Практическое задание №1

Установка необходимых пакетов:

```
#!pip install -q tqdm
#!pip install --upgrade --no-cache-dir gdown
```

Монтирование Вашего Google Drive к текущему окружению:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

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

Константы, которые пригодятся в коде далее, и ссылки (gdrive идентификаторы) на предоставляемые наборы данных:

```
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
    'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
    'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}
```

Импорт необходимых зависимостей:

```
from pathlib import Path
import numpy as np
from typing import List
from tqdm.notebook import tqdm
from time import sleep
#from PIL import Image
#import IPython.display
from sklearn.metrics import balanced_accuracy_score, ConfusionMatrixDisplay, preci
import gdown
```

```
# Дополнительнные модули (made by Sokol Stepan)
#import tensorflow. as tf
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

```
from keras.utils import to_categorical
from keras.applications import MobileNetV2
from keras.layers import Dense,Conv2D,MaxPooling2D,Dropout, Flatten, BatchNormaliz
from keras.models import Sequential, load_model
from keras import Model
from keras.optimizers import RMSprop, Adam
from keras.regularizers import L1, L2
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.layers.attention.multi_head_attention import activation
from keras.preprocessing.image import ImageDataGenerator
import pickle
```

Класс Dataset

Предназначен для работы с наборами данных, обеспечивает чтение изображений и соответствующих меток, а также формирование пакетов (батчей).

```
class Dataset:

    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        try:
          url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={DAT
          output = f'{name}.npz'
          gdown.download(url, output, quiet=False)
          print(f'Loading dataset {self.name} from npz.')
        except:
          print("Вредный коллаб запрещает качать данные")
        np_obj = np.load(f'{name}.npz')
        # ВНЕСЕНО ИЗМЕНЕНИЕ В КЛАСС: ПЕРЕВОД ТРЕХКАНАЛЬНЫХ ЦВЕТНЫХ СНИМКОВ
        # В ГРАДАЦИИ СЕРОГО ДЛЯ ЭКОНОМИИ ОПЕРАТИВНОЙ ПАМЯТИ.
        self.images = np.mean(np_obj['data'], axis=3)[...,np.newaxis]
        self.labels = np_obj['labels']
        self.n_files = self.images.shape[0]
        self.is_loaded = True
        print(f'Done. Dataset {name} consists of {self.n_files} images.')

    def image(self, i):
        # read i-th image in dataset and return it as numpy array
        if self.is_loaded:
            return self.images[i, :, :]

    def images_seq(self, n=None):
        # sequential access to images inside dataset (is needed for testing)
        for i in range(self.n_files if not n else n):
            yield self.image(i)

    def random_image_with_label(self):
        # get random image with label from dataset
        i = np.random.randint(self.n_files)
```

```
        return self.image(i), self.labels[i]

    def random_batch_with_labels(self, n):
        # create random batch of images with labels (is needed for training)
        indices = np.random.choice(self.n_files, n)
        imgs = []
        for i in indices:
            img = self.image(i)
            imgs.append(self.image(i))
        logits = np.array([self.labels[i] for i in indices])
        return np.stack(imgs), logits

    def image_with_label(self, i: int):
        # return i-th image with label from dataset
        return self.image(i), self.labels[i]
```

Пример использвания класса Dataset

Загрузим обучающий набор данных, получим произвольное изображение с меткой. После чего визуализируем изображение, выведем метку. В будущем, этот кусок кода можно закомментировать или убрать.

```
#d_train_tiny = Dataset('test_small')

#img, lbl = d_train_tiny.random_image_with_label()
#print()
#print(f'Got numpy array of shape {img.shape}, and label with code {lbl}.')
#print(f'Label code corresponds to {TISSUE_CLASSES[lbl]} class.')

#pil_img = Image.fromarray(img)
#IPython.display.display(pil_img)
#d_train_tiny.images.shape
```

Класс Metrics

Реализует метрики точности, используемые для оценивания модели:

1. точность,
2. сбалансированную точность.

```
class Metrics:

    @staticmethod
    def accuracy(gt: List[int], pred: List[int]):
        assert len(gt) == len(pred), 'gt and prediction should be of equal length'
        return sum(int(i[0] == i[1]) for i in zip(gt, pred)) / len(gt)

    @staticmethod
    def accuracy_balanced(gt: List[int], pred: List[int]):
```

```
        return balanced_accuracy_score(gt, pred)

