СОДЕРЖАНИЕ

1 Постановка задачи

Цель лабораторной работы: реализовать нейронную сеть для классификации изображений.

Классификация изображений представляет собой отнесение изображения к одной из нескольких категорий на основании его содержания.

Будем использовать следующую модель задачи классификации [1].

Ω – множество объектов распознавания (пространство образов).

ω: $\omega \in \Omega$ – объект распознавания (образ).

$g(\omega)$: $\Omega \to M$, $M = \{1, 2, \ldots, m\}$ – индикаторная функция, разбивающая пространство образов Ω на m непересекающихся классов Ω^1, Ω^2, …, Ω^m. Индикаторная функция неизвестна наблюдателю.

X – пространство наблюдений, воспринимаемых наблюдателем (пространство признаков).

$x(\omega)$: $\Omega \to X$ – функция, ставящая в соответствие каждому объекту ω точку $x(\omega)$ пространстве признаков. Вектор $x(\omega)$ – это образ объекта, воспринимаемый наблюдателем. В пространстве признаков определены непересекающиеся множества точек $K_i \in X$, $i = 1, 2, \ldots, m$, соответствующих образам одного класса.

$\hat{g}(\omega)$: $X \to M$ – решающее правило – оценка для $g(\omega)$ на основании $x(\omega)$, т.е. $\hat{g}(x) = \hat{g}(x(\omega))$.

Пусть $x_j = x(\omega_j)$, $j = 1, 2, \ldots, N$ – доступная наблюдателю информация о функциях $g(\omega)$ и $x(\omega)$, но сами этим функции наблюдателю неизвестны. Тогда (g_j, x_j), $j = 1, 2, \ldots, N$ – есть множество прецедентов.

Задача заключается в построении такого решающего правила $\hat{g}(\omega)$, чтобы распознавание проводилось с минимальным числом ошибок.

Обычный случай – считать пространство признаков евклидовым, т.е. $X = R^1$. Качество решающего правила измеряют частотой появления правильных решений. Обычно его оценивают, наделяя множество объектов Ω некоторой вероятностной мерой.

Тогда задача записывается в виде $\min P\{\hat{g}(x(\omega)) \neq g(\omega)\}$.

2 Архитектура нейронной сети

Искусственная нейронная сеть (нейросеть, сеть, ИНС) – это способ собрать нейроны в сеть так, чтобы она решала определённую задачу, например, задачу классификации.

Нейроны собираются по слоям. Есть входной слой, куда подаётся входной сигнал, есть выходной слой, откуда снимается результат работы нейросети, и между ними есть скрытые слои. Если скрытых слоёв больше, чем один, нейросеть считается глубокой [1], если один слой, то сеть неглубокая.

Для классификации изображения собак по породам были построенные две сети с архитектурами DenseNet121 и VGG16. Далее, рассмотрим эти архитектуры.

2.1 DenseNet121

DenseNet – это свёрточная нейронная сеть, в которой каждый слой соединён со всеми другими слоями, которые находятся глубже в сети [2]. Первый слой соединён со вторым, третьим, четвёртым и так далее, второй слой соединён с третьим, четвёртым, пятым и так далее. Такие связи нужны для того, чтобы обеспечить максимальный информационный поток между слоями сети.

В таблице 1 описана архитектура сети DenseNet121.

Таблица 1 – Архитектура DenseNet121 [2]

Слой	Размерность выходных данных	Особенности слоя
1	2	3
Свёртка (convolution)	112 × 112	Размер ядра – 7 × 7, длина шага – 2

