Лабораторная "Сверточные нейронные сети"

В рамках этой работы должны быть выполнены следующие этапы.

1. Создание сверточной сети для классификации изображений.
2. Реализация аугментации данных для изображений.

Сверточные нейронные сети

На вход сверточной сети обычно подается тензор размерности d и размером $n_1 \times n_2 \times \cdots \times n_d$. Для задачи распознавания изображений обычно $d = 3$, по первым двум измерениям отсчитывается в высота и ширина изображения, по третьему — канал.

Сверточная сеть состоит из слоев свертки и слоев субдискретизации (в английской литературе: "pooling layer").

Слой свертки работает следующим образом. В слое указывается размер тензора, по которому будет проводиться свертка и количество выходов. Входной слой делится на тензоры, равные размеру свертки, это разделение происходит с заданными сдвигами по каждому измерению. Для каждый выделенный тензор пропускается через полносвязный слой, проходя далее через функцию активации, при этом для тензором веса сохраняются одни и те же. На выходе слоя будет получаться тоже тензор. Таким образом для слоя свертки необходимо задать следующие параметры: размер свертки, сдвиги по каждому измерению, количество выходов и функцию активации. Веса полносвязного слоя свертки являются обучаемыми параметрами.

Слой субдискретизации работает следующим образом. Входной слой делится на тензоры заданного размера, после чего каждый тензор преобразуется в единственное значение нелинейным преобразованием. Наиболее употребительным является выбор максимума по тензору. Обычно при разделении на тензоры сдвиг по каждому измерению равен размеру тензора по этому измерению, то есть полученные тензоры не пересекаются.

Слои свертки и слои субдискретизации чередуются несколько раз, после чего выход последнего слоя пропускается через полносвязную сеть.

Аугментация данных

Аугментация данных — увеличение изначального набора данных засчёт применения к нему геометрических, цветовых/яркостных искажений, замены фона, добавления бликов, шумов и т.д.

Информация, содержащаяся в изображениях, всегда избыточна, то есть незначительные изменения данных не должны менять смысл изображения. Это означает, что хорошая модель должна быть устойчива по отношению к таким небольшим изменениям данных. Также это означает, что при обучении модели можно расширять наборы данных такими слегка измененными наблюдениями, сохраняя изначальные метки. Это позволит расширить набор данных, а также сделать модель более устойчивой к незначительным изменениям.

Для изображений используются следующие методы аугментации.

1. Случайные сдвиги изображений. Изображение сдвигается на некоторую часть от длины и ширины, при этом часть изображения обрезается, а другая часть дополняется. Например, используется дополнение отраженным изображением или постоянным значением.
2. Случайные повороты изображений. Аналогично случайному сдвигу.
3. Отражение по горизонтали и/или вертикали.
4. Случайное изменение цветовой составляющей: яркости, контрастности, насыщенности и т.д.

Примеры

В данном разделе приведены примеры реализации сверточных нейронных сетей для классификации на языке Python, а также реализация различных методов аугментации изображений.

Для реализации нейронных сетей использовался фреймворк Tensorflow, в частности его модуль Keras, который предоставляет интерфейс для создания нейронных сетей, программного составления графа вычислений для обратного распространения ошибки и прочего.

Для анализа данных использовались библиотеки pandas и numpy, для визуализации использовались библиотеки matplotlib.

В качестве среды выполнения использовался сервис Google Colab, который предоставляет бесплатную возможность запуска файлов Jupiter Notebook на виртуальных машинах компании Google.

Для выполнения работы не обязательно использовать именно эту среду выполнения.

Сверточная нейронная сеть для классификации

В качестве примера для классификации изображений возьмем набор данных CIFAR-10.

Набор данных CIFAR-10 состоит из 60000 цветных изображений размером 32x32, представляющих 10 различных классов по 6000 изображений в каждом классе.

В каждом изображении 3 канала, таким

В наборе данных присутствуют следующие классы:

- самолёт,
- автомобиль,
- птица,
- кошка,
- олень,
- собака,
- лягушка,
- лошадь,
- корабль,
- грузовик.

Данные разделены на тестовую и обучающую выборки: 10000 тестовых изображений и 50000 обучающих изображений. Тестовый набор содержит по 1000 случайно выбранных изображений каждого класса.