    @staticmethod
    def print_all(gt: List[int], pred: List[int], info: str):
        print(f'metrics for {info}:')
        print('\t accuracy {:.4f}:'.format(Metrics.accuracy(gt, pred)))
        print('\t balanced accuracy {:.4f}:'.format(Metrics.accuracy_balanced(gt,
        #LBL4 Построение матрицы ошибок, оценивание чувствительности и специфичнос
        print('\t precision {:.4f}:'.format(precision_score(gt, pred, average='wei
        print('\t recall {:.4f}:'.format(recall_score(gt, pred, average='weighted'
        ConfusionMatrixDisplay.from_predictions(gt, pred)
```

Класс Model

Класс, хранящий в себе всю информацию о модели.

Вам необходимо реализовать методы save, load для сохранения и заргрузки модели. Особенно актуально это будет во время тестирования на дополнительных наборах данных.

> *Пожалуйста, убедитесь, что сохранение и загрузка модели работает корректно. Для этого обучите модель, протестируйте, сохраните ее в файл, перезапустите среду выполнения, загрузите обученную модель из файла, вновь протестируйте ее на тестовой выборке и убедитесь в том, что получаемые метрики совпадают с полученными для тестовой выбрки ранее.*

Также, Вы можете реализовать дополнительные функции, такие как:

1. валидацию модели на части обучающей выборки;
2. использование кроссвалидации;
3. автоматическое сохранение модели при обучении;
4. загрузку модели с какой-то конкретной итерации обучения (если используется итеративное обучение);
5. вывод различных показателей в процессе обучения (например, значение функции потерь на каждой эпохе);
6. построение графиков, визуализирующих процесс обучения (например, график зависимости функции потерь от номера эпохи обучения);
7. автоматическое тестирование на тестовом наборе/наборах данных после каждой эпохи обучения (при использовании итеративного обучения);
8. автоматический выбор гиперпараметров модели во время обучения;
9. сохранение и визуализацию результатов тестирования;
10. Использование аугментации и других способов синтетического расширения набора данных (дополнительным плюсом будет обоснование необходимости и обоснование выбора конкретных типов аугментации)

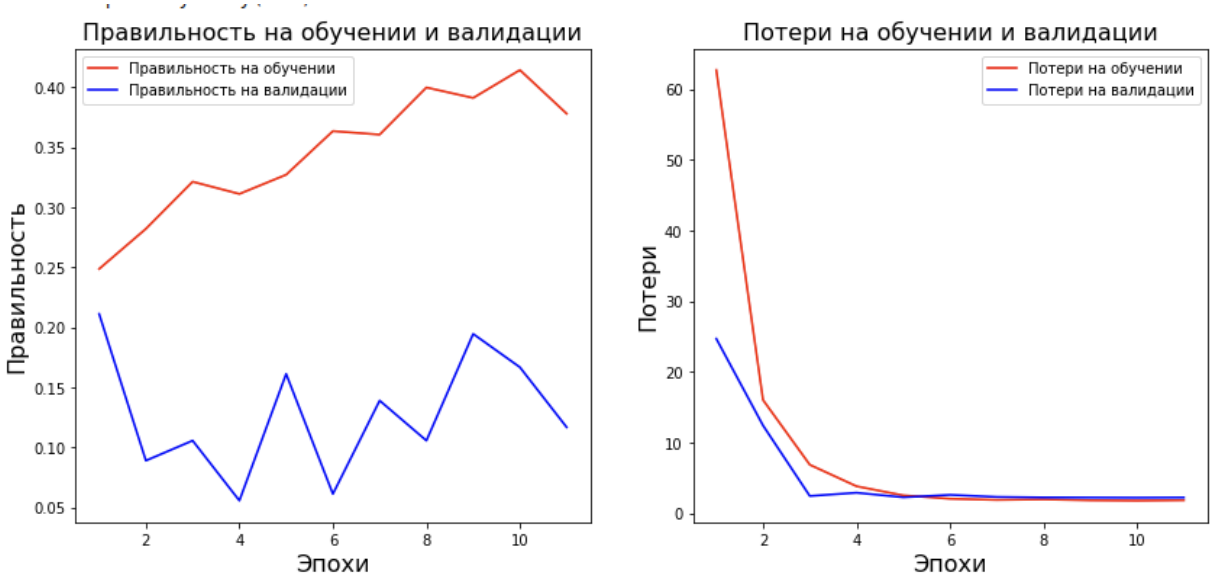
11. и т.д.

Полный список опций и дополнений приведен в презентации с описанием задания.

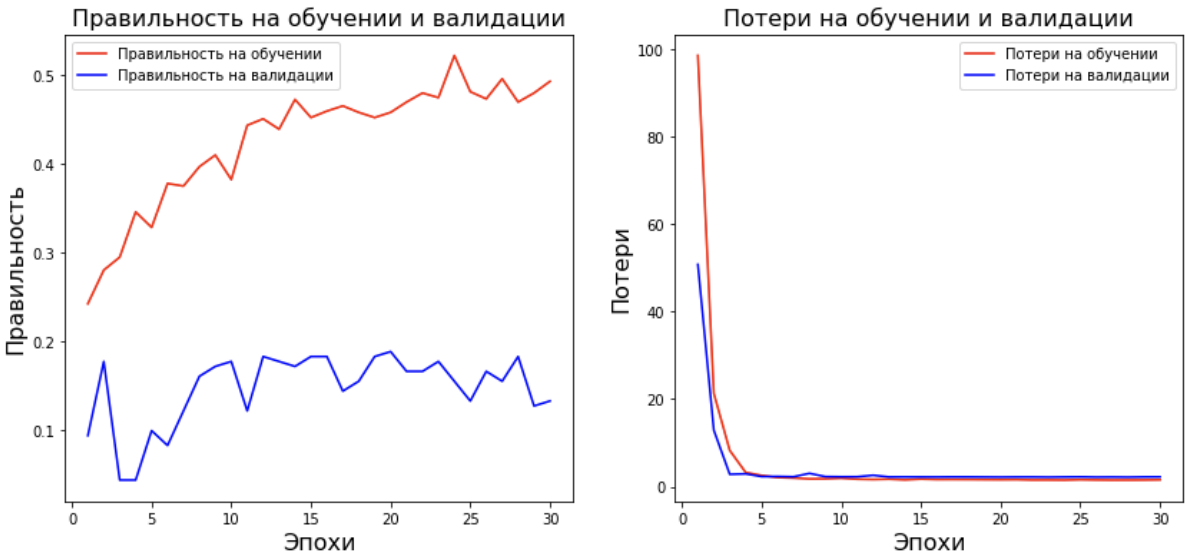
При реализации дополнительных функций допускается добавление параметров в существующие методы и добавление новых методов в класс модели.

Результаты разных сетей

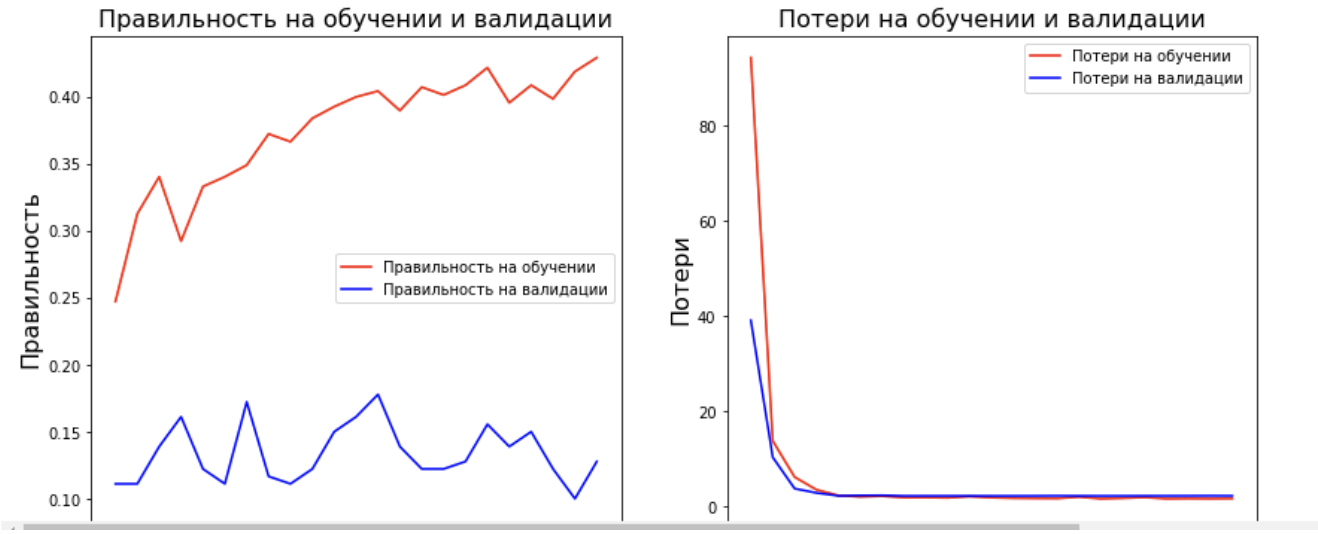
```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#         self.do1 = Dropout(rate=0.5)
#         self.fl  = Flatten()
#         self.dense2 = Dense(9, activation='softmax')
```




```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#         self.do1 = Dropout(rate=0.25)
#         self.fl  = Flatten()
#         self.dense2 = Dense(9, activation='softmax')
```




```
#   def __init__(self):
#       super(Model_, self).__init__()
#       self.conv1 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#       self.do1 = Dropout(rate=0.25)
#       self.fl  = Flatten()
#       self.dense2 = Dense(9, activation='softmax')
```

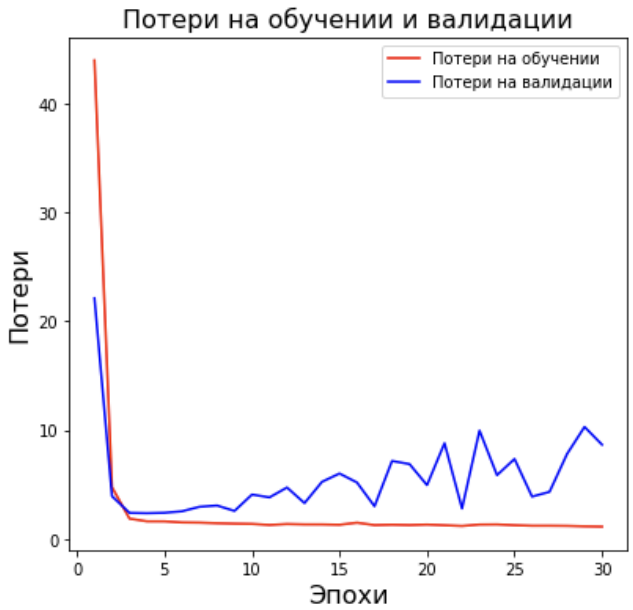



