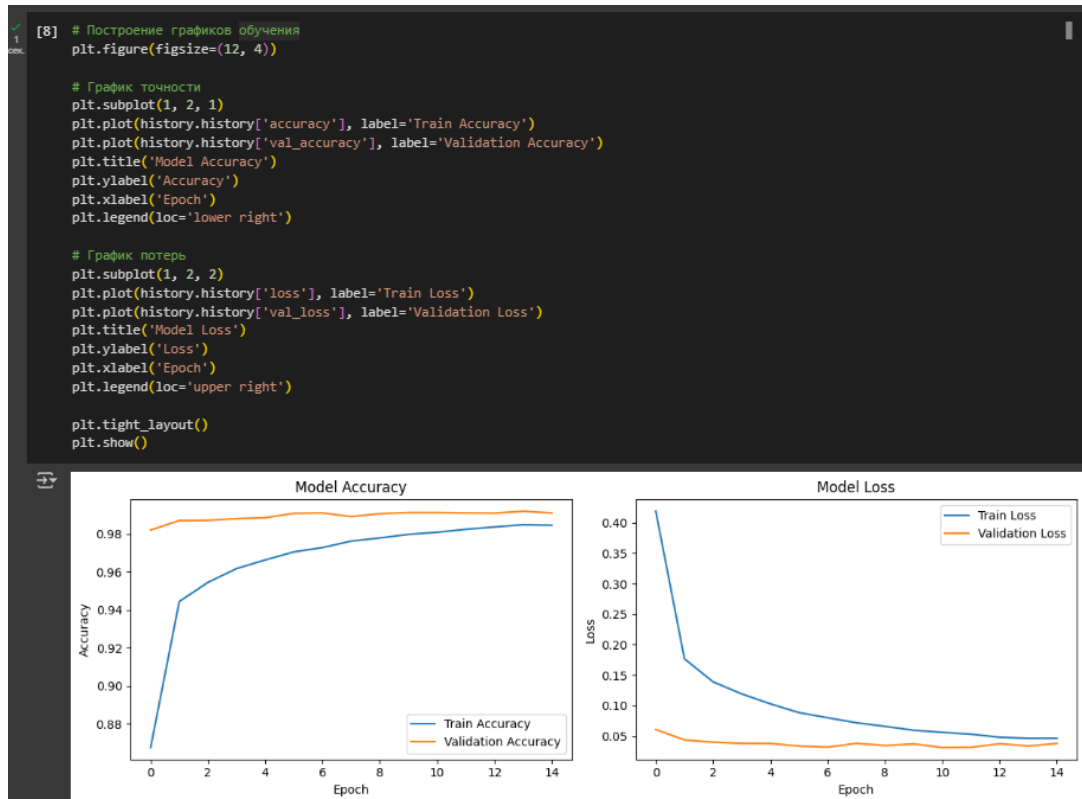


Рисунок 66. Построение графиков

Задание 2.

Условие: необходимо использовать датасет "Пассажиры автобуса", создать нейронную сеть для решения задачи классификации пассажиров на входящих и выходящих.

Добиться точности работы модели выше 90% на проверочной выборке.

Для этого, для начала выполним загрузку необходимых библиотек.

```
✓ 26 сек. # методы для отрисовки изображений
from PIL import Image

# Для отрисовки графиков
import matplotlib.pyplot as plt

# Для генерации случайных чисел
import random

# Библиотека работы с массивами
import numpy as np

# Для работы с файлами
import os

# импортируем модуль для загрузки данных
import gdown

# для разделения выборок
from sklearn.model_selection import train_test_split

# для создания сети
from tensorflow.keras.models import Sequential

# для создания слоев
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization

# для работы с изображениями
from tensorflow.keras.preprocessing import image

# оптимизатор
from tensorflow.keras.optimizers import Adam
```