Продолжение таблицы 1

1	2	3
Подвыборка (pooling)	56 × 56	Подвыборка максимальных значений (max pooling) с ядром 3 × 3 и шагом – 2
Плотный блок №1 (dense block)	56 × 56	Каждый плотный блок имеет две свёртки с ядрами размером 1 × 1 и 3 × 3. Повторить эти свёртки 6 раз
Переходный №1 (trasition layer)	56 × 56	В переходном слое сокращается количество каналов до половины существующих. Свёртка с ядром 1 × 1
	28 × 28	Подвыборка средних значений с ядром 2 × 2 и шагом длиной 2
Плотный блок №2	28 × 28	Повторить свёртки с ядрами 1 × 1 и 3 × 3 12 раз
Переходный слой №1	28 × 28	Свёртка с ядром 1 × 1
	14 × 14	Подвыборка средних значений с ядром 2 × 2 и шагом длиной 2
Плотный блок №3	14 × 14	Повторить свёртки с ядрами 1 × 1 и 3 × 3 24 раза
Переходный №3	14 × 14	Свёртка с ядром 1 × 1
	7 × 7	Подвыборка средних значение с ядром 2 × 2 и шагом длиной 2
Плотный блок №4	7 × 7	Повторить свёртки с ядрами 1 × 1 и 3 × 3 16 раз
Слой классификации	1 × 1	Подвыборка глобального среднего значения с ядром 7 × 7 и шагом длиной 2
		Функция софтмакс от количества классов

DenseNet начинается с слоя свёртки и подвыборки. Затем следует плотный блок, за которым следует переходный слой, ещё один плотный блок, за которым следует переходный слой, ещё один плотный блок, за которым

следует переходный слой, и, наконец, плотный блок, за которым следует слой классификации.

2.2 VGG16

VGG16 – свёрточная нейронная сеть, особенностью которой являются слои свёртки с фильтром 3 × 3 с шагом 1 [3]. В таблице 2 описана архитектура сети VGG16.

Таблица 2 – Архитектура VGG16 [3]

Слой	Размерность выходных данных	Особенности слоя
1	2	3
Свёртка (convolution)	224 × 224	Размер ядра – 3 × 3, длина шага – 1
Свёртка	224 × 224	
Подвыборка (pooling)	112 × 112	Подвыборка максимальных значений (max pooling) с ядром 2 × 2 и шагом – 2
Свёртка	112 × 112	Размер ядра – 3 × 3, длина шага – 1
Свёртка	112 × 112	
Подвыборка	56 × 56	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Свёртка	56 × 56	Размер ядра – 3 × 3, длина шага – 1
Свёртка	56 × 56	
Свёртка	56 × 56	
Подвыборка	28 × 28	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Свёртка	28 × 28	Размер ядра – 3 × 3, длина шага – 1
Свёртка	28 × 28	
Свёртка	28 × 28	

Продолжение таблицы 2

1	2	3
Подвыборка	14 × 14	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Свёртка	14 × 14	Размер ядра – 3 × 3, длина шага – 1
Свёртка	14 × 14	
Свёртка	14 × 14	
Подвыборка	7 × 7	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Слой классификации	4096	Полносвязанный слой
	4096	Полносвязанный слой
	1	Функция софтмакс от количества классов

На вход слоя свёртки подаётся RGB изображение размером 224 × 224 пикселей. Затем изображение проходят через свёрточные слои, в которых используются фильтры с ядром размера 3x3.

После всех свёрточных слоёв идут три полносвязных слоя: первые два имеют по 4096 каналов, третий – число каналов соответствует числу классов. Последним идёт софтмакс-слой.

Все скрытые слои снабжены функцией активации ReLU.

3 Метод обучения

Для того, чтобы классифицировать изображение нейронная сеть должна выполнить извлечение признаков. В данном случае распознавания изображений такими признаками являются группы пикселей, такие как линии и точки, которые сеть будет анализировать на наличие некоторой закономерности.

Извлечение признаков – это процесс извлечения соответствующих

признаков из входного изображения. Сеть самонастраивается и вырабатывает сама необходимую иерархию абстрактных признаков (последовательности карт признаков), фильтруя маловажные детали и выделяя существенное. Фильтрация происходит в свёрточные слоях, при этом ядра свёртки не закладываются исследователем заранее, а формируются самостоятельно путём обучения сети классическим методом обратного распространения ошибки. Изображения могут содержат аннотации или метаданные, которые помогают сети находить соответствующие признаки.

Операция подвыборки (pooling) выполняет уменьшение размерности сформированных карт признаков. Информация о факте наличия искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный (или средний) и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к размеру входного изображения.

Скалярный результат каждой свёртки попадает на функцию активации, которая представляет собой некую нелинейную функцию. Слой активации обычно логически объединяют со слоем свёртки. Функция нелинейности может быть любой по выбору исследователя, традиционно для этого используются гиперболический тангенс, сигмоиду или ReLU.