```
#   def __init__(self):
#       super(Model_, self).__init__()
#       self.conv1 = Conv2D(filters=8, kernel_size=3, strides=1, activation="relu
#       self.conv2 = Conv2D(filters=8, kernel_size=3, strides=1, activation="relu
#       self.do1 = Dropout(rate=0.25)
#       self.fl  = Flatten()
#       self.dense2 = Dense(9, activation='softmax')
```

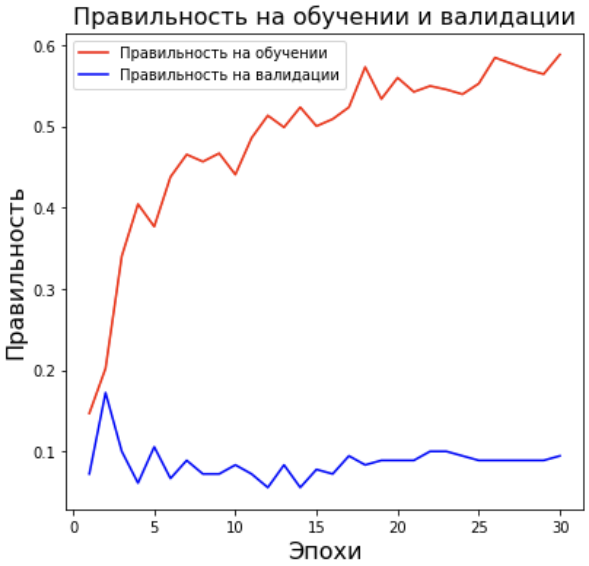
![image.png](

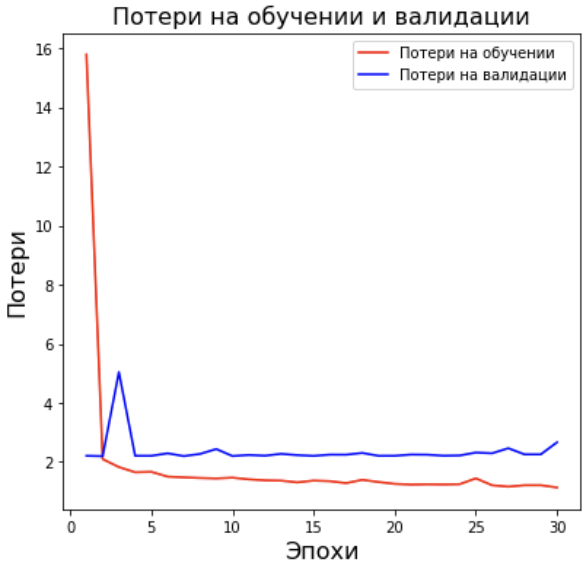
)


```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=8, kernel_size=3, strides=1, activation="relu
#         self.conv2 = Conv2D(filters=16, kernel_size=3, strides=1, activation="rel
#         self.do1 = Dropout(rate=0.25)
#         self.fl  = Flatten()
#         self.dense2 = Dense(9, activation='softmax')
```




```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=16, kernel_size=3, strides=1, activation="rel
#         self.conv2 = Conv2D(filters=16, kernel_size=3, strides=1, activation="rel
#         self.do1 = Dropout(rate=0.25)
#         self.fl  = Flatten()
#         self.dense2 = Dense(9, activation='softmax')
```



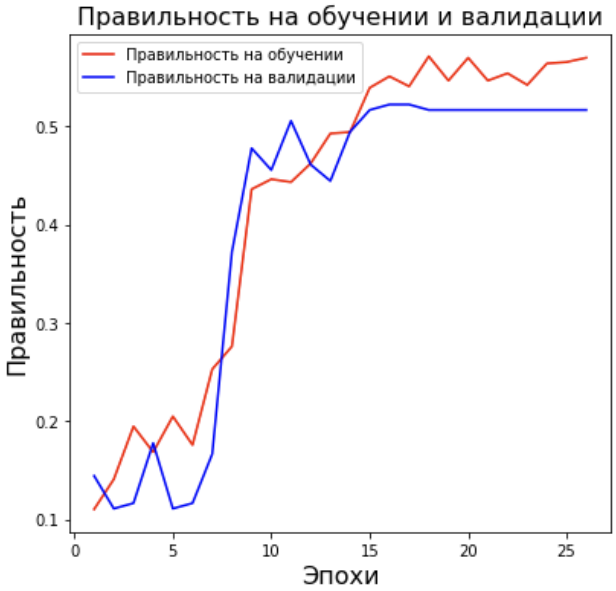


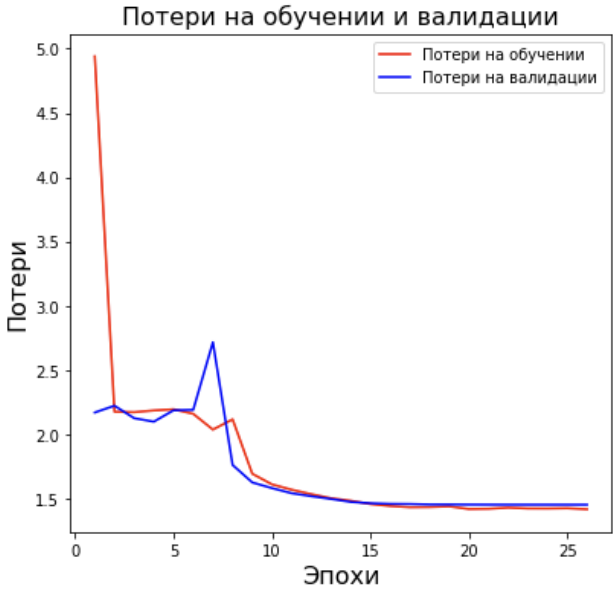

```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=8, kernel_size=3, strides=1, activation="relu
#         self.conv2 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#         self.do1 = Dropout(rate=0.25)
#         self.fl  = Flatten()
#         self.dense2 = Dense(9, activation='softmax')
```


```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#         self.max1 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#         self.max2 =  MaxPooling2D(pool_size=(2, 2))
#         self.fl  = Flatten()
#         self.do1 = Dropout(rate=0.25)
#         self.dense2 = Dense(9, activation='softmax')
```

