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:

Говоров Егор Юрьевич

3 курс, группа ИВТ-б-о-22-1,

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

_____

(подпись)

Руководитель практики:

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

_____

(подпись)

Ставрополь, 2024 г.

Тема: Сверточные нейронные сети.

Цель: изучить архитектуру и принципы работы сверточных нейронных сетей.

Ссылка: https://github.com/Artorias1469/NN_3.git

Ход работы:

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:

```
[ ] # 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 1. Загрузка датасета MNIST

Далее выполним загрузку необходимых библиотек:

```python
# Стандартная библиотека
import random
import warnings

# Сторонние библиотеки
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

# TensorFlow / Keras
from tensorflow.keras import utils

# Магическая команда Jupyter
%matplotlib inline  # Вывод изображений в ноутбуке, а не в консоль или файл

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

Рисунок 2. Загрузка библиотек

Далее выполним вывод изображений каждого класса для ознакомления с датасетом:

```python
# вывод изображений каждого класса для ознакомления с датасетом

fig, axs = 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
  axs[i].imshow(Image.fromarray(img), cmap='gray') # выводим изображение

plt.show()
```

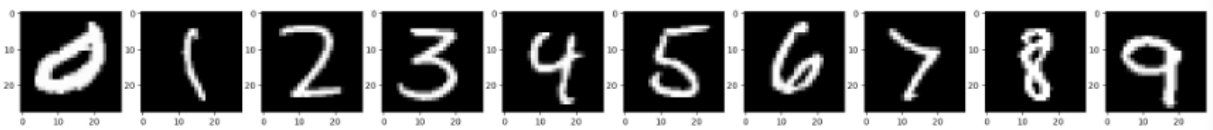

Рисунок 3. Вывод изображений Далее

посмотрим форматы выборок перед обучением:

```python
# 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 OHE
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 4. Форматы выборок

Далее выполним создание модели и выведем ее структуру:

```python
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras.optimizers import Adam


# 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 (пуллинг)
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"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
batch_normalization (BatchNormalization)	(None, 26, 26, 32)	128
conv2d_1 (Conv2D)	(None, 24, 24, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 12, 12, 64)	0
dropout (Dropout)	(None, 12, 12, 64)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 128)	1,179,776
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

```
Total params: 1,200,010 (4.58 MB)
Trainable params: 1,199,946 (4.58 MB)
Non-trainable params: 64 (256.00 B)
```

Рисунок 5. Создание модели Затем выполним обучение модели:

```python
# 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 6. 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: 0.037876155227422714
Test accuracy: 0.9909999966621399
```

Рисунок 7. Оценка точности После

выполним построение графиков:

```
# Построение графиков обучения
plt.figure(figsize=(12, 4))

# График точности
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='lower right')

# График потерь
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()
```