После нескольких слоёв свёртки изображения и уплотнения с помощью подвыборки система перестраивается от конкретной сетки пикселей с высоким разрешением к абстрактным картам признаков, как правило на каждом следующем слое увеличивается число каналов и уменьшается размерность изображения в каждом канале. В конце концов остаётся большой набор каналов, хранящих небольшое число данных, которые интерпретируются как самые абстрактные понятия, выявленные из исходного изображения.

Эти данные объединяются и передаются на обычную полносвязную

нейронную сеть, которая тоже может состоять из нескольких слоёв. При этом полносвязные слои уже утрачивают пространственную структуру пикселей и обладают сравнительно небольшой размерностью (по отношению к количеству пикселей исходного изображения). Полученный вектор передаётся функции активации в качестве параметра, а вектор значений функции активации описывает вероятности принадлежности изображения ко всем классам (размерность вектора соответствует числу классов).

Для решения поставленной задачи была построена свёрточная нейронная сеть, которая была обучена на 20580 изображений собак, каждому изображению соответствовала одна порода собак, всего – 120 пород. Сеть была обученная для того, чтобы предсказывать по изображению собаки её породу.

Для обучения использовался набор данных Stanford Dogs Dataset [2]. Набор данных представляет собой цветные изображения в формате JPEG и файлы в формате XML с описание каждого изображения. В рамках текущей задачи, в файлах с описаниями важны метки, которые описывают координаты расположения собаки на изображении (x_{min}, x_{max}, y_{min}, y_{max}), название изображения (filename) и название породы собаки (name).

Особенности метода обучения:

– архитектура сети – DenseNet121;

– функция активации – ReLU;

– оптимизатор градиентного спуска – Adam;

– функция потерь градиентного спуска – категориальная перекрёстная энтропия;

– метрика – точность;

– количество эпох – 20.

Коэффициент скорости обучение уменьшается в 5 раз (во время градиентного спуска), если значение функции потери не уменьшилось за одну последующую эпоху. Уменьшение коэффициента не происходит, пока значение функции потери не уменьшилось как минимум на 0,0001.

Существует минимальное значение коэффициента, при котором он больше не может быть уменьшен – 10^{-7}.

Обучение может закончится раньше, чем за 20 эпох, если значение категориальной перекрёстной энтропии не изменилось за 5 эпох.

4 Вычислительные эксперименты

Вычислительные эксперименты заключались в обучении двух нейросетей с разными архитектурами – DenseNet121 и VGG16. Сети обучались на одних и тех же данных – обработанных изображениях собак, валидировались – на других. Целью обучения было получить сеть, которая предсказывает породу собаки по изображению, всего 120 классов (120 пород). Метод обучения сетей описан в пункте 3.

В таблице 3 представлена история обучения на наборе данных Stanford Dogs Dataset сети с архитектурой DenseNet121

Таблица 3 – История обучения DenseNet121

Эпоха	Обучение		Валидация		Коэффициент скорости обучение
	Функция потерь	Точность	Функция потерь	Точность	
1	2	3	4	5	6
1	3,26	0,26	0,73	0,77	1e-3
2	1,16	0,65	0,59	0,80	
3	0,94	0,71	0,53	0,83	
4	0,85	0,74	0,53	0,82	
5	0,66	0,79	0,45	0,85	2e-4
6	0,60	0,81	0,44	0,86	
7	0,52	0,83	0,44	0,86	
8	0,52	0,83	0,43	0,86	4e-5
9	0,50	0,84	0,43	0,86	
10	0,46	0,85	0,42	0,86	8e-6
11	0,49	0,84	0,42	0,86	
12	0,47	0,85	0,42	0,86	
13	0,47	0,85	0,42	0,86	
14	0,48	0,85	0,42	0,86	1,6e-6
15	0,46	0,85	0,42	0,86	

Продолжение таблицы 3

1	2	3	4	5	6
16	0,47	0,85	0,42	0,86	3,2e-7
17	0,47	0,84	0,42	0,86	
Общее время обучения	47 минут 48 секунд				
Максимальная точность на валидационном наборе	0,86				

На рисунке 1 представлен графики зависимости точности от эпохи для двух наборов данных – набор для обучения и набор для валидации. Как можем убедится на графиках, точность не увеличивается, начиная с девятой эпохи.