Рисунок 68. Загрузка библиотек

Затем выполним загрузку датасета:

```
✓ 7 сек. [2] # загрузка датасета, укажем путь к базе в Google Drive, база в виде .zip-архива
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/14/bus.zip', None, quiet=True)

# Распакуем архив в директорию 'content/bus'
!unzip -q "bus.zip" -d /content/bus

# Папка с папками картинок, рассортированных по категориям
IMAGE_PATH = '/content/bus/'

# Получение списка папок, находящегося по адресу в скобках
os.listdir(IMAGE_PATH)

→ ['Выходящий', 'Входящий']
```

Рисунок 69. Загрузка датасета

После выполним определение списка имен классов и определение количества классов и выведем результат.

```
✓ 0 сек. [5] # Определение
CLASS_LIST = sorted(os.listdir(IMAGE_PATH))

# Определение количества классов
CLASS_COUNT = len(CLASS_LIST)

# Проверка результата
print(f'Количество классов: {CLASS_COUNT}, метки классов: {CLASS_LIST}')

→ Количество классов: 2, метки классов: ['Входящий', 'Выходящий']
```

Рисунок 70. Вывод результата

Далее получим список файлов для каждого класса:

```
[4] # Получения списка файлов для каждого класса

for cls in CLASS_LIST:
    print(cls, ':', os.listdir(f'{IMAGE_PATH}{cls}'))
```

Входящий : ['05621.jpg', '05975.jpg', '00902.jpg', '02139.jpg', '04833.jpg', '04259.jpg', '01807.jpg', '04531.jpg', '06348.jpg', '0267.jpg', '00902.jpg', '02139.jpg', '01807.jpg', '00578.jpg', '02474.jpg', '00243.jpg', '01955.jpg', '01808.jpg', '01856.jpg', '008.jpg']
Выходящий : ['00902.jpg', '02139.jpg', '01807.jpg', '00578.jpg', '02474.jpg', '00243.jpg', '01955.jpg', '01808.jpg', '01856.jpg', '008.jpg']

Рисунок 71. Список файлов для каждого класса

Далее выполним отрисовку изображений и получим две случайные картинки, одна из которых будет соответствовать входящему, а другая выходящему.

```
[6] # Создание заготовки для изображений всех классов
fig, axs = plt.subplots(1, CLASS_COUNT, figsize=(10, 5))

# Для всех номеров классов:
for i in range(CLASS_COUNT):

    # Формирование пути к папке содержимого класса
    car_path = f'{IMAGE_PATH}{CLASS_LIST[i]}/'

    # Выбор случайного фото из i-го класса
    img_path = car_path + random.choice(os.listdir(car_path))

    # Отображение фотографии (подробнее будет объяснено далее)
    axs[i].set_title(CLASS_LIST[i])
    axs[i].imshow(image.open(img_path))
    axs[i].axis('off')

# Отрисовка всего полотна
plt.show()
```



Рисунок 72. Отображение входящих и выходящих

Далее выполним вывод общего размера базы обучения:

```
[7] data_files = [] # Список путей к файлам картинок
    data_labels = [] # Список меток классов, соответствующих файлам

    for class_label in range(CLASS_COUNT): # Для всех классов по порядку номеров (их меток)
        class_name = CLASS_LIST[class_label] # Выборка имени класса из списка имен
        class_path = IMAGE_PATH + class_name # Формирование полного пути к папке с изображениями класса
        class_files = os.listdir(class_path) # Получение списка имен файлов с изображениями текущего класса
        print(f'Размер класса {class_name} составляет {len(class_files)} фото')

        # Добавление к общему списку всех файлов класса с добавлением родительского пути
        data_files += [f'{class_path}/{file_name}' for file_name in class_files]

        # Добавление к общему списку меток текущего класса - их ровно столько, сколько файлов в классе
        data_labels += [class_label] * len(class_files)

    print()
    print('Общий размер базы для обучения:', len(data_labels))
```

Размер класса Входящий составляет 6485 фото
Размер класса Выходящий составляет 2596 фото
Общий размер базы для обучения: 9081