```
# Внесена Стратификация в train test split, отключены аугментации кроме флипов
```

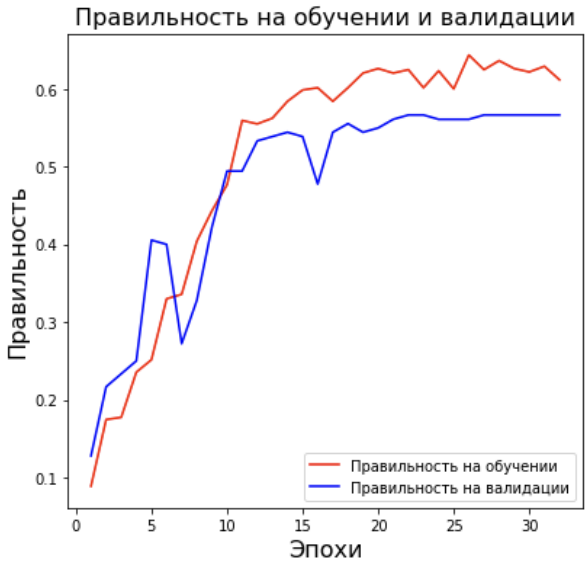

```
#    def __init__(self):
#        super(Model_, self).__init__()
#        self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#        self.max1 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#        self.max2 =  MaxPooling2D(pool_size=(2, 2))
#        self.fl  = Flatten()
#        self.do1 = Dropout(rate=0.25)
#        self.dense2 = Dense(9, activation='softmax')

# та же сеть, что и выше, только с внесенным шедулером по шагу обучения от 0.001 д
```

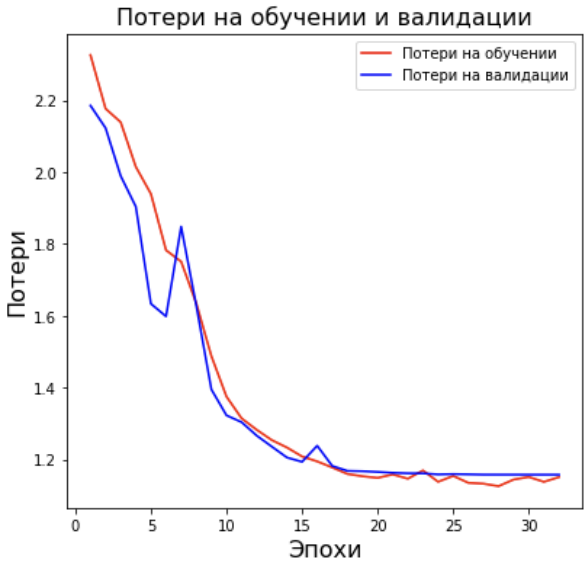





```
#   def __init__(self):
#       super(Model_, self).__init__()
#       self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
```

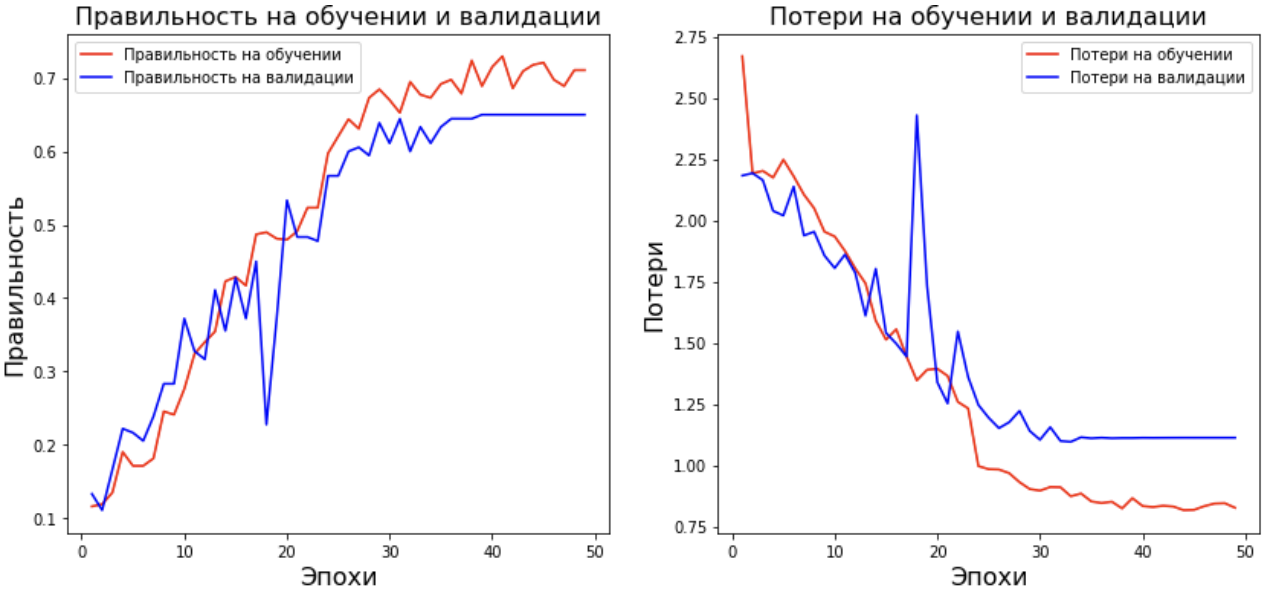
```
#         self.max1 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#         self.max2 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv3 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#         self.max3 =  MaxPooling2D(pool_size=(2, 2))
#         self.fl  = Flatten()
#         self.do1 = Dropout(rate=0.25)
#         self.dense2 = Dense(9, activation='softmax')
```



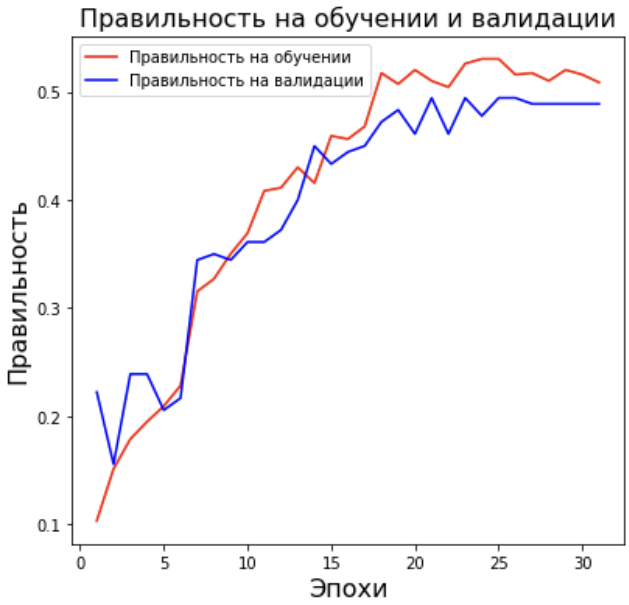


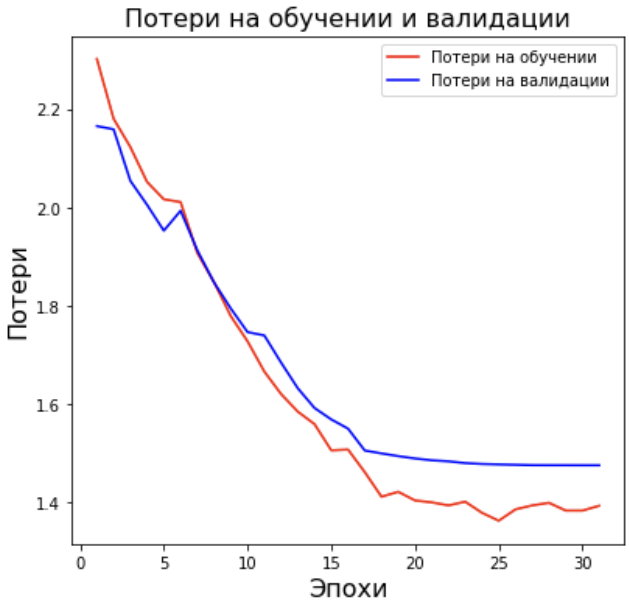