In [2]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, losses, datasets
import matplotlib.pyplot as plt
```

Данные скачаем из модуля `tensorflow.datasets`. Каждое изображение там представлено тензором размерности (32, 32, 3). Каждый пиксель в каждом канале имеет значение от 0 до 255.

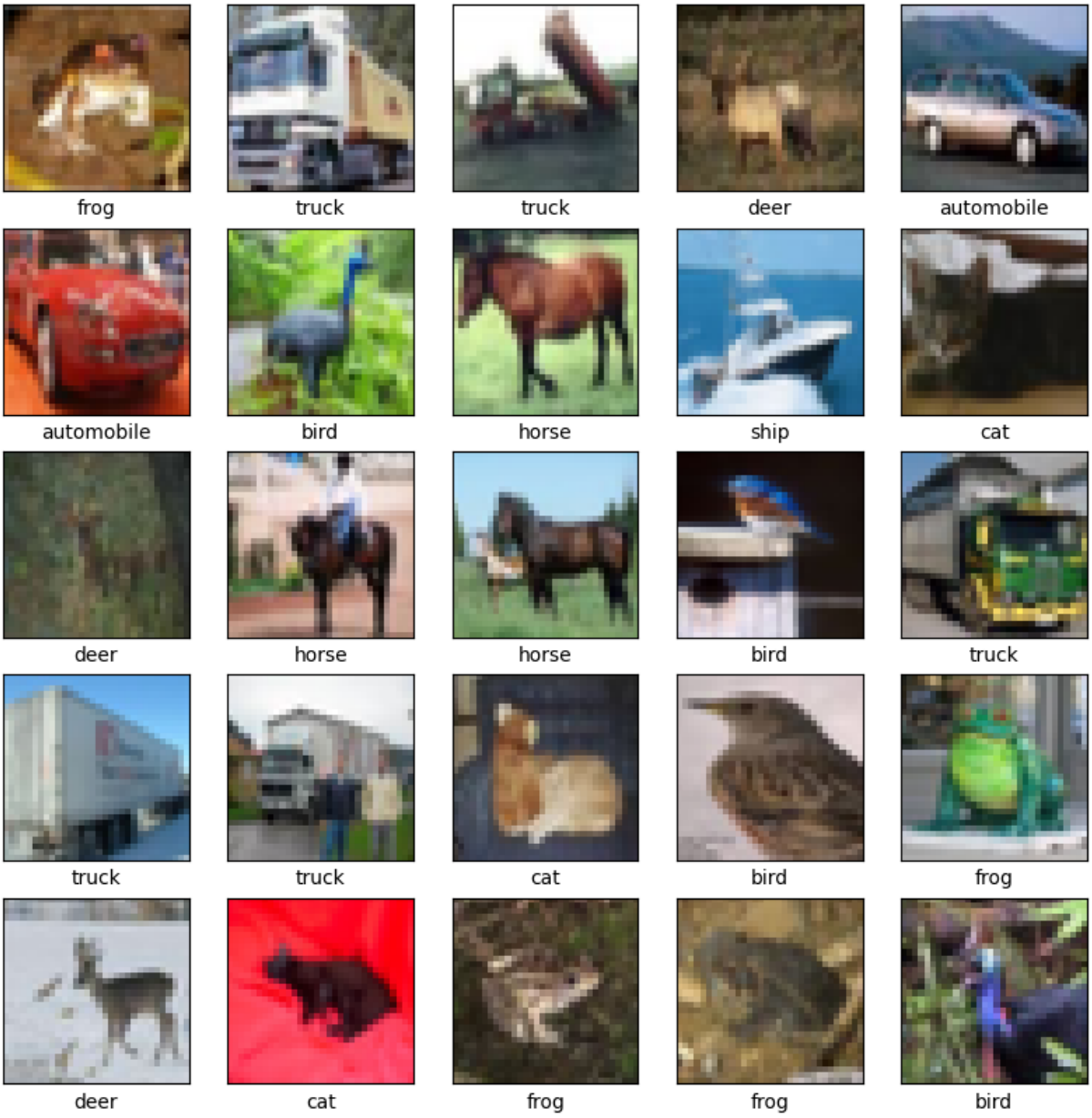
In [3]:
```
(train_images, train_labels), (test_images, test_labels) = \
    datasets.cifar10.load_data()

# названия классов
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

Посмотрим немного на данные.

In [4]:
```
# выберем 25 изображений для визуализации

```

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



Теперь создадим нейросеть. Сначала опишем ее структуру.

1. На входе тензор размерности (32, 32, 3).
2. Каждое значение преобразуем в диапазон $[0; 1]$ делением на 255.
3. Сделаем слой свертки, после чего сделаем слой субдискретизации. Повторим это несколько раз.
4. Вытянем получившийся тензор в вектор.

5. Создадим полносвязную нейронную сеть.
6. На выходе будет 10 нейронов без функции активации.
7. Для получения распределения вероятности.