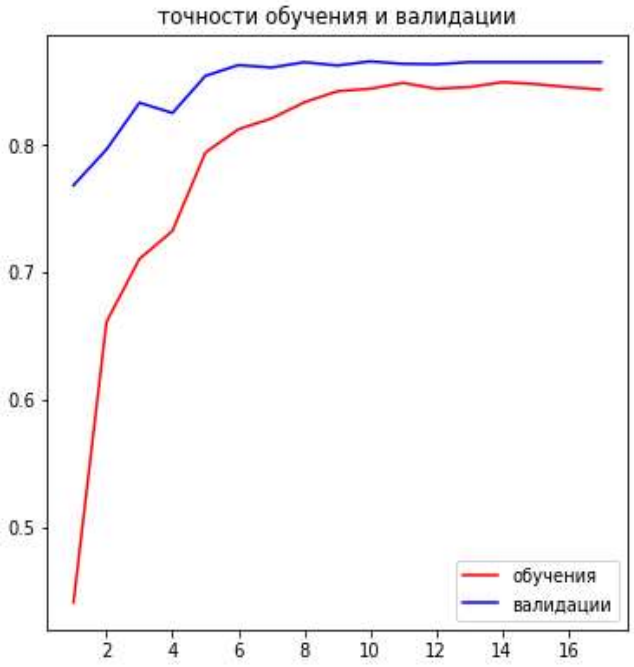

Рисунок 1 – Графики зависимости точности от эпохи для сети DenseNet121

На рисунке 2 представлены графики зависимости функции потери от эпохи для двух наборов данных – набор для обучения и набор для валидации. Отметим, что сеть не может уменьшить потери, начиная с девятой эпохи.

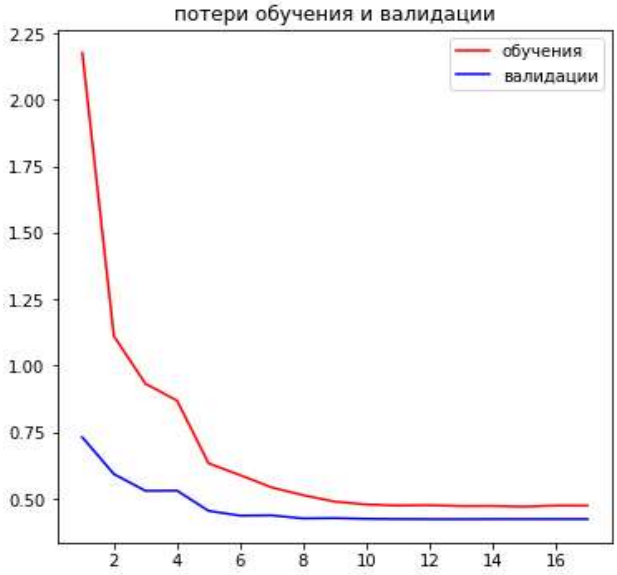

Рисунок 2 – Графики зависимости функции потери от эпохи для сети DenseNet121

В таблице 4 представлена история обучения на наборе данных Stanford Dogs Dataset сети с архитектурой VGG16.

Таблица 4 – История обучения VGG16

Эпоха	Обучение		Валидация		Коэффициент скорости обучение
	Функция потерь	Точность	Функция потерь	Точность	
1	2	3	4	5	6
1	4,56	0,30	1,49	0,59	1e-3
2	2,17	0,48	1,29	0,64	
3	2,00	0,51	1,30	0,67	
4	1,47	0,61	0,89	0,76	2e-4
5	1,11	0,68	0,86	0,77	
6	0,98	0,71	0,81	0,78	
7	0,92	0,72	0,80	0,77	
8	0,85	0,74	0,77	0,78	
9	0,79	0,76	0,75	0,79	
10	0,74	0,77	0,74	0,78	
11	0,70	0,79	0,74	0,78	
12	0,61	0,80	0,71	0,79	4e-5
13	0,57	0,82	0,70	0,79	
14	0,55	0,82	0,70	0,80	