```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#         self.max1 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#         self.max2 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#         self.max3 =  MaxPooling2D(pool_size=(2, 2))
#         self.fl  = Flatten()
#         self.do1 = Dropout(rate=0.25)
#         self.dense2 = Dense(9, activation='softmax')
```

```
#    def __init__(self):
#        super(Model_, self).__init__()
#       self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="relu
#        self.max1 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#        self.max2 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#        self.conv4 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#        self.max3 =  MaxPooling2D(pool_size=(2, 2))
#        self.fl  = Flatten()
#        self.do1 = Dropout(rate=0.25)
#        self.dense2 = Dense(9, activation='softmax')
# Качество 0.65
#Начинается уплотнение, двойные слои свертки
```




```
#    def __init__(self):
#        super(Model_, self).__init__()
#        self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#        self.max1 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#        self.conv3 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#        self.max2 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv4 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#        self.conv5 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#        self.max3 =  MaxPooling2D(pool_size=(2, 2))
#        self.fl  = Flatten()
#        self.do1 = Dropout(rate=0.25)
#        self.dense2 = Dense(9, activation='softmax')
# Стало хуже. 0.48
```

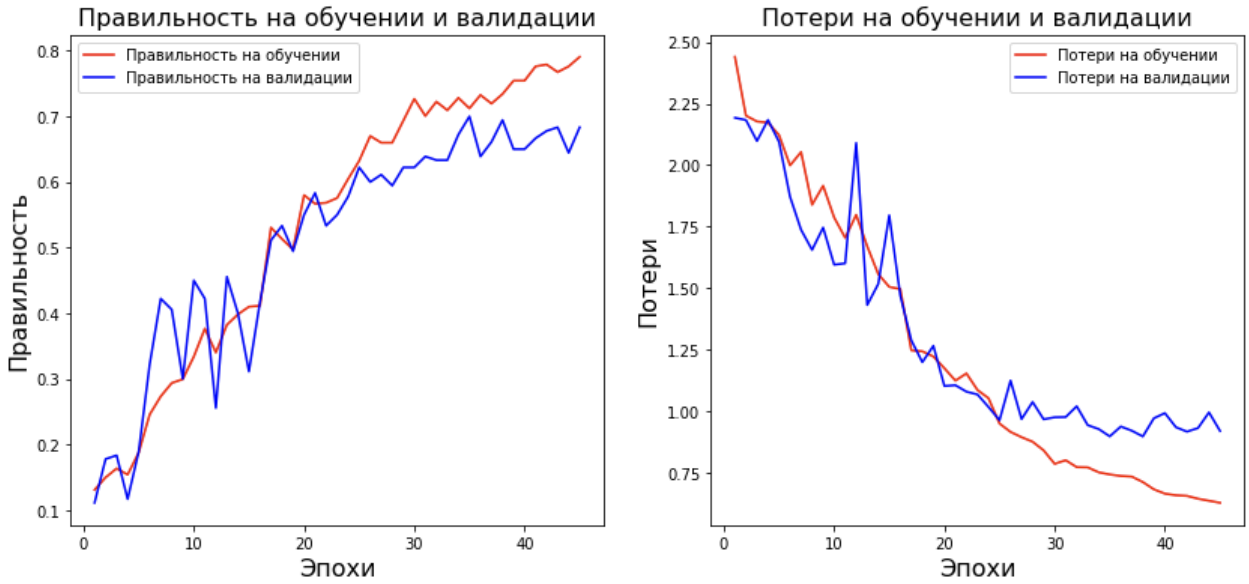




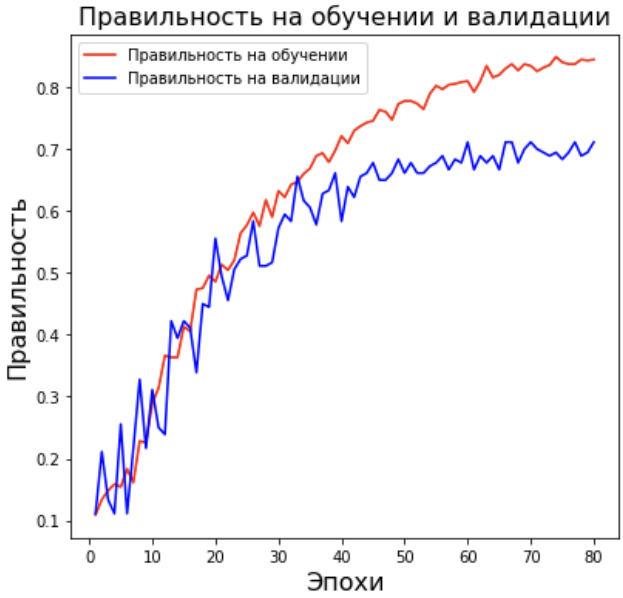

```
#    def __init__(self):
#        super(Model_, self).__init__()
#        self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#        self.max1 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#        self.max2 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#        self.max3 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv4 = Conv2D(filters=256, kernel_size=3, strides=1, activation="re
#        self.max4 =  MaxPooling2D(pool_size=(2, 2))
#        self.fl  = Flatten()
#        self.do1 = Dropout(rate=0.25)
#        self.dense2 = Dense(9, activation='softmax')
# Лучше не стало 0.62
```

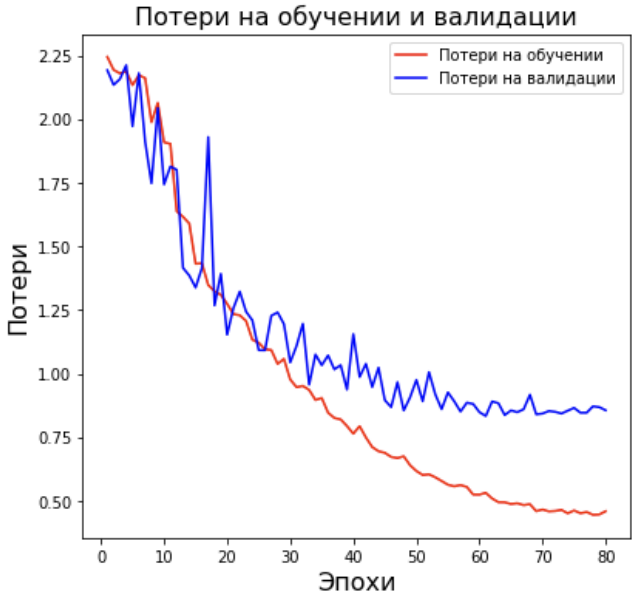