Рисунок 73. Размер базы обучения

Далее выполним преобразование всех изображений в numpy-массив нужного размера, после чего выведем формы массива x и y.

```
from tensorflow.keras.utils import to_categorical

target_size = (128, 128)

X = []

# Преобразуем все изображения в numpy-массив нужного размера
for path in data_files:
    img = image.load_img(path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = img_array / 255.0
    X.append(img_array)

# Преобразуем список изображений в массив numpy
X = np.array(X)

# Преобразуем метки в numpy-массив
y = np.array(data_labels)

print(f'Форма массива X: {X.shape}')
print(f'Форма массива y: {y.shape}')
```

Форма массива X: (9081, 128, 128, 3)
Форма массива y: (9081,)

Рисунок 74. Преобразование изображений

Затем выполним разделение на обучающую и тестовую выборки

```
# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

Рисунок 75. Разделение на обучающую и тестовую выборки

После чего посмотрим на результат разделения:

```
print(f'Размер обучающей выборки: {X_train.shape}, метки: {y_train.shape}')
print(f'Размер тестовой выборки: {X_test.shape}, метки: {y_test.shape}')
```

Размер обучающей выборки: (7264, 128, 128, 3), метки: (7264,)
Размер тестовой выборки: (1817, 128, 128, 3), метки: (1817,)

Рисунок 76. Результат разделения на обучающую и тестовую выборки

Далее выполним создание модели и выполним ее компиляцию.

```
[18] # Преобразуем метки в one-hot
y_train_hot = to_categorical(y_train, num_classes=CLASS_COUNT)
y_test_hot = to_categorical(y_test, num_classes=CLASS_COUNT)

# Создание модели
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(target_size, 3)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(CLASS_COUNT, activation='softmax')
])

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

Рисунок 77. Создание модели

Далее выполним разделение обучающей выборки на train и validation:

```
✓ 1 [24] # Разделение обучающей выборки на train и validation
    OK X_train_final, X_val, y_train_final, y_val = train_test_split(
        X_train, y_train_hot, test_size=0.2, random_state=42)
```

Рисунок 78. Разделение обучающей выборки на train и validation

После чего выполним обучение модели:

```
[ ] # Обучение модели
history = model.fit(
    X_train_final, y_train_final,
    epochs=30,
    batch_size=32,
    validation_data=(X_val, y_val),
    verbose=1
)
```