Продолжение таблицы 4

1	2	3	4	5	6
15	0,52	0,83	0,69	0,80	4e-5
16	0,49	0,85	0,68	0,80	
17	0,50	0,84	0,68	0,80	
18	0,47	0,84	0,68	0,80	8e-6
19	0,46	0,85	0,67	0,80	
20	0,46	0,85	0,67	0,80	
Общее время обучения	58 минут 51 секунд				
Максимальная точность на валидационном наборе	0,80				

На рисунке 3 представлены графики зависимости точности от эпохи для двух наборов данных – набор для обучения и набор для валидации.

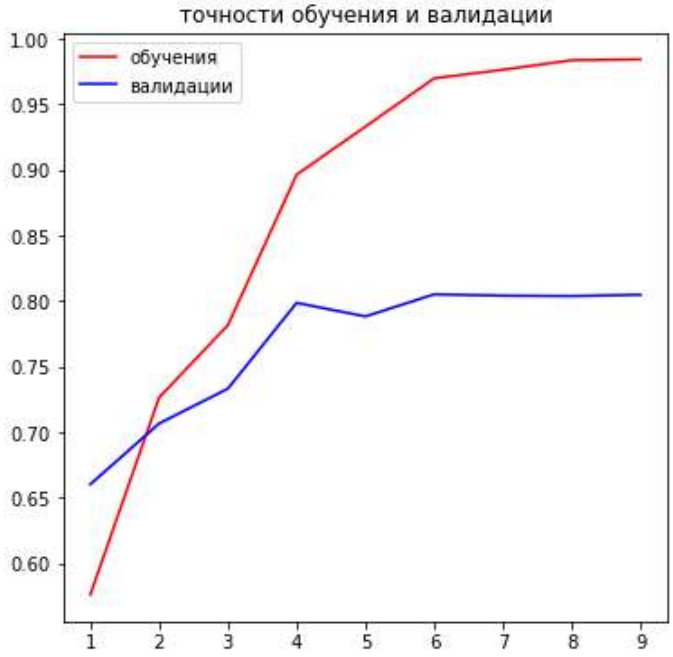

Рисунок 3 – Графики зависимости точности от эпохи для сети VGG16

На рисунке 4 представлен графики зависимости функции потери от эпохи для двух наборов данных – набор для обучения и набор для валидации.

Рассмотрев оба графика, приходим к следующему выводу: точность на валидационном наборе не увеличивается, начиная с четвёртой эпохи модель

начинает переобучиваться на тренировочных данных.

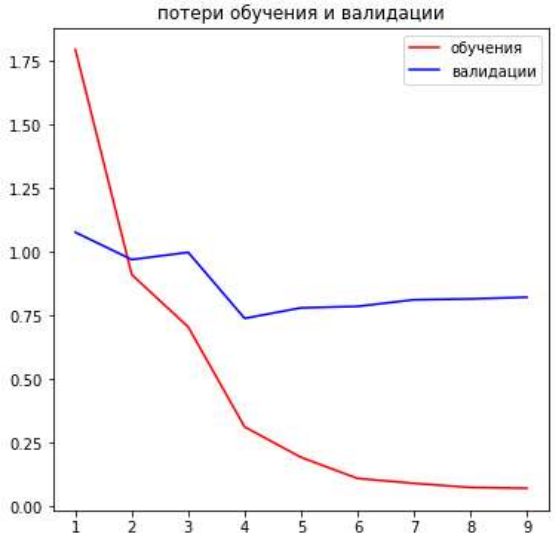

Рисунок 4 – Графики зависимости функции потери от эпохи для сети VGG16

5 Результаты работы

Применим обученные сети на практике. Для этого передадим каждой нейросети четыре изображения собак со следующими породами: шпиц (pomeranian), доберман (doberman), мопс (pug) и чау (chow). Эти изображения не принимали участия в обучении и валидации.

На рисунке 5 изображены результаты предсказания пород собак сети с архитектурой DenseNet121.