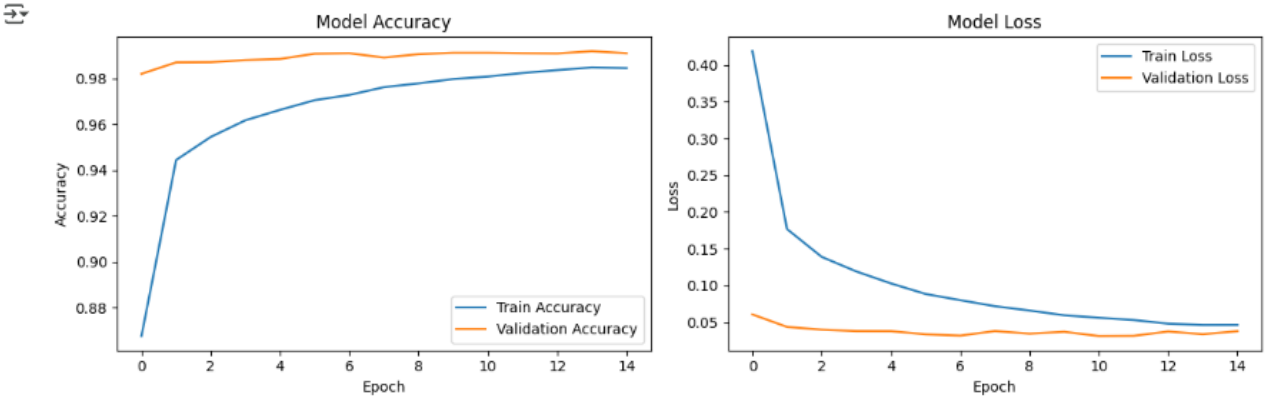

Рисунок 8. Построение графиков

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.

```
[ ] # Стандартная библиотека
    import os  # Для работы с файлами
    import random  # Для генерации случайных чисел
    import warnings  # Подавление предупреждений
    import zipfile  # работа с zip-архивами

    warnings.filterwarnings("ignore")

    # Сторонние библиотеки
    import gdown  # Импортируем модуль для загрузки данных из интернета
    import matplotlib.pyplot as plt  # Для отрисовки графиков
    import numpy as np  # Для работы с массивами
    from PIL import Image  # Методы для отрисовки изображений
    from sklearn.model_selection import train_test_split  # Для разделения выборок

    # TensorFlow / Keras
    from tensorflow.keras.layers import (
        BatchNormalization,
        Conv2D,
        Dense,
        Dropout,
        Flatten,
        MaxPooling2D,
    )  # Для создания слоев нейронной сети
    from tensorflow.keras.models import Sequential  # Для создания сети
    from tensorflow.keras.optimizers import Adam  # Оптимизатор для обучения модели
    from tensorflow.keras.preprocessing import image  # Для работы с изображениями
```

Рисунок 9. Загрузка библиотек

Затем выполним загрузку датасета:

```
# загрузка датасета
if "bus.zip" in os.listdir():
    pass
else:
    gdown.download(
        "https://storage.yandexcloud.net/aiueducation/Content/base/l4/bus.zip",
        None,
        quiet=True,
    )
```

При распаковке архива код:

- Проверяет наличие папки bus — если она есть, архив не распаковывается повторно
- Если папки нет — распаковывает содержимое архива bus.zip в новую папку bus.
- Затем получает список подпапок или файлов в bus/ и выводит его на экран.

```
[ ] # Распакуем архив
    if "bus" in os.listdir(): # проверяем, есть ли папка bus в текущей директории
        pass
    else: # если папки нет, то создаем ее и распаковываем архив
        with zipfile.ZipFile("bus.zip", "r") as zip_ref:
            zip_ref.extractall("bus")

    # Папка с папками картинок
    IMAGE_PATH = "bus/"

    # Получение списка папок
    print(os.listdir(IMAGE_PATH))

    ['Выходящий', 'Входящий']
```

Рисунок 10. Загрузка датасета

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

```
# 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 11. Вывод результата

Далее получим список файлов для каждого класса:

```
# Получения списка файлов для каждого класса

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

```
Входящий : ['04694.jpg', '00994.jpg', '00821.jpg', '04495.jpg', '05657.jpg', '02371.jpg', '02797.jpg', '05736.jpg', '02635.jpg', '0462
Выходящий : ['00994.jpg', '00821.jpg', '02371.jpg', '01199.jpg', '00149.jpg', '02546.jpg', '01768.jpg', '02422.jpg', '01207.jpg', '011
```

Рисунок 12. Список файлов для каждого класса

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

```
# 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 обьяснено 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 13. Отображение входящих и выходящих

Далее выполним вывод общего размера базы обучения:

```
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 14. 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)          # Загружаем изображение
    img_array = image.img_to_array(img)                          # Переводим в массив numpy
    img_array = img_array / 255.0                                # Нормализация значений
    X.append(img_array)                                          # Добавляем к общему списку

# 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 15. 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 16. 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 17. Результат разделения на обучающую и тестовую выборки

Далее выполним создание модели и выполним ее компиляцию.

```
[ ] # 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 18. Создание модели

Далее выполним разделение обучающей выборки на train и validation:

```
[ ] # Разделение обучающей выборки на train и validation
    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 19. Разделение обучающей выборки на train и validation