Epoch 2/30
182/182 ————— 4s 22ms/step - accuracy: 0.9383 - loss: 0.1635 - val_accuracy: 0.9540 - val_loss: 1.0282
Epoch 3/30
182/182 ————— 4s 21ms/step - accuracy: 0.9653 - loss: 0.0952 - val_accuracy: 0.9215 - val_loss: 0.2050
Epoch 4/30
182/182 ————— 4s 21ms/step - accuracy: 0.9739 - loss: 0.0671 - val_accuracy: 0.9794 - val_loss: 0.0599
Epoch 5/30
182/182 ————— 4s 23ms/step - accuracy: 0.9877 - loss: 0.0415 - val_accuracy: 0.9725 - val_loss: 0.0764
Epoch 6/30
182/182 ————— 4s 22ms/step - accuracy: 0.9902 - loss: 0.0320 - val_accuracy: 0.9711 - val_loss: 0.0784
Epoch 7/30
182/182 ————— 5s 22ms/step - accuracy: 0.9923 - loss: 0.0233 - val_accuracy: 0.9725 - val_loss: 0.0779
Epoch 8/30
182/182 ————— 4s 22ms/step - accuracy: 0.9944 - loss: 0.0229 - val_accuracy: 0.9759 - val_loss: 0.0786
Epoch 9/30
182/182 ————— 5s 21ms/step - accuracy: 0.9928 - loss: 0.0203 - val_accuracy: 0.9800 - val_loss: 0.0611
Epoch 10/30
182/182 ————— 4s 23ms/step - accuracy: 0.9896 - loss: 0.0306 - val_accuracy: 0.9683 - val_loss: 0.0928
Epoch 11/30
182/182 ————— 4s 23ms/step - accuracy: 0.9861 - loss: 0.0527 - val_accuracy: 0.9718 - val_loss: 0.0774
Epoch 12/30
182/182 ————— 4s 23ms/step - accuracy: 0.9883 - loss: 0.0330 - val_accuracy: 0.9738 - val_loss: 0.0845
Epoch 13/30
182/182 ————— 5s 22ms/step - accuracy: 0.9950 - loss: 0.0146 - val_accuracy: 0.9862 - val_loss: 0.0440
Epoch 14/30
182/182 ————— 5s 22ms/step - accuracy: 0.9982 - loss: 0.0066 - val_accuracy: 0.9917 - val_loss: 0.0358
Epoch 15/30
182/182 ————— 4s 22ms/step - accuracy: 0.9985 - loss: 0.0069 - val_accuracy: 0.9787 - val_loss: 0.0663
Epoch 16/30
182/182 ————— 4s 23ms/step - accuracy: 0.9977 - loss: 0.0064 - val_accuracy: 0.9752 - val_loss: 0.0885
Epoch 17/30
182/182 ————— 5s 21ms/step - accuracy: 0.9938 - loss: 0.0162 - val_accuracy: 0.9814 - val_loss: 0.0525
Epoch 18/30
182/182 ————— 5s 23ms/step - accuracy: 0.9983 - loss: 0.0095 - val_accuracy: 0.9794 - val_loss: 0.0612
Epoch 19/30
182/182 ————— 4s 23ms/step - accuracy: 0.9949 - loss: 0.0134 - val_accuracy: 0.9821 - val_loss: 0.0777
Epoch 20/30
182/182 ————— 5s 22ms/step - accuracy: 0.9954 - loss: 0.0133 - val_accuracy: 0.9759 - val_loss: 0.0820
Epoch 21/30
182/182 ————— 5s 22ms/step - accuracy: 0.9982 - loss: 0.0302 - val_accuracy: 0.9725 - val_loss: 0.0968
Epoch 22/30
182/182 ————— 4s 24ms/step - accuracy: 0.9866 - loss: 0.0355 - val_accuracy: 0.9511 - val_loss: 0.2089
Epoch 23/30
182/182 ————— 4s 23ms/step - accuracy: 0.9814 - loss: 0.0571 - val_accuracy: 0.9745 - val_loss: 0.0764
Epoch 24/30
182/182 ————— 4s 22ms/step - accuracy: 0.9913 - loss: 0.0222 - val_accuracy: 0.9828 - val_loss: 0.0589
Epoch 25/30
182/182 ————— 5s 22ms/step - accuracy: 0.9947 - loss: 0.0139 - val_accuracy: 0.9835 - val_loss: 0.0594
Epoch 26/30
182/182 ————— 5s 22ms/step - accuracy: 0.9973 - loss: 0.0059 - val_accuracy: 0.9821 - val_loss: 0.0639
Epoch 27/30
182/182 ————— 4s 22ms/step - accuracy: 0.9967 - loss: 0.0120 - val_accuracy: 0.9800 - val_loss: 0.0735
Epoch 28/30
182/182 ————— 4s 23ms/step - accuracy: 0.9965 - loss: 0.0088 - val_accuracy: 0.9745 - val_loss: 0.0923
Epoch 29/30
182/182 ————— 5s 24ms/step - accuracy: 0.9976 - loss: 0.0097 - val_accuracy: 0.9821 - val_loss: 0.0556
Epoch 30/30
182/182 ————— 5s 23ms/step - accuracy: 0.9984 - loss: 0.0060 - val_accuracy: 0.9787 - val_loss: 0.0889

Рисунок 79. Обучение модели

Далее выполним оценку модели:

```
[ ] from sklearn.metrics import f1_score

# Предсказания модели на тестовой выборке
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)      # Получаем метки предсказаний
y_true = np.argmax(y_test_hot, axis=1)        # Преобразуем one-hot в метки

# Расчёт f1-метрики с учётом дисбаланса
f1 = f1_score(y_true, y_pred, average='weighted')

print(f'Точность модели на тестовой выборке: {f1:.4f}')
```

57/57 ————— 2s 21ms/step
Точность модели на тестовой выборке: 0.9790

Рисунок 80. Оценка модели

Задание 3.