Создадим базовую модель со следующими параметрами:

1. слой свертки:
   - размер 3 на 3,
   - 32 канала,
   - шаг (stride) 1 по каждому направлению,
   - функция активации `RELU`,
   - без дополнения (padding);
2. слой субдискретизации:
   - размер 2 на 2,
   - берем максимум,
   - с шагом 2 по каждому направлению;
3. слой свертки:
   - размер 3 на 3,
   - 64 канала,
   - шаг (stride) 1 по каждому направлению,
   - функция активации `RELU`,
   - без дополнения (padding);
4. слой субдискретизации, такой же как в п. 2;
5. слой свертки, такой же как в п. 3;
6. вытягиваем тензор в вектор;
7. полносвязный слой с 64 нейронами и функцией активации `RELU`;
8. полносвязный слой с 10 нейронами без функции активации;
9. функция `softmax`.

In [5]:
```
cnn_model = tf.keras.Sequential([
    # Задаем свертку
    # 32 – количество каналов на выходе
    # (3, 3) – размер окна
    # по умолчанию stride'ы – (1, 1)
    # по умолчанию padding не используется
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),

    # Задаем пулинг
    # (2, 2) – размер окна
    # по умолчанию сдвиг идет на размер окна
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),

    # теперь вытянем все в линию
    # на вход подается тензор (32, 32, 3)
    #    – (ширина, высота, каналы)
```

```
    # после первой свертки получается (30, 30, 32)
    #    так как размер 3x3 и сдвиг 1x1
    # после 2x2 пулинга остается (15, 15, 32)
    # далее свертка делает (13, 13, 64), потом пулинг
    #    (6, 6, 64) и еще одна свертка
    # делает (4, 4, 64). Это вытягиваем в вектор
    #    длиной 4x4x64 = 1024.
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10),
    layers.Softmax()],
    name = "cnn_model"
)

cnn_model.summary()
```

```
Model: "cnn_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

 softmax (Softmax)           (None, 10)                0

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Чтобы сделать преобразотку данных, добавим дополнительный слой.

In [6]:
```
preprocessing = layers.Rescaling(1 / 255.)
```

Теперь составим целую модель.

In [7]:
```
base_model = tf.keras.Sequential([
    preprocessing,
    cnn_model],
```

```
    name = "base_model"
)
```

В качестве функции потерь возьмем перекрестную энтропию для нескольких классов.

Метки классов в датасете представлены номером класса (от 0 до 9). Для этого можно использовать реализацию SparseCategoricalCrossentropy из `tensorflow`.

Также будем отслеживать значения точности во время обучения.

In [8]:
```
base_model.compile(
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
```

Теперь запустим обучение модели. Для обучения нужно выбрать количество эпох, размер пакета для градиентного спуска.

Также добавим тестовую выборку для отслеживания качества работы модели.

Также измерим время обучения модели. Для этого вставим директиву `%%time` в начало ячейки.

In [ ]:
```
base_model_history = base_model.fit(train_images, train_labels, epochs=20,
            validation_data=(test_images, test_labels))
```

Теперь можем посмотреть информацию о процессе обучения в объекте, который вернула функция `fit`.

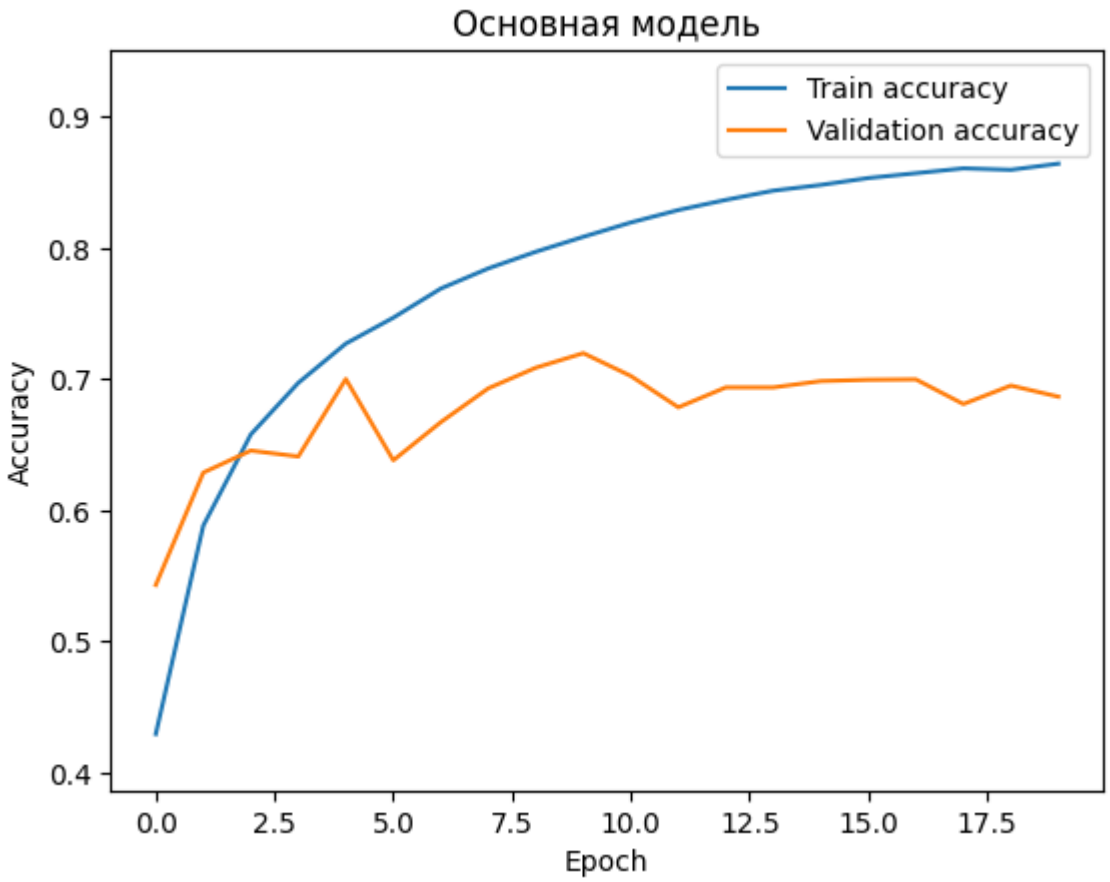
В его поле `history` лежат значения функции потерь и собранные метрики за каждую эпоху.

Напишем функцию для визуализации этих значений.

In [10]:
```
from typing import Dict, List, Tuple