После чего выполним обучение модели:

```python
# 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 1/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 24s 74ms/step - accuracy: 0.7688 - loss: 0.8891 - val_accuracy: 0.7302 - val_loss: 0.5431
Epoch 2/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.9373 - loss: 0.1605 - val_accuracy: 0.5162 - val_loss: 1.2133
Epoch 3/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.9712 - loss: 0.0789 - val_accuracy: 0.9580 - val_loss: 0.1210
Epoch 4/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - accuracy: 0.9832 - loss: 0.0439 - val_accuracy: 0.9828 - val_loss: 0.0562
Epoch 5/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 6s 23ms/step - accuracy: 0.9881 - loss: 0.0333 - val_accuracy: 0.9725 - val_loss: 0.0770
Epoch 6/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.9884 - loss: 0.0340 - val_accuracy: 0.9814 - val_loss: 0.0514
Epoch 7/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.9939 - loss: 0.0200 - val_accuracy: 0.9828 - val_loss: 0.0474
Epoch 8/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - accuracy: 0.9955 - loss: 0.0163 - val_accuracy: 0.9862 - val_loss: 0.0423
Epoch 9/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.9871 - loss: 0.0373 - val_accuracy: 0.9491 - val_loss: 0.1341
Epoch 10/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - accuracy: 0.9896 - loss: 0.0276 - val_accuracy: 0.9821 - val_loss: 0.0532
Epoch 11/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - accuracy: 0.9899 - loss: 0.0293 - val_accuracy: 0.9869 - val_loss: 0.0444
Epoch 12/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 5s 24ms/step - accuracy: 0.9957 - loss: 0.0118 - val_accuracy: 0.9869 - val_loss: 0.0447
Epoch 13/30
182/182 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - accuracy: 0.9936 - loss: 0.0185 - val_accuracy: 0.9759 - val_loss: 0.0681
```

Рисунок 20. Обучение модели

Далее выполним оценку модели:

```python
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 21. Оценка модели

```python
# Визуализация обучения
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

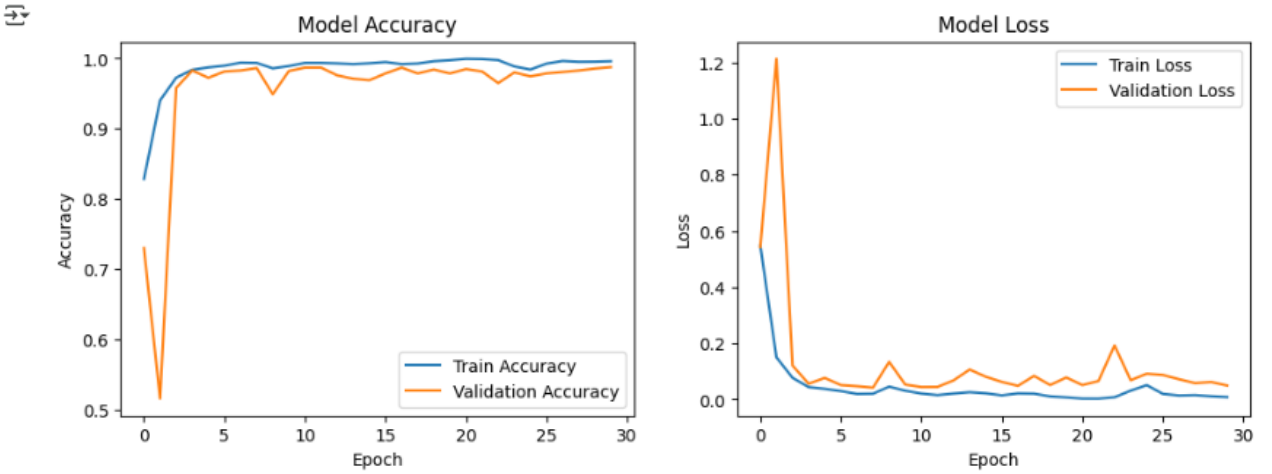

Рисунок 22. Визуализация

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 прицпах 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:

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

warnings.filterwarnings("ignore")

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

# TensorFlow / Keras
from tensorflow.keras.applications import VGG19
from tensorflow.keras.layers import (
    Activation,
    BatchNormalization,
    Conv2D,
    Dense,
    Dropout,
    Flatten,
    GlobalAveragePooling2D,
    Input,
    LeakyReLU,
    MaxPooling2D,
)
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Рисунок 23. Загрузка библиотек

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

Загрузка и распаковка датасета

Код проверяет, есть ли в текущей директории файл с именем middle_fmr.zip.