Условие: необходимо использовать базу данных автомобилей, создать сеть с точностью распознавания не ниже 93% на проверочной выборке.

Для решения задачи можно использовать любой подход:

- модель без аугментации данных
- аугментация данных с помощью ImageDataGenerator
- аугментация данных с помощью самописного генератора изображений
- использовать готовую архитектуру из набора tf.keras.applications (Обратите внимание: на занятии мы не рассматривали данный модуль фреймворка Керас. Ваша задача: попробовать самостоятельно разобраться в принципах его работы. В разборе домашнего задания вы получите ссылку на ноутбук Базы Знаний УИИ, где подробно раскрывается вопрос использования готовых архитектур)

Для начала выполним загрузку необходимых библиотек:

```
[ ] import gdown
import os
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Dense,
    Conv2D,
    MaxPooling2D,
    Flatten,
    Dropout,
    BatchNormalization,
    Input,
    Activation,
    GlobalAveragePooling2D,
    LeakyReLU,
)

# оптимизатор Adam и SGD
from tensorflow.keras.optimizers import Adam, SGD

# Импорт предварительно обученной модели VGG19 из Keras
from tensorflow.keras.applications import VGG19

# Импорт генератора изображений, который используется для увеличения обучающего набора данных с помощью аугментации
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Рисунок 81. Загрузка библиотек

Далее выполним загрузку zip-архива с датасетом из облака на диск виртуальной машины colab:


```
[ ] # Загрузка zip-архива с датасетом из облака на диск виртуальной машины colab
gdown.download(
    "https://storage.yandexcloud.net/aiueducation/Content/base/15/middle_fmz.zip",
    None,
    quiet=True,
)

'middle_fmz.zip'

os.mkdir("cars")
# Распаковка zip-архива с датасетом из облака на диск виртуальной машины colab
!unzip -qo "middle_fmz.zip" -d cars/cars_train
```

Рисунок 82. Загрузка датасета

Далее определим размер деления выборки на тестовую, проверочную и обучающую:

```
# Размер деления выборки на тестовую, проверочную и обучающую
TEST_SPLIT = VAL_SPLIT = 0.1

# Пути к директориям с данными
TRAIN_PATH = Path("cars/cars_train")
VAL_PATH = Path("cars/cars_val")
TEST_PATH = Path("cars/cars_test")

# Создание директорий для тестовой и проверочной выборок
if not (TEST_PATH.exists() and VAL_PATH.exists()):
    TEST_PATH.mkdir(exist_ok=True)
    VAL_PATH.mkdir(exist_ok=True)

for classfolder in TRAIN_PATH.iterdir():
    classfolder_test = TEST_PATH / classfolder.name
    classfolder_val = VAL_PATH / classfolder.name

    classfolder_test.mkdir(exist_ok=True)
    classfolder_val.mkdir(exist_ok=True)

    files = list(classfolder.iterdir())
    len_class = len(files)
    test_len = int(len_class * TEST_SPLIT)
    val_len = int(len_class * VAL_SPLIT)

    # Распределение файлов:
    # - Первые test_len файлов - тестовая выборка
    # - Следующие val_len файлов - проверочная выборка
    # - Остальные остаются в обучающей выборке
    for i, img in enumerate(files):
        if i < test_len:
            img.rename(classfolder_test / img.name)
        elif i < test_len + val_len:
            img.rename(classfolder_val / img.name)
        else:
            break
```

Рисунок 83. Размер деления выборок

Далее выполним аугментацию и нормализацию данных:

```
# Аугментация и нормализация данных
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.05,
    zoom_range=0.2,
    brightness_range=(0.7, 1.3),
    horizontal_flip=True,
    rescale=1.0 / 255.0,
)

# Нормализация данных для тестовой и проверочной выборок
test_and_val_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
)
```

Рисунок 84. Нормализация данных

Далее выполним распределение изображений по классам, для обучающей выборки, проверочной и тестовой выборки.