def plot_history(
    history: Dict[str, List[float]],
    title: str = "",
    metric_name: str = 'loss',
    ylim: Tuple[float, float] = None):

    train_values = history[metric_name]
    plt.plot(train_values, label=f'Train {metric_name}')
    try:
        val_values = history['val_' + metric_name]
        plt.plot(val_values, label=f'Validation {metric_name}')
    except KeyError:
        val_values = []
    plt.title(title)

    all_values = train_values + val_values
    ylim = ylim or (0.9 * min(all_values), 1.1 * max(all_values))
```

```
    plt.ylim(ylim)

    plt.ylabel(metric_name.capitalize())
    plt.xlabel("Epoch")
    plt.legend(loc='best')
    plt.show()
```

In [11]:
```
plot_history(base_model_history.history, "Основная модель", "accuracy")
```



По графику видно, что модель работает на тренировочной выборке гораздо лучше, чем на тестовой, то есть модель переобучилась.

Итоговая точность:

In [25]:
```
base_model_history.history['val_accuracy'][-1]
```

Out[25]:
```
0.6866999864578247
```

Аугментация изображений

Аугментацию для изображений можно реализовать следующими способами.

1. В модель можно добавить несколько слоев, которые будут реализовывать случайное небольшое преобразование изображения во время обучения, но не

будут работать во время вычисления значения модели. Тогда во время обучения каждое подаваемое на вход изображение будет изменено во время исполнения.
2. Можно предварительно обработать изображения и сохранить измененый набор данных, на основе которого потом производить обучение.

В рамках этого примера реализуем первый вариант.

Реализуем следующие аугментации:

- случайное отражение изображния,
- случайный поворот изображения,
- случайное изменение контраста изображени,
- случайное изменение насыщенности изображения.

Для первых трех аугментаций в библиотеке `tensorflow` есть реализации:

- класс RandomFlip из `tesnorflow.keras.layers` реализует случайное отражение изображения;
- класс RandomRotation из `tesnorflow.keras.layers` реализует случайный поворот изображения;
- класс RandomContrast из `tesnorflow.keras.layers` реализует случайное изменение контраста изображения.

Сделаем слой, который будет применять эти аугментации.

In [26]:
```
augmentation = tf.keras.Sequential([
    # случайным образом отражаем по горизонтали
    layers.RandomFlip("horizontal"),
    # вращаем на +-18 градусов
    layers.RandomRotation(0.05),
    # изменим контраст на +-20%
    layers.RandomContrast((0.8, 1.2))
])
```

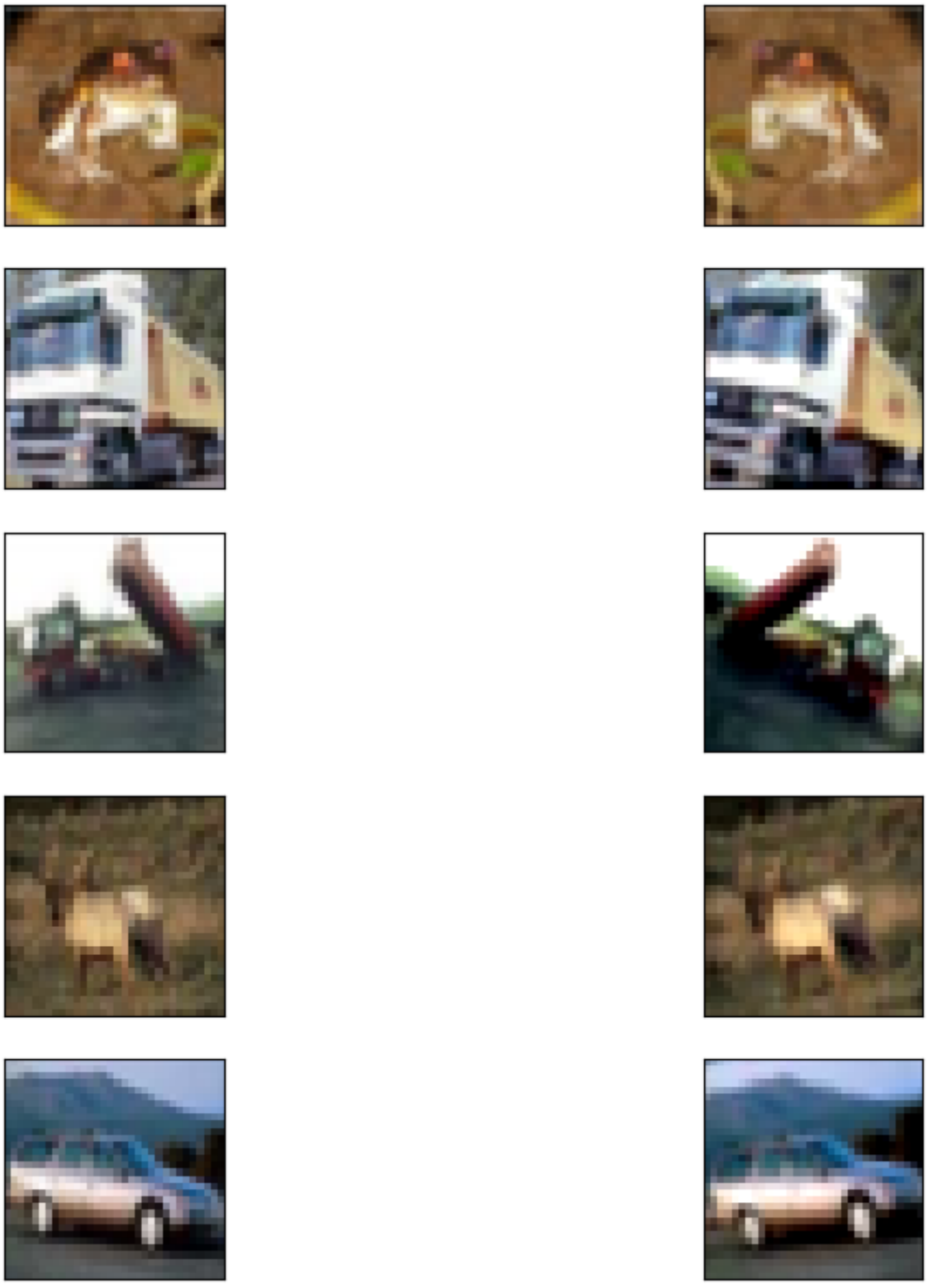
Посмотрим, как работает полученная аугментация. Для этого пропустим несколько изображений из прошлого примера через этот слой.

Реализация требует, чтобы значения каждого пиксела были в диапазоне $[0; 1]$, поэтому пропустим изображение через слой преодобработки из прошлого примера.

In [27]:
```
def show_augmentation(augmentation):
    plt.figure(figsize=(10,10))

    for i in range(5):
        original = preprocessing(train_images[i])
        augmented = augmentation(original, training=True)

        plt.subplot(5, 2, 2*i+1)
        plt.xticks([])
```

```
        plt.yticks([])
        plt.grid(False)
        plt.imshow(original)

        plt.subplot(5, 2, 2*i+2)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(augmented)

    plt.show()