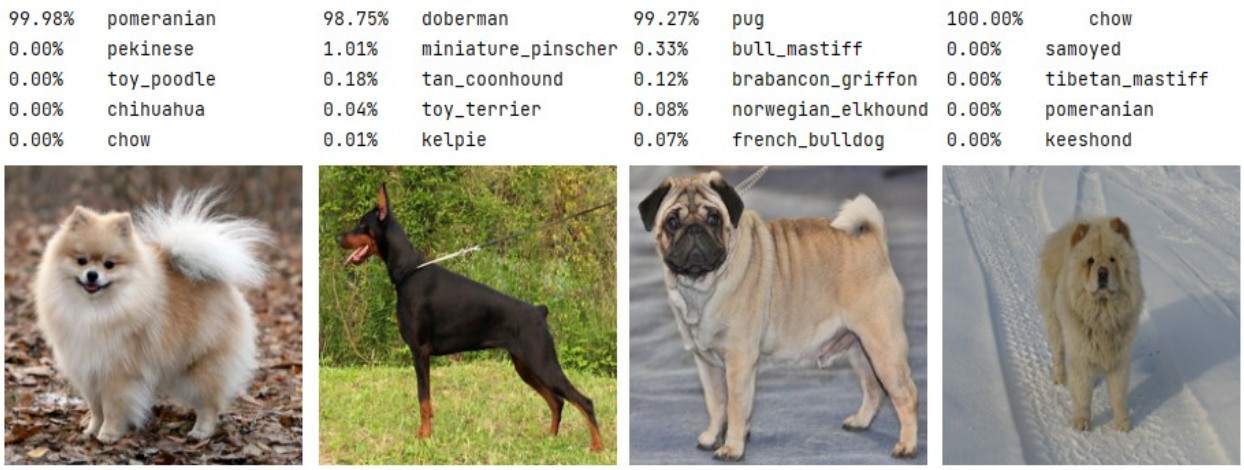

Рисунок 5 – Предсказания DenseNet121

На рисунке 6 изображены результаты предсказания пород собак сети

с архитектурой VGG16.

Как видим обе нами обученные сети точно предсказывают породу собаки по её изображению.

Рисунок 6 – Предсказания VGG16

Стоит отметить, что на практике лучше использовать модель DenseNet121, поскольку размер обученной модели – 48 Мб – это в пять раз меньше размера модели VGG16 – 285 Мб.

Исходный код лабораторной работы представлен в приложении А и доступен в публичном репозитории – https://github.com/algorithm-ssau/generating-image-captions/tree/main/lab1-dog-breed-classifier.

6 Вывод

В настоящей лабораторной работе были реализованы при помощи фреймворка Keras две нейронные сети с разными архитектурами для классификации изображений собак. Также, были изучены: метод обучения, параметры обучения, параметры нейронной сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Schmidhuber, J. Deep Learning in Neural Networks: An Overview // Neural Networks. 61: 85–117. — 2015. — doi:10.1016/j.neunet.2014.09.003. — arXiv:1404.7828. — PMID 25462637.

2 Densely Connected Convolutional Networks [Электронный ресурс]. URL: https://arxiv.org/abs/1608.06993 (дата обращения: 30.03.2021).

3 Very Deep Convolutional Networks for Large-Scale Image Recognition [Электронный ресурс]. URL: https://arxiv.org/abs/1409.1556 (дата обращения: 30.03.2021).