```
#   def __init__(self):
#       super(Model_, self).__init__()
```

```
#       self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#       self.max1 =  MaxPooling2D(pool_size=(2, 2))
#       self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#       self.max2 =  MaxPooling2D(pool_size=(2, 2))
#       self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#       self.max3 =  MaxPooling2D(pool_size=(2, 2))
#       self.conv4 = Conv2D(filters=256, kernel_size=3, strides=1, activation="re
#       self.max4 =  MaxPooling2D(pool_size=(2, 2))
#       self.fl  = Flatten()
#       self.do1 = Dropout(rate=0.25)
#       self.dense1 = Dense(512, activation='relu')
#       self.dense2 = Dense(9, activation='softmax')
# Стало лучше - 0.68. Шедулер подкручен на более плавное снижение шага обучения 0
# Добавлен полносвязный слой
```

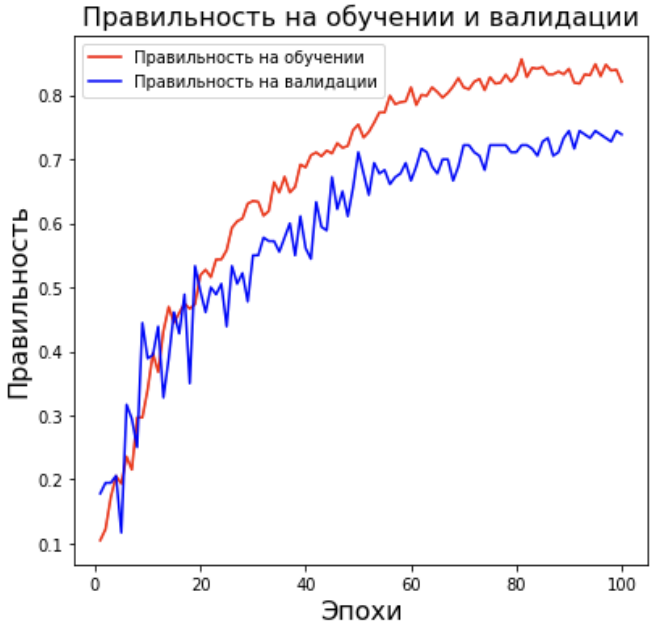


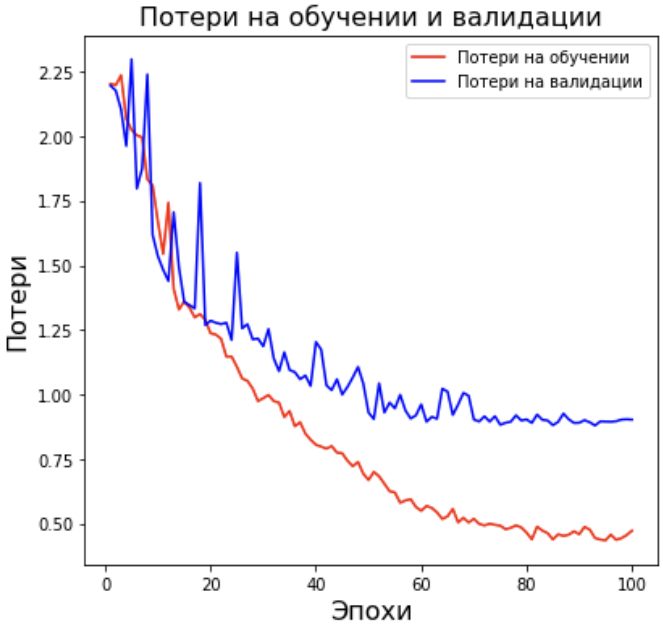

```
#   def __init__(self):
#       super(Model_, self).__init__()
#       self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#       self.max1 =  MaxPooling2D(pool_size=(2, 2))
#       self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#       self.max2 =  MaxPooling2D(pool_size=(2, 2))
#       self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#       self.max3 =  MaxPooling2D(pool_size=(2, 2))
#       self.conv4 = Conv2D(filters=256, kernel_size=3, strides=1, activation="re
#       self.max4 =  MaxPooling2D(pool_size=(2, 2))
#       self.conv4 = Conv2D(filters=512, kernel_size=3, strides=1, activation="re
#       self.max4 =  MaxPooling2D(pool_size=(2, 2))
#       self.fl  = Flatten()
#       self.do1 = Dropout(rate=0.25)
#       self.dense1 = Dense(512, activation='relu')
#       self.dense2 = Dense(9, activation='softmax')

# Добавлен блок свертка-пулинг. Шедулер еще плавнее 0.75, больше эпох,качество ста
```



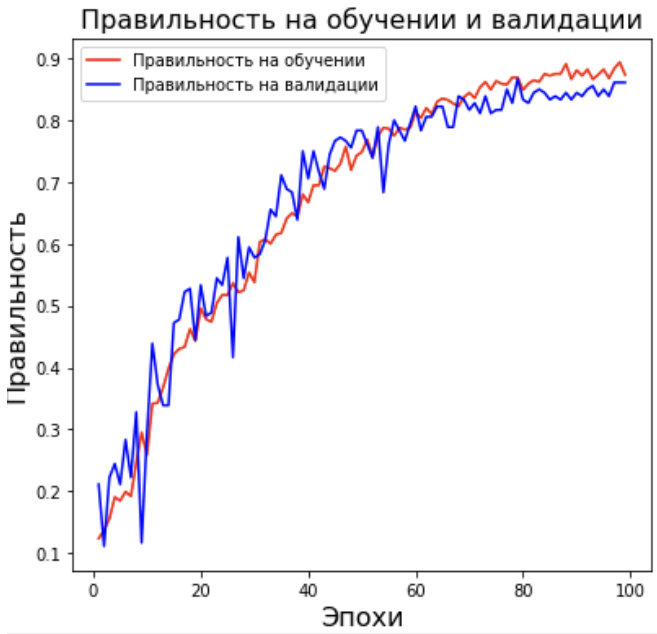


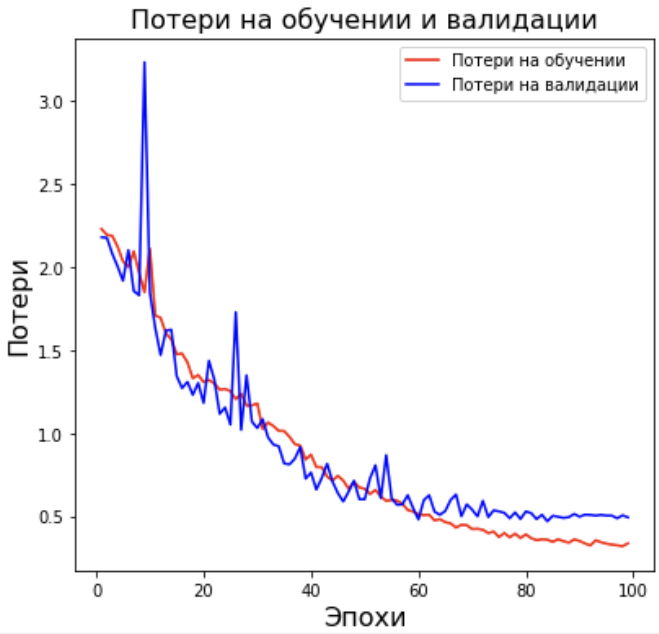