- Если файл уже присутствует, загрузка не выполняется (пропускается).
- Если файла нет, с помощью gdown скачивается zip-архив по заданной ссылке и сохраняется в текущую папку.

```python
# Загрузка zip-архива с датасетом из облака на диск виртуальной машины colab
if "middle_fmr.zip" in os.listdir():
    pass
else:
    # Загрузка zip-архива с датасетом из облака на диск
    gdown.download(
        "https://storage.yandexcloud.net/aiueducation/Content/base/l5/middle_fmr.zip",
        None,
        quiet=True,
    )
```

```
'middle_fmr.zip'
```

Проверка, создание папки и распаковка архива

Код проверяет, существует ли папка с именем cars в текущей директории.

- Если папка уже есть, ничего не происходит (pass).
- Если папки нет, она создаётся с помощью os.mkdir("cars").

Далее открывается архив middle_fmr.zip в режиме чтения, и его содержимое распаковывается в папку cars/cars_train. Таким образом, данные из архива будут организованы в отдельной вложенной папке внутри cars.

```python
if "cars" in os.listdir():  # Проверка наличия папки
    pass
else:
    # Создание папки cars для хранения датасета
    os.mkdir("cars")
    # Распаковка zip-архива
    with zipfile.ZipFile("middle_fmr.zip", "r") as zip_ref:
        zip_ref.extractall("cars/cars_train")
```

Рисунок 24. 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
TRAIN_PATH = Path("cars/cars_train")
VAL_PATH = Path("cars/cars_val")
TEST_PATH = Path("cars/cars_test")

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)

        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 25. Размер деления выборок

Далее выполним аргументацию и нормализацию данных:

```
# Аугументация 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,
)

test_and_val_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
)
```

Рисунок 26. Нормализация данных

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

```
# 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.
```

Следующий код выводит распределение количества изображений по классам в каждой из выборок: обучающей, проверочной и тестовой.

Функция np.bincount() подсчитывает, сколько раз встречается каждый класс (числовой индекс) в списке классов, который хранится в атрибуте .classes у каждого генератора.

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

```
Распределение изображений для классов, обучающей выборки: [872 929 944]
Распределение изображений для классов, проверочной выборки: [108 116 117]
Распределение изображений для классов, тестовой выборки: [108 116 117]
```

Рисунок 27. 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 28. Вывод изображений

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

```python
# Загружаем модель VGG19 без верхних слоев (головы), 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,                     # Базовая модель VGG19
        GlobalAveragePooling2D(),       # Глобальный усредняющий пуллинг
        Dropout(0.6),                   # Dropout
        Dense(512, activation="relu"),  # Полносвязный слой с ReLU
        Dropout(0.6),                   # Dropout
        Dense(3, activation="softmax")  # Выходной слой для 3 классов
    ]
)

# Компиляция модели с оптимизатором SGD и функцией потерь для многоклассовой классификации.
model_vgg.compile(
    # optimizer=Adam(learning_rate=0.00001),  # альтернативный оптимизатор (закомментирован)
    optimizer=SGD(learning_rate=0.003, momentum=0.9),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
```

Рисунок 29. Создание модели

Далее выведем структуру и параметры:

```python
# 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: 20,288,579 (77.39 MB)
Trainable params: 9,703,427 (37.02 MB)
Non-trainable params: 10,585,152 (40.38 MB)
```