```
[6] # Параметры изображения
IMG_HEIGHT = 188
IMG_WIDTH = 192
BATCH_SIZE = 64

# Обучающая выборка генерируется из папки обучающего набора
train_generator = train_datagen.flow_from_directory(
    # Путь к обучающим изображениям
    TRAIN_PATH,
    # Параметры требуемого размера изображения
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    # Размер батча
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
)

# Проверочная выборка генерируется из папки проверочного набора
validation_generator = test_and_val_datagen.flow_from_directory(
    VAL_PATH,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=True,
)

# Тестовая выборка генерируется из папки тестового набора
test_generator = test_and_val_datagen.flow_from_directory(
    TEST_PATH,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False,
)

Found 2745 images belonging to 3 classes.
Found 341 images belonging to 3 classes.
Found 341 images belonging to 3 classes.

print("Распределение изображений, обучающей выборки:", np.bincount(train_generator.classes))
print("Распределение изображений, проверочной выборки:", np.bincount(validation_generator.classes))
print("Распределение изображений, тестовой выборки:", np.bincount(test_generator.classes))

Распределение изображений, обучающей выборки: [872 929 944]
Распределение изображений, проверочной выборки: [188 116 117]
Распределение изображений, тестовой выборки: [188 116 117]
```

Рисунок 85. Распределение изображений по классам

Далее выполним вывод первых нескольких изображений из батча:

```
[ ] # Получаем один батч изображений и меток из генератора
images, labels = next(train_generator)

# Выводим первые несколько изображений из батча
num_images_to_show = 2
plt.figure(figsize=(10, 5))

for i in range(num_images_to_show):
    plt.subplot(1, num_images_to_show, i + 1)
    plt.imshow(images[i])
    plt.title(f"Label: {labels[i]}")
    plt.axis("off")

plt.show()
```

Рисунок 86. Вывод изображений

Далее создадим модель, используя готовую архитектуру vgg19. Для этого загрузим модель VGG19 без верхних слоев, с предобученными весами ImageNet.

```
# Для создания модели используем готовую архитектуру vgg19
# Загружаем модель VGG19 без верхних слоев, с предобученными весами ImageNet
base_model = VGG19(
    weights="imagenet", include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)
)

# Замораживаем все слои базовой модели (чтобы не обучались)
for layer in base_model.layers:
    layer.trainable = False

# Размораживаем последние 5 слоев для дообучения
for layer in base_model.layers[-5:]:
    layer.trainable = True

# Создаем модель на основе VGG19 с добавлением своих слоев классификации
model_vgg = Sequential(
    [
        base_model,
        GlobalAveragePooling2D(), # Глобальный усредняющий пулинг
        Dropout(0.6),
        Dense(512, activation="relu"),
        Dropout(0.6),
        Dense(3, activation="softmax")
    ]
)

# Компиляция модели с оптимизатором SGD и функцией потерь для многоклассовой классификации.
model_vgg.compile(
    # альтернативный оптимизатор (закомментирован)
    optimizer=SGD(learning_rate=0.003, momentum=0.9),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
```

Рисунок 87. Создание модели

Далее выведем структуру и параметры:

```
[ ] # Выведем структуру и параметры
model_vgg.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 3, 6, 512)	20,024,384
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262,656
dropout_3 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 3)	1,539

Total params: 20,286,979 (77.39 MB)
Trainable params: 9,703,427 (37.02 MB)
Non-trainable params: 10,583,552 (40.38 MB)