```
#    def __init__(self):
#        super(Model_, self).__init__()
#        self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#        self.max1 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv5 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#        self.max5 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#        self.max2 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#        self.max3 =  MaxPooling2D(pool_size=(2, 2))
#        self.conv4 = Conv2D(filters=512, kernel_size=3, strides=1, activation="re
#        self.max4 =  MaxPooling2D(pool_size=(2, 2))
#        self.fl  = Flatten()
#        self.do1 = Dropout(rate=0.25)
#        self.dense1 = Dense(512, activation='relu')
#        self.dense2 = Dense(9, activation='softmax')
# Продублирован первый блок свертка-пулинг. Стало лучше 0.74
```



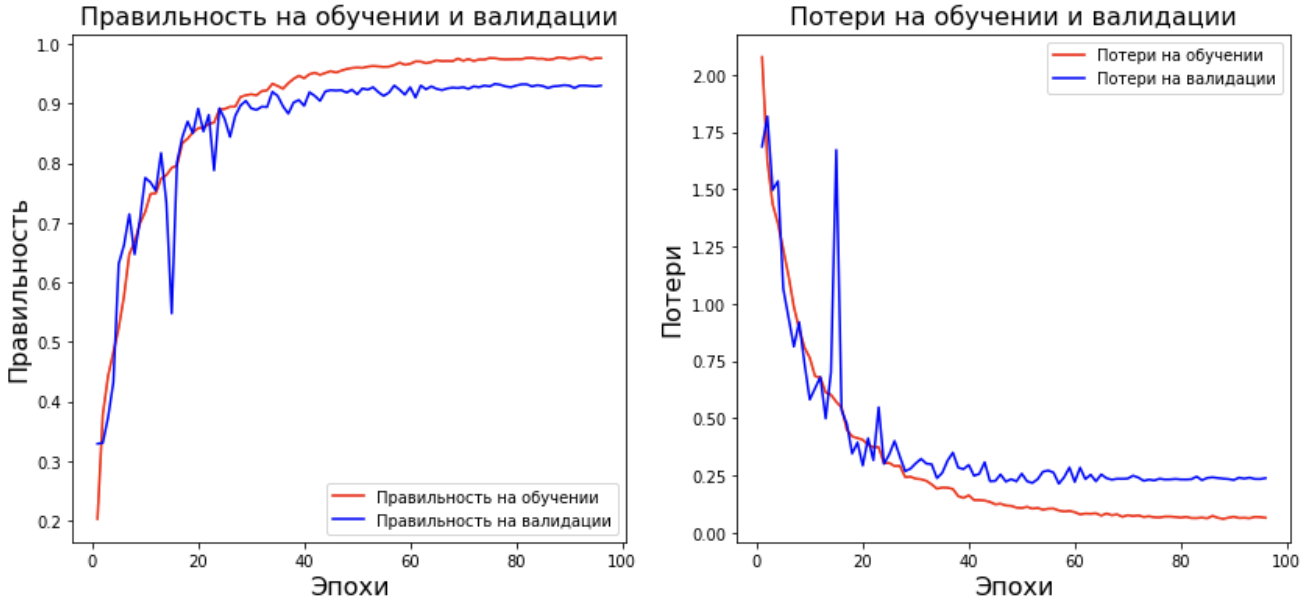


```
#     def __init__(self):
#         super(Model_, self).__init__()
#         self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#         self.max1 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv5 = Conv2D(filters=32, kernel_size=3, strides=1, activation="rel
#         self.max5 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="rel
#         self.max2 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="re
#         self.max3 =  MaxPooling2D(pool_size=(2, 2))
#         self.conv4 = Conv2D(filters=512, kernel_size=3, strides=1, activation="re
#         self.max4 =  MaxPooling2D(pool_size=(2, 2))
#         self.fl  = Flatten()
#         self.do1 = Dropout(rate=0.25)
#         self.dense1 = Dense(512, activation='relu')
#         self.dense2 = Dense(9, activation='softmax')
# Увеличена ротация до 50 градусов. Больше эпох.
# Качество 0.85
```





Код тот же, что и выше, но на больших данных (small) качество под 0.95



Моделирование

```
class Model_(Model): # Попробуем за основу решения взять Tensorflow Keras

    def __init__(self):
        super(Model_, self).__init__()
        self.conv1 = Conv2D(filters=32, kernel_size=3, strides=1, activation="relu
        self.max1  =  MaxPooling2D(pool_size=(2, 2))
        self.bn1   = BatchNormalization()
        self.conv5 = Conv2D(filters=32, kernel_size=3, strides=1, activation="relu
        self.max5  =  MaxPooling2D(pool_size=(2, 2))
        self.bn5   = BatchNormalization()
        self.conv2 = Conv2D(filters=64, kernel_size=3, strides=1, activation="relu
        self.max2  =  MaxPooling2D(pool_size=(2, 2))
        self.bn2   = BatchNormalization()
        self.conv3 = Conv2D(filters=128, kernel_size=3, strides=1, activation="rel
        self.max3  =  MaxPooling2D(pool_size=(2, 2))
        self.bn3   = BatchNormalization()
        self.conv4 = Conv2D(filters=512, kernel_size=3, strides=1, activation="rel
        self.max4 =  MaxPooling2D(pool_size=(2, 2))
        self.fl  = Flatten()
        self.do1 = Dropout(rate=0.25)
        self.dense1 = Dense(1024, activation='relu')
        self.dense2 = Dense(512, activation='relu')
        self.dense3 = Dense(9, activation='softmax')


    def call(self, input_tensor, training=False):
        x = self.conv1(input_tensor)
        x = self.max1(x)
        x = self.bn1(x)
        x = self.conv5(x)
        x = self.max5(x)
        x = self.bn5(x)
```

```
        x = self.conv2(x)
        x = self.max2(x)
        x = self.bn2(x)
        x = self.conv3(x)
        x = self.max3(x)
        x = self.bn3(x)
        x = self.conv4(x)
        x = self.max4(x)
        x = self.fl(x)
        x = self.do1(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.dense3(x)
        return x


    def train(self, dataset: Dataset):
        X_train, X_valid, y_train, y_valid = train_test_split(dataset.images.astyp
                                                              dataset.labels,
                                                              test_size=0.15,
                                                              shuffle=True,
                                                              stratify=dataset.lab
        print(f'training started')
        datagen = ImageDataGenerator(
                rotation_range=50,
                horizontal_flip=True, vertical_flip=True)
        datagen.fit(X_train)

        #LBL5 - адаптивный шаг обучения при достижении плато по правильности
        scheduler = ReduceLROnPlateau(
                                    monitor="val_acc",
                                    factor=0.75,
                                    patience=3,
                                    verbose=0,
                                    mode="auto",
                                    min_delta=0.000001,
                                    cooldown=0,
                                    min_lr=0
                                )




        #LBL1 - реализация раннего прерывания обучения
        callback = EarlyStopping(monitor='val_acc', patience=20, verbose=1)


        self.compile(loss='sparse_categorical_crossentropy',
                    metrics=['acc'],
                    optimizer=RMSprop(
                                      learning_rate=0.001,
                                      rho=0.9,
                                      momentum=0.2,
                                      epsilon=1e-07,
                                      centered=False,
```