show_augmentation(augmentation)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wi
th RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wi
th RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wi
th RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wi
th RGB data ([0..1] for floats or [0..255] for integers).
```



Готового слоя для случайного изменения насыщенности изображения нет, поэтому придется создать слой самостоятельно.

Один из способов — это создать класс-наследник класса `tensorflow.keras.layers.Layer` и переопределить в нем метод `call`, который реализует преобразование, производимое этим слоем.

Для изменения насыщенности изображения воспользуемся функцией stateless_random_saturation из модуля `tensorflow.image`.

Эта функция принимает границы интервала, из которого будет случайным образом выбрано значение, на которое умножается значение насыщенности изображения. Также функция принимает `seed` для генератора случайных чисел.

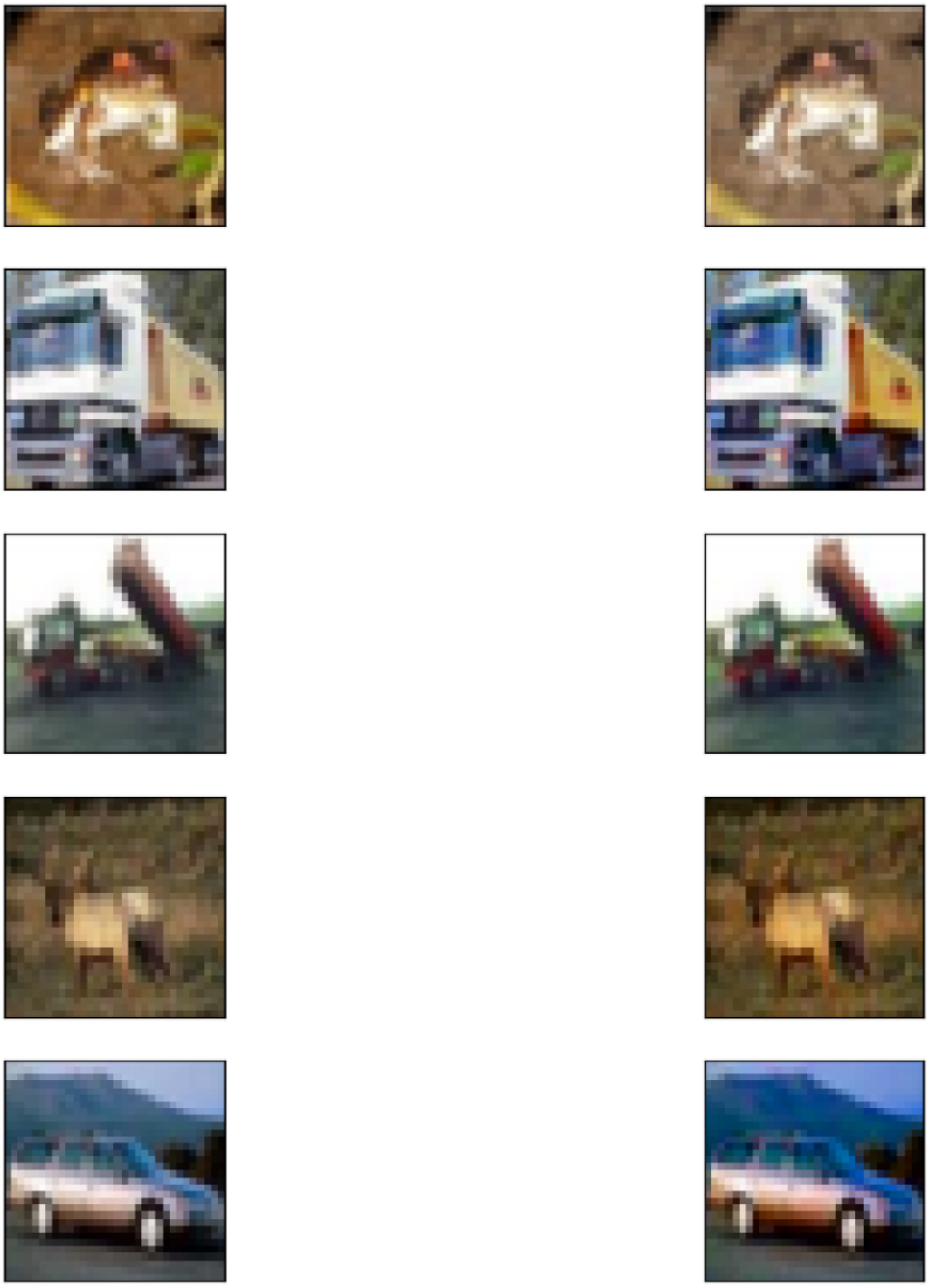
Реализуем наш слой с таким же интерфейсом.

In [21]:
```
from tensorflow.image import stateless_random_saturation

class RandomSaturation(layers.Layer):
    def __init__(self, lower, upper, seed=None, **kwargs):
        super().__init__(**kwargs)
        self.lower = lower
        self.upper = upper
        self.seed = seed or 0

    def call(self, x, training=True):
        """
        Эта функция реализует отображение этим слоем.
        Используем параметр `training`, чтобы отключить этот
        слой во время использования модели
        """
        if training:
            self.seed += 1
            return stateless_random_saturation(
                x,
                self.lower,
                self.upper,
                (self.seed, self.seed))
        else:
            return x