Приложение А

Исходный код

Файл lab1.ipynb:

```
import warnings
warnings.filterwarnings("ignore")

import tensorflow as tf
from keras.models import *
from keras.layers import *

from keras.utils import *
from keras.callbacks import *
from keras.applications.densenet import DenseNet121, preprocess_input

import numpy as np
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import os
import xml.etree.ElementTree as ET

from skimage.io import imread
from PIL import Image


dataset_folder_path = "D:/datasets/stanford-dogs-dataset"
img_folders_path = dataset_folder_path + "/images/Images"
cropped_path = dataset_folder_path + "/cropped"
annots_path = dataset_folder_path + "/annotations/Annotation"

breed_dirs_list = os.listdir(img_folders_path)

num_classes = len(breed_dirs_list)
print("{} пород".format(num_classes))

num_total_images = 0

for breed_dir in breed_dirs_list:
    num_total_images += len(os.listdir(img_folders_path +
"/{}".format(breed_dir)))

print("{} изображений всего".format(num_total_images))

dir_idx_map = {}
idx_dir_map = {}

for i, v in enumerate(breed_dirs_list):
    dir_idx_map.update({v : i})
    idx_dir_map.update({i : v})


def print_imgs(dir_name, num_to_show):
    plt.figure(figsize=(16,16))

    dir_with_imgs = (img_folders_path + "/{}").format(dir_name)
    imgs = os.listdir(dir_with_imgs)[:num_to_show]

    rows_num = num_to_show/4+1
```

```
    cols_num = 4
    for i in range(num_to_show):
        img = mpimg.imread(dir_with_imgs + "/" + imgs[i])

        position = i+1
        plt.subplot(rows_num, cols_num, position)
        plt.imshow(img)
        plt.axis("off")

idx=119
print(breed_dirs_list[idx])
print_imgs(breed_dirs_list[idx], 20)


os.mkdir(cropped_path)

for breed_dir in breed_dirs_list:
    os.mkdir(cropped_path + "/" + breed_dir)

cropped_dirs_list = os.listdir(cropped_path)
print("создано {} папок для хранения обрезанных изображений собак по
породам".format(len(cropped_dirs_list)))


%%time

for breed_name in cropped_dirs_list:
    for file in os.listdir(annots_path + "/{}".format(breed_name)):
        img = Image.open(img_folders_path + "/{}/{}.jpg".format(breed_name,
file))

        tree = ET.parse(annots_path + "/{}/{}".format(breed_name, file))
        xmin =
int(tree.getroot().findall("object")[0].find("bndbox").find("xmin").text)
        xmax =
int(tree.getroot().findall("object")[0].find("bndbox").find("xmax").text)
        ymin =
int(tree.getroot().findall("object")[0].find("bndbox").find("ymin").text)
        ymax =
int(tree.getroot().findall("object")[0].find("bndbox").find("ymax").text)

        img = img.crop((xmin, ymin, xmax, ymax))
        img = img.convert("RGB")
        img = img.resize((224, 224))
        img.save(cropped_path + "/" + breed_name + "/" + file + ".jpg")


def paths_labels_targets():
    paths = list()
    labels = list()
    targets = list()

    for breed_name in breed_dirs_list:
        curr_breed_path = cropped_path + "/{}".format(breed_name)

        for img_name in os.listdir(curr_breed_path):
            paths.append(curr_breed_path + "/" + img_name)
            labels.append(breed_name)
            targets.append(dir_idx_map[breed_name])

    return paths, labels, targets

paths, labels, targets = paths_labels_targets()

assert len(paths) == len(labels)
```

```
assert len(paths) == len(targets)

targets = to_categorical(targets, num_classes=num_classes)
targets.shape


class ImageGenerator(Sequence):

    def __init__(self, paths, targets, batch_size, shape):
        self.paths = paths
        self.targets = targets
        self.batch_size = batch_size
        self.shape = shape

    def __getitem__(self, batch_idx):
        start_path_idx = batch_idx * self.batch_size
        end_path_idx = (batch_idx + 1) * self.batch_size
        batch_paths = self.paths[start_path_idx : end_path_idx]

        X = np.zeros((len(batch_paths),
                      self.shape[0], self.shape[1], self.shape[2]),
                     dtype=np.float32)

        for i, path in enumerate(batch_paths):
            img = imread(path)
            img = preprocess_input(img)
            X[i] = img

        y = self.targets[start_path_idx : end_path_idx]

        return X, y

    def __iter__(self):
        for item in (self[i] for i in range(len(self))):
            yield item

    def __len__(self):
        return int(np.ceil(len(self.paths) / float(self.batch_size)))


train_paths, valid_paths, train_targets, valid_targets =
train test split(paths,

targets,

test_size=0.15,

random_state=42)

train_gen = ImageGenerator(train_paths, train_targets, batch_size=32,
shape=(224,224,3))