Рисунок 30. 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 1/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 30s 623ms/step - accuracy: 0.3449 - loss: 1.2925 - val_accuracy: 0.5484 - val_loss: 1.0763
Epoch 2/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 500ms/step - accuracy: 0.4160 - loss: 1.0622 - val_accuracy: 0.5660 - val_loss: 0.8664
Epoch 3/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 516ms/step - accuracy: 0.5551 - loss: 0.9139 - val_accuracy: 0.6158 - val_loss: 0.8235
Epoch 4/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 511ms/step - accuracy: 0.6155 - loss: 0.8228 - val_accuracy: 0.7771 - val_loss: 0.5708
Epoch 5/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 519ms/step - accuracy: 0.7251 - loss: 0.6853 - val_accuracy: 0.8152 - val_loss: 0.4455
Epoch 6/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 503ms/step - accuracy: 0.7769 - loss: 0.5714 - val_accuracy: 0.8152 - val_loss: 0.5319
Epoch 7/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 42s 517ms/step - accuracy: 0.7452 - loss: 0.6268 - val_accuracy: 0.7947 - val_loss: 0.4582
Epoch 8/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 40s 502ms/step - accuracy: 0.8071 - loss: 0.5227 - val_accuracy: 0.8680 - val_loss: 0.3591
Epoch 9/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 506ms/step - accuracy: 0.8453 - loss: 0.4166 - val_accuracy: 0.8534 - val_loss: 0.3749
Epoch 10/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 506ms/step - accuracy: 0.8469 - loss: 0.4055 - val_accuracy: 0.8592 - val_loss: 0.3814
Epoch 11/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 520ms/step - accuracy: 0.8620 - loss: 0.3697 - val_accuracy: 0.8534 - val_loss: 0.3432
Epoch 12/150
43/43 ━━━━━━━━━━━━━━━━━━━━ 22s 517ms/step - accuracy: 0.8670 - loss: 0.3514 - val_accuracy: 0.8798 - val_loss: 0.3059
```

Рисунок 31. Обучение модели

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

```
# Визуализируем ход обучения модели

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

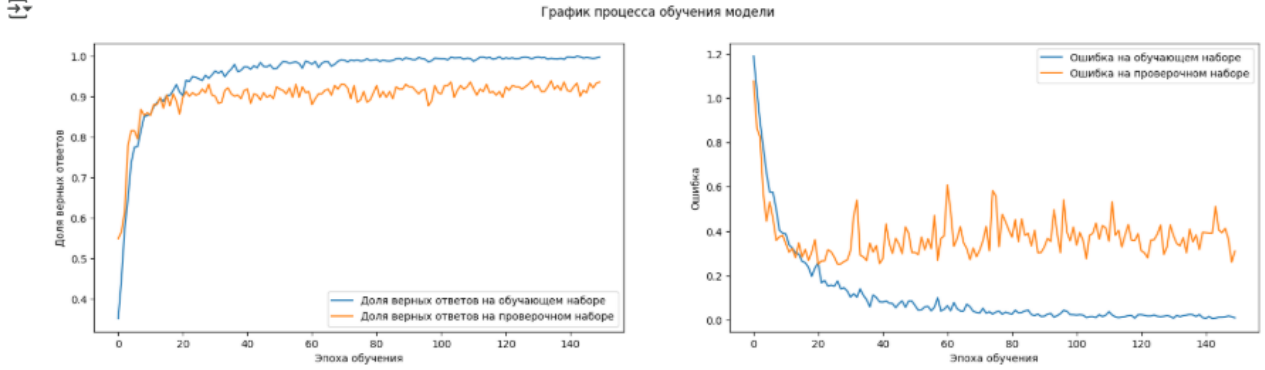

Рисунок 32. Графики обучения

Далее проверим точность обучающей, тестовой и проверочной выборок:

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

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 170ms/step - accuracy: 0.9141 - loss: 0.4854
```

Вывод итоговой точности модели

В этом блоке выводятся основные показатели точности модели:

- Точность на обучающей выборке (accuracy) — насколько хорошо модель обучилась на тренировочных данных.
- Точность на проверочной выборке (val_accuracy) — качество модели на валидационных данных, используемых для контроля переобучения.
- Точность на тестовой выборке (test_acc) — итоговая оценка модели на новых, ранее не виденных данных.

```
[ ] 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 33. Точность выборки

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