# изменяем насыщенность в пределе 50-200%
my_augmentation = RandomSaturation(0.5, 2)
```

Теперь объект этого класса можно использовать так же, как слои для аугментации данных из реализации `tensorflow`.

Посмотрим на него в действии.

In [22]:
```
show_augmentation(my_augmentation)
```



Посмотрим, как аугментация влияет на процесс обучения.

Обучим ту же модель, что и в прошлом примере, но добавим к ней два варианта аугментации.

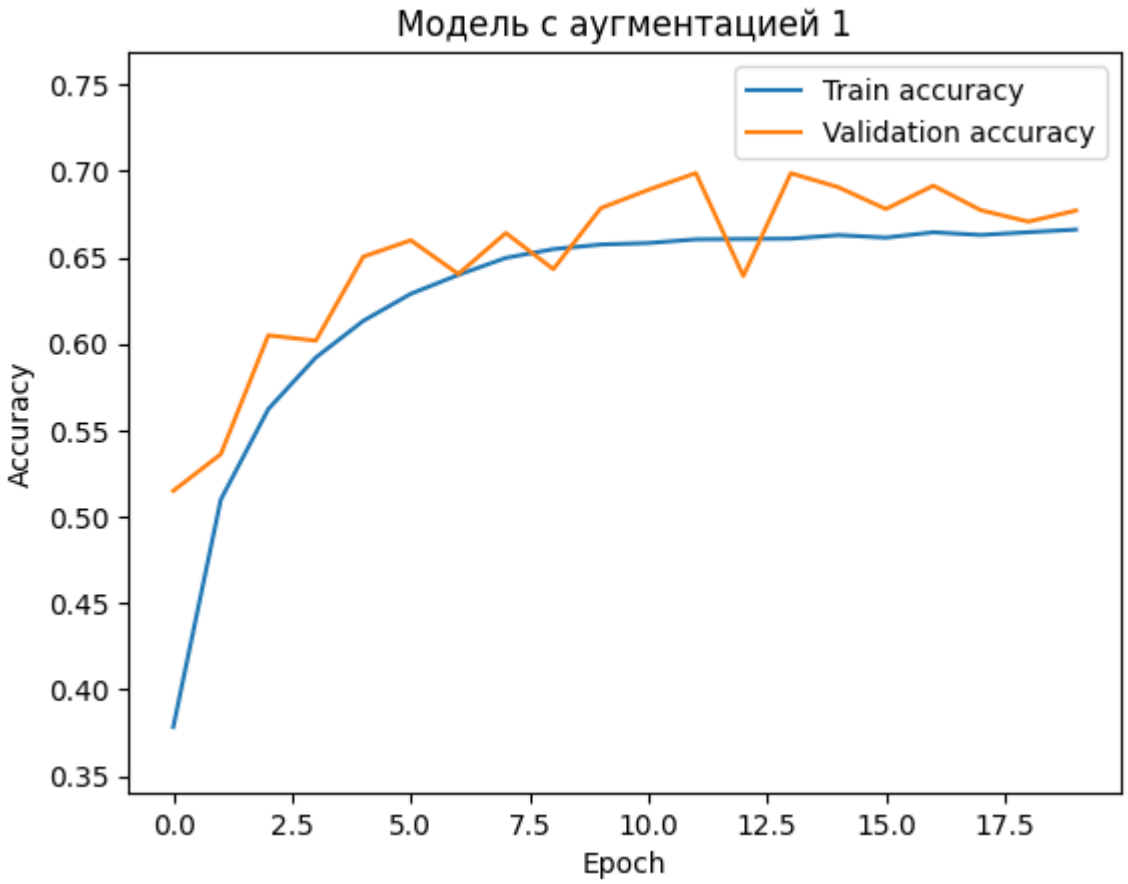
In [33]:
```
model_w_augmentation = tf.keras.Sequential([
    preprocessing,
    augmentation,
```