```
                                    name="RMSprop",
                                   ))
        #LBL2 - Валидация модели на части обучающей выборки
        history = self.fit(
                           #X_train, y_train,
                           datagen.flow(X_train, y_train, batch_size=32),
                           steps_per_epoch=len(X_train) // 32,
                           epochs=150,
                           callbacks=[callback, scheduler],
                           validation_data=(X_valid, y_valid))
        sleep(2)
        print(f'training done')
        history_dict = history.history
        loss_values = history_dict['loss']
        val_loss_values = history_dict['val_loss']
        accuracy = history_dict['acc']
        val_accuracy = history_dict['val_acc']

        epochs = range(1, len(loss_values) + 1)
        fig, ax = plt.subplots(1, 2, figsize=(14, 6))
        #LBL3 Построение графиков, визуализирующих процесс обучения
        # (график зависимости функции потерь от номера эпохи обучения, и т.п.)
        # График правильности по эпохам
        ax[0].plot(epochs, accuracy, 'r', label='Правильность на обучении')
        ax[0].plot(epochs, val_accuracy, 'b', label='Правильность на валидации')
        ax[0].set_title('Правильность на обучении и валидации', fontsize=16)
        ax[0].set_xlabel('Эпохи', fontsize=16)
        ax[0].set_ylabel('Правильность', fontsize=16)
        ax[0].legend()

        # График потерь по эпохам
        ax[1].plot(epochs, loss_values, 'r', label='Потери на обучении')
        ax[1].plot(epochs, val_loss_values, 'b', label='Потери на валидации')
        ax[1].set_title('Потери на обучении и валидации', fontsize=16)
        ax[1].set_xlabel('Эпохи', fontsize=16)
        ax[1].set_ylabel('Потери', fontsize=16)
        ax[1].legend()


    def test_on_dataset(self, dataset: Dataset):
        X_test = dataset.images.astype('float32')/255.0
        preds = self.predict(X_test)
        preds = np.argmax(preds, axis=1)
        return preds

    def save_(self, name: str):
        with open(f'/content/drive/MyDrive/{name}.pickle', 'wb') as f:
          pickle.dump(self, f)
        print('Model saved')
        #self.save(f'/content/drive/MyDrive/{name}', save_format='tf')
```

Классификация изображений

Используя введенные выше классы можем перейти уже непосредственно к обучению модели классификации изображений. Пример общего пайплайна решения задачи приведен ниже. Вы можете его расширять и улучшать. В данном примере используются наборы данных 'train_small' и 'test_small'.

```
d_train = Dataset('train_small')
d_test = Dataset('test_small')
print(len(d_train.images))
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1qd45xXfDwc
To: /content/train_small.npz
100%|██████████| 841M/841M [00:09<00:00, 88.3MB/s]
Loading dataset train_small from npz.
Done. Dataset train_small consists of 7200 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1wbRsog0n7u
To: /content/test_small.npz
100%|██████████| 211M/211M [00:02<00:00, 97.1MB/s]
Loading dataset test_small from npz.
Done. Dataset test_small consists of 1800 images.
7200
```

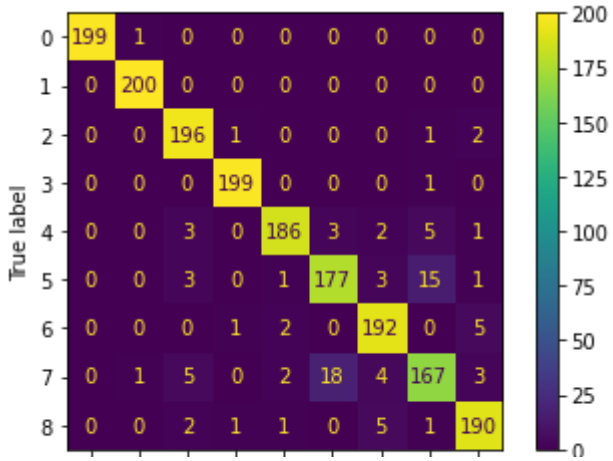
```
model = Model_()
if not EVALUATE_ONLY:
    model.train(d_train)
    model.save_('worst')
else:
    output = 'worst.pickle'
    gdown.download(f'https://drive.google.com/uc?export=download&confirm=pbef&id=1
    with open('worst.pickle', 'rb') as f:
      model = pickle.load(f)
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1-bPLSa9urY
To: /content/worst.pickle
100%|██████████| 172M/172M [00:00<00:00, 220MB/s]
```

Пример тестирования модели на полном наборе данных:

```
# evaluating model on full test dataset (may take time)
pred_2 = model.test_on_dataset(d_test)
Metrics.print_all(d_test.labels, pred_2, 'test')
```

```
WARNING:tensorflow:5 out of the last 67 calls to <function Model.make_predict
57/57 [==============================] - 1s 16ms/step
metrics for test:
        accuracy 0.9478:
        balanced accuracy 0.9478:
        precision 0.9475:
        recall 0.9478:
```



Результат работы пайплайна обучения и тестирования выше тоже будет оцениваться. Поэтому не забудьте присылать на проверку ноутбук с выполненными ячейками кода с демонстрациями метрик обучения, графиками и т.п. В этом пайплайне Вам необходимо продемонстрировать работу всех реализованных дополнений, улучшений и т.п.

Настоятельно рекомендуется после получения пайплайна с полными результатами обучения экспортировать ноутбук в pdf (файл -> печать) и прислать этот pdf вместе с самим ноутбуком.

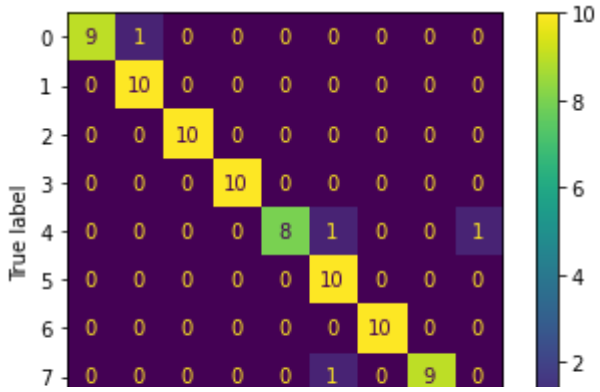
Тестирование модели на других наборах данных

Ваша модель должна поддерживать тестирование на других наборах данных. Для удобства, Вам предоставляется набор данных test_tiny, который представляет собой малую часть (2% изображений) набора test. Ниже приведен фрагмент кода, который будет осуществлять тестирование для оценивания Вашей модели на дополнительных тестовых наборах данных.

Прежде чем отсылать задание на проверку, убедитесь в работоспособности фрагмента кода ниже.

```
output = 'worst.pickle'
gdown.download(f'https://drive.google.com/uc?export=download&confirm=pbef&id=1-bPL
with open('worst.pickle', 'rb') as f:
  final_model = pickle.load(f)
#final_model.load_('worst')
d_test_tiny = Dataset('test_tiny')
preds = final_model.predict(d_test_tiny.images.astype('float32')/255.0)
preds = np.argmax(preds, axis=1)
Metrics.print_all(d_test_tiny.labels, preds, 'test-tiny')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1-bPLSa9urY
To: /content/worst.pickle
100%|██████████| 172M/172M [00:01<00:00, 132MB/s]
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1viiB0s041C
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 20.5MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
3/3 [==============================] - 2s 512ms/step
metrics for test-tiny:
        accuracy 0.9556:
        balanced accuracy 0.9556:
        precision 0.9613:
        recall 0.9556:
```



Отмонтировать Google Drive.

```
drive.flush_and_unmount()
```

Дополнительные "полезности"

Ниже приведены примеры использования различных функций и библиотек, которые могут быть полезны при выполнении данного практического задания.

[ ] ↳ *Скрыто 17 ячеек.*

Платные продукты Colab - Отменить подписку

✓ 0 сек. выполнено в 08:54