Рисунок 88. Структура модели

Далее выполним обучение модели:

```
[ ] # Обучим полученную модель
history_vgg = model_vgg.fit(
    train_generator,
    epochs=150,
    validation_data=validation_generator,
    verbose=1,
)

Epoch 122/150      22s 517ms/step - accuracy: 0.9933 - loss: 0.0200 - val_accuracy: 0.9150 - val_loss: 0.3029
Epoch 123/150      22s 512ms/step - accuracy: 0.9964 - loss: 0.0148 - val_accuracy: 0.9267 - val_loss: 0.2880
Epoch 124/150      22s 502ms/step - accuracy: 0.9938 - loss: 0.0184 - val_accuracy: 0.9238 - val_loss: 0.3682
Epoch 125/150      22s 503ms/step - accuracy: 0.9941 - loss: 0.0228 - val_accuracy: 0.9238 - val_loss: 0.3619
Epoch 126/150      22s 520ms/step - accuracy: 0.9966 - loss: 0.0147 - val_accuracy: 0.9179 - val_loss: 0.3878
Epoch 127/150      22s 519ms/step - accuracy: 0.9979 - loss: 0.0139 - val_accuracy: 0.9238 - val_loss: 0.4278
Epoch 128/150      22s 519ms/step - accuracy: 0.9949 - loss: 0.0160 - val_accuracy: 0.9296 - val_loss: 0.2958
Epoch 129/150      22s 509ms/step - accuracy: 0.9946 - loss: 0.0151 - val_accuracy: 0.9384 - val_loss: 0.3299
Epoch 130/150      22s 506ms/step - accuracy: 0.9943 - loss: 0.0189 - val_accuracy: 0.9288 - val_loss: 0.4287
Epoch 131/150      23s 532ms/step - accuracy: 0.9963 - loss: 0.0122 - val_accuracy: 0.9267 - val_loss: 0.3880
Epoch 132/150      22s 520ms/step - accuracy: 0.9955 - loss: 0.0179 - val_accuracy: 0.9120 - val_loss: 0.3447
Epoch 133/150      22s 516ms/step - accuracy: 0.9960 - loss: 0.0194 - val_accuracy: 0.9120 - val_loss: 0.3343
Epoch 134/150      22s 513ms/step - accuracy: 0.9896 - loss: 0.0241 - val_accuracy: 0.9238 - val_loss: 0.3788
Epoch 135/150      22s 501ms/step - accuracy: 0.9943 - loss: 0.0180 - val_accuracy: 0.9384 - val_loss: 0.3033
Epoch 136/150      22s 513ms/step - accuracy: 0.9952 - loss: 0.0157 - val_accuracy: 0.9179 - val_loss: 0.4095
Epoch 137/150      22s 520ms/step - accuracy: 0.9930 - loss: 0.0185 - val_accuracy: 0.9267 - val_loss: 0.3516
Epoch 138/150      23s 522ms/step - accuracy: 0.9940 - loss: 0.0156 - val_accuracy: 0.9150 - val_loss: 0.3834
Epoch 139/150      22s 516ms/step - accuracy: 0.9897 - loss: 0.0283 - val_accuracy: 0.9355 - val_loss: 0.3178
Epoch 140/150      22s 503ms/step - accuracy: 0.9964 - loss: 0.0159 - val_accuracy: 0.9150 - val_loss: 0.3949
Epoch 141/150      42s 519ms/step - accuracy: 0.9966 - loss: 0.0082 - val_accuracy: 0.9120 - val_loss: 0.3937
Epoch 142/150      40s 503ms/step - accuracy: 0.9930 - loss: 0.0205 - val_accuracy: 0.9238 - val_loss: 0.3926
Epoch 143/150      22s 511ms/step - accuracy: 0.9984 - loss: 0.0072 - val_accuracy: 0.9326 - val_loss: 0.3917
Epoch 144/150      22s 514ms/step - accuracy: 0.9983 - loss: 0.0075 - val_accuracy: 0.9003 - val_loss: 0.5115
Epoch 145/150      22s 515ms/step - accuracy: 0.9950 - loss: 0.0134 - val_accuracy: 0.9150 - val_loss: 0.4005
Epoch 146/150      22s 516ms/step - accuracy: 0.9971 - loss: 0.0095 - val_accuracy: 0.9091 - val_loss: 0.3934
Epoch 147/150      21s 498ms/step - accuracy: 0.9952 - loss: 0.0140 - val_accuracy: 0.9326 - val_loss: 0.4134
Epoch 148/150      42s 514ms/step - accuracy: 0.9923 - loss: 0.0186 - val_accuracy: 0.9208 - val_loss: 0.3644
Epoch 149/150      22s 514ms/step - accuracy: 0.9942 - loss: 0.0132 - val_accuracy: 0.9326 - val_loss: 0.2607
Epoch 150/150      22s 513ms/step - accuracy: 0.9969 - loss: 0.0126 - val_accuracy: 0.9355 - val_loss: 0.3100
```