```
    tf.keras.models.clone_model(cnn_model)],
    name = "model-with-augmentation-1"
)
model_w_augmentation.compile(
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model_w_augmentation2 = tf.keras.Sequential([
    preprocessing,
    my_augmentation,
    tf.keras.models.clone_model(cnn_model)],
    name = "model-with-augmentation-2"
)
model_w_augmentation2.compile(
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
```

Обучим эти модели и посмотрим на динамику.

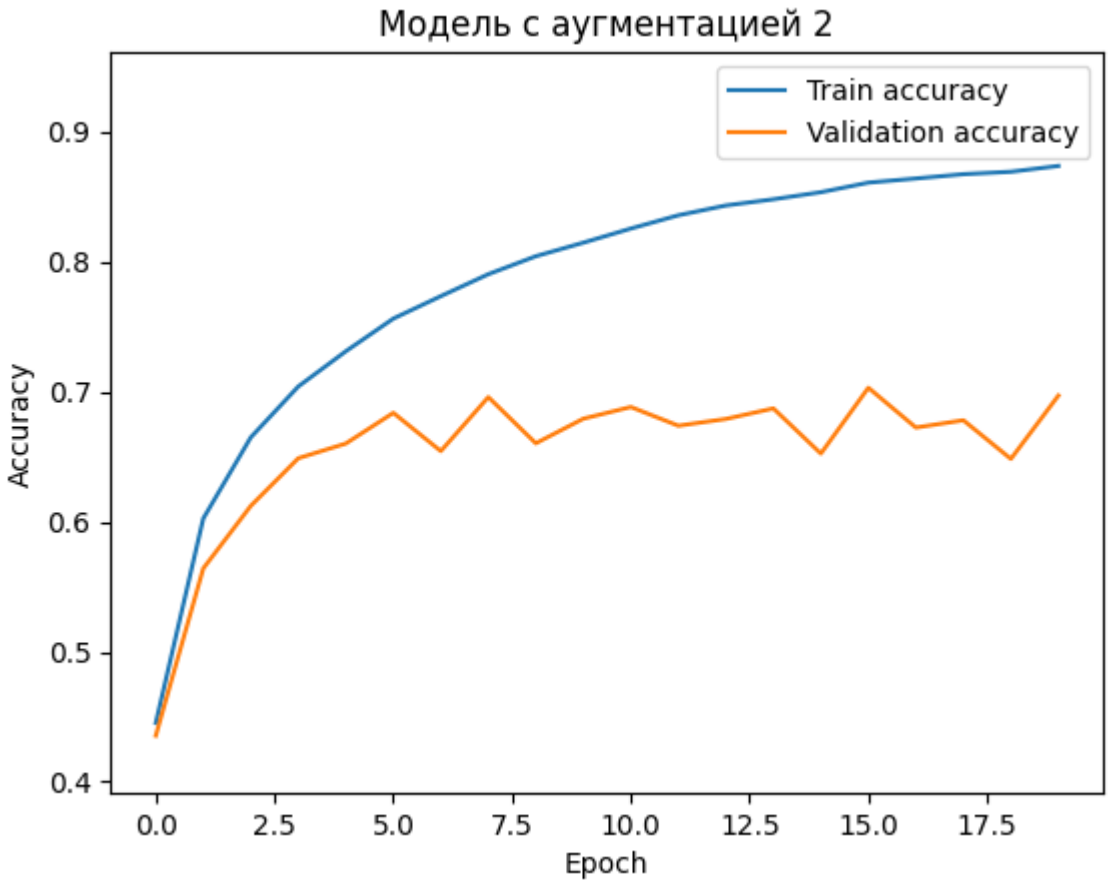
In [ ]:
```
model_aug_history = model_w_augmentation.fit(
    train_images, train_labels, epochs=20,
    validation_data=(test_images, test_labels))
```

In [35]:
```
plot_history(model_aug_history.history,
             "Модель с аугментацией 1", "accuracy")
```



In [ ]:
```
model_aug2_history = model_w_augmentation2.fit(
    train_images, train_labels, epochs=20,
    validation_data=(test_images, test_labels))
```

In [37]:
```
plot_history(model_aug2_history.history,
             "Модель с аугментацией 2", "accuracy")
```



По графикам видно, что первая модель с аугментацией не переобучалась, но результаты получились схожие с моделью, которая обучалась без аугментации.

Второй вариант аугментации не позволил избежать переобучения.

Задания

1. Реализуйте минимум 3 разные нейросети для классификации изображений из примера с другими параметрами: глубина сети, размер сверток, размер полносвязной сети; обучите сеть с аугментацией данных и без, сравните результаты.

2. Реализуйте и добавьте другие варианты аугментации:

   - случайный сдвиг изображения по горизонтали или вертикали;
   - перевод изображения из цветного в черно-белое с вероятностью 50%.

Обучите наиболее удачную модель из задания 1 с этими аугментациями и сравните результаты.

3. Реализуйте сверточную нейронную сеть для классификации изображений из набора данных CIFAR-100. Попробуйте несколько разных нейросетей для классификации. Оцените для них количество параметров, время обучения. Отметьте, было ли переобучение сети.

4. Реализуйте сверточную нейронную сеть для классификации для набора данных 17 Category Flower Dataset. Попробуйте несколько разных нейросетей для классификации. Оцените для них количество параметров, время обучения. Отметьте, было ли переобучение сети.