valid_gen = ImageGenerator(valid_paths, valid_targets, batch_size=32,
shape=(224,224,3))


img_input = Input((224, 224, 3))

densenet_model = DenseNet121(input_tensor=img_input,
                        weights="imagenet",
                        include_top=False)

x = densenet_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.5)(x)
```

```
x = Dense(512, activation="relu")(x)
x = Dropout(0.5)(x)
classes_output = Dense(num_classes, activation="softmax")(x)

model = Model(img_input, classes_output)

for layer in model.layers[:-6]:
    layer.trainable = False

model.summary()


model.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["acc"])

checkpoint = ModelCheckpoint("dog-breed-classifier-densenet121.h5",
                             monitor="val_acc",
                             verbose=1,
                             save_best_only=True,
                             mode="max")

reduce_lr = ReduceLROnPlateau(monitor="val_loss",
                              factor=0.2,
                              patience=1,
                              verbose=1,
                              mode="min",
                              min_delta=0.0001,
                              cooldown=2,
                              min_lr=1e-7)

early_stop = EarlyStopping(monitor="val_loss",
                           mode="min",
                           patience=5)


%%time

history = model.fit_generator(generator=train_gen,
                              steps_per_epoch=len(train_gen),
                              validation_data=valid_gen,
                              validation_steps=len(valid_gen),
                              epochs=20,
                              callbacks=[checkpoint, reduce_lr, early_stop])


plt.rcParams["figure.figsize"] = (6,6)

plt.title("точности обучения и валидации")
acc = history.history["acc"]
epochs = range(1, len(acc) + 1)
val_acc = history.history["val_acc"]
plt.plot(epochs, acc, "red", label='обучения')
plt.plot(epochs, val_acc, "blue", label="валидации")
plt.legend()

plt.figure()
plt.title("потери обучения и валидации")
loss = history.history["loss"]
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "red", label="обучения")
plt.plot(epochs, val_loss, "blue", label="валидации")
plt.legend()

plt.show()
```

```
import json

def read_json(file_name: str):
    with open(file_name) as file_in:
        return json.load(file_in)

def predict(file_name):
    img = Image.open(file_name)
    img = img.convert("RGB")
    img = img.resize((224, 224))
    img.save(file_name)

    plt.figure(figsize=(4, 4))
    plt.imshow(img)
    plt.axis("off")

    img = imread(file_name)
    img = tf.keras.applications.densenet.preprocess_input(img)
    img = tf.expand_dims(img, axis=0)
    probs = model.predict(img)

    for idx in probs.argsort()[0][::-1][:5]:
        print("{:.2f}%".format(probs[0][idx]*100), "\t",
breed_names_map[str(idx)])

def download and predict(url, file name):
    os.system("curl -s {} -o {}".format(url, file_name))
    predict(file_name)

breed_names_path = "D:/YandexDisk/models/dog-breed-classifier.json"
breed_names_map = read_json(breed_names_path)


download_and_predict("https://i.imgur.com/QzxTOG1.jpg",
                     "pomeranian.jpg")


from keras.applications.vgg16 import VGG16, preprocess_input

img_input = Input((224, 224, 3))

vgg16_model = VGG16(input_tensor=img_input,
                    weights = 'imagenet',
                    include_top=False)

for layer in vgg16_model.layers:
    layer.trainable = False

for layer in vgg16 model.layers:
    print(layer,layer.trainable)

model = Sequential()
model.add(vgg16_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(4096, activation="relu"))
model.add(Dense(4096, activation="relu"))
model.add(Dense(num_classes,activation='softmax'))

model.summary()
```

```
%%time

history = model.fit(train_gen,
                    steps_per_epoch=len(train_gen),
                    validation_data=valid_gen,
                    validation_steps=len(valid_gen),
                    epochs=20,
                    callbacks=[checkpoint, reduce_lr, early_stop])


plt.rcParams["figure.figsize"] = (6,6)

plt.title("точности обучения и валидации")
acc = history.history["acc"]
epochs = range(1, len(acc) + 1)
val_acc = history.history["val_acc"]
plt.plot(epochs, acc, "red", label='обучения')
plt.plot(epochs, val_acc, "blue", label="валидации")
plt.legend()
plt.figure()
plt.title("потери обучения и валидации")
loss = history.history["loss"]
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "red", label="обучения")
plt.plot(epochs, val_loss, "blue", label="валидации")
plt.legend()

plt.show()
```