Рисунок 89. Обучение модели

Далее построим графики, для того чтобы посмотреть на ход обучения модели:

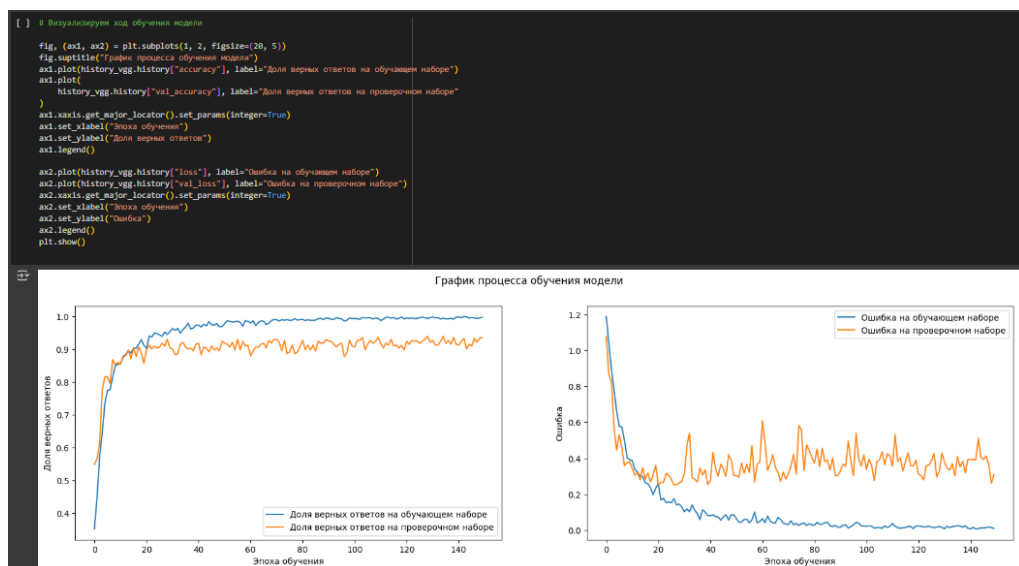


Рисунок 90. Графики обучения

Далее проверим точность обучающей, тестовой и проверочной выборок:

```
[ ] _, test_acc = model_vgg.evaluate(test_generator)

6/6 ————— 1s 178ms/step - accuracy: 0.9141 - loss: 0.4854

[ ] print(
    f"Точность на обучающей выборке: {history_vgg.history['accuracy'][-1] * 100:.2f},\n"
    f"точность на проверочной выборке: {history_vgg.history['val_accuracy'][-1] * 100:.2f},\n"
    f"точность на тестовой выборке: {test_acc * 100:.2f}."
)

Точность на обучающей выборке: 99.71,
точность на проверочной выборке: 93.55,
точность на тестовой выборке: 91.50.
```

Рисунок 91. Точность выборки

Ссылка на папку с гугл диском, в которой содержатся все выполняемые файлы:

https://drive.google.com/drive/folders/1q0vBhFsbjSJEXt7AeUDecJlKPq7JАНhk?usp=drive_link

Вывод: в процессе выполнения работы были изучены архитектура и принципы работы сверточных нейронных сетей.