МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет)»

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

# ОТЧЁТ

к лабораторной работе №1 по дисциплине

«Нейронные сети глубокого обучения»

Тема: «Кто за ЗОЖ?»

Студент В.А. Артамонов

Студент Д.А. Смирнов

Преподаватель А.Н. Жданова

Самара 2022

СОДЕРЖАНИЕ

[ОТЧЁТ 1](#_Toc99091104)

[1 Постановка задачи 3](#_Toc99091105)

[2 Библиотеки python 4](#_Toc99091106)

[3 Архитектура нейронной сети 5](#_Toc99091107)

[4 Метод обучения 9](#_Toc99091108)

[5 Вычислительные эксперименты 12](#_Toc99091109)

[6 Результаты работы 14](#_Toc99091110)

[7 Вывод 14](#_Toc99091111)

[Список использованных источников 15](#_Toc99091112)

[Приложение А 16](#_Toc99091113)

[Исходный код 16](#_Toc99091114)

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: ω ∈ Ω – объект распознавания (образ).

g(ω): Ω → M, M = {1, 2, …, m} – индикаторная функция, разбивающая пространство образов Ω на m непересекающихся классов Ω1, Ω2, …, Ωm. Индикаторная функция неизвестна наблюдателю.

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

x(ω): Ω → X – функция, ставящая в соответствие каждому объекту ω точку x(ω) пространстве признаков. Вектор x(ω) – это образ объекта, воспринимаемый наблюдателем. В пространстве признаков определены непересекающиеся множества точек Ki ∈ X, i = 1, 2, …, m, соответствующих образам одного класса.

ĝ(ω): X → M – решающее правило – оценка для g(ω) на основании x(ω), т.е. ĝ(x) = ĝ(x(ω)).

Пусть xj = x(ωj), j = 1, 2, …, N – доступная наблюдателю информация о функциях g(ω) и x(ω), но сами этим функции наблюдателю неизвестны. Тогда (gj, xj), j = 1, 2, …, N – есть множество прецедентов.

Задача заключается в построении такого решающего правила ĝ(ω), чтобы распознавание проводилось с минимальным числом ошибок.

Обычный случай – считать пространство признаков евклидовым, т.е. X = Rl. 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{ĝ(x(ω)) ≠ g(ω)}.

1. Библиотеки python

В лабораторной работе использовались такие библиотеки как: tensorflow и numpy.

* 1. TensorFlow

TensorFlow-это библиотека с открытым исходным кодом для приложений машинного обучения. Это система второго поколения Google Brain, после замены близкого источника Неверия, и используется Google как для исследовательских, так и для производственных приложений. Приложения TensorFlow могут быть написаны на нескольких языках: Python, Go, Java и C. Эта статья посвящена его версии Python и рассматривает установку библиотеки, базовые низкоуровневые компоненты и построение нейронной сети с нуля для выполнения обучения на реальном наборе данных.

Продолжительность обучения нейронных сетей глубокого обучения часто является узким местом в более сложных сценариях. Поскольку нейронные сети, а также другие алгоритмы ML в основном работают с матричными умножениями, их гораздо быстрее запускать на графических процессорах (GPU), а не на стандартных центральных процессорах (CPU).

TensorFlow поддерживает как процессоры, так и графические процессоры, и Google даже выпустила собственное специализированное оборудование для вычислений в облаке, называемое Tensor Processing Unit (TPU), которое обеспечивает лучшую производительность среди различных процессоров [1].

* 1. NumPy

Numeric Python (NumPy) - это несколько модулей для вычислений с многомерными массивами, необходимых для многих численных приложений. Массив - это набор однородных элементов, доступных по индексам. Массивы модуля Numeric могут быть многомерными, то есть иметь более одной размерности. Количество размерностей и длина массива по каждой оси называются формой массива (shape). Размещение массива в памяти проводится в соответствии с опциями и может быть выполнено как в языке С (по последнему индексу), как в языке Fortran (по первому индексу) или беспорядочно.

Особо подчеркнем отличие массива от набора данных (списка или кортежа). Величины, входящие в массив, имеют одинаковый тип и их количество жестко задается при инициализации. Элементы массива не являются объектами, это переменные в обычном понимании этого слова. Массивы позволяют экономить память и увеличивать скорость работы с большим количеством однотипных данных.

Как представлены массивы в Python? В Python массивы – это объекты, содержащие буфер данных и информацию о форме, размерности, типе данных и т.д. Как и у любого объекта, у массива можно менять атрибуты напрямую: array.shape=(2,3) или через вызов функции np.reshape(array,(2,3)). Такая же ситуация и с методами (функциями для массивов), заданными в этом классе, многие из них могут вызываться как методы array.resize(2,4) или как самостоятельные функции NumPy: np.resize(array,(2,4)). Некоторые функции являются только методами: array.flat, array.flatten, array.T. Для правильного использования таких функций, необходимо обращаться к их описанию [2].

1. Архитектура нейронной сети

Глубокие остаточные сети, такие как популярная модель ResNet-50, представляют собой сверточную нейронную сеть (CNN) глубиной 50 слоев. Остаточная нейронная сеть (ResNet) - это [искусственная нейронная сеть (ANN)](https://viso.ai/deep-learning/ann-and-cnn-analyzing-differences-and-similarities/), которая складывает остаточные блоки друг на друга, образуя сеть.

Для классификации изображения фруктов и овощей по разновидностям была использована сверточная нейронная сеть ResNet50.

В последние годы область [компьютерного зрения](https://viso.ai/computer-vision/what-is-computer-vision/) претерпела далеко идущие преобразования в связи с внедрением новых технологий. В результате этих достижений стало возможным[, что модели компьютерного зрения](https://viso.ai/deep-learning/ml-ai-models/) превзошли людей в эффективном решении различных проблем, связанных с [распознаванием изображений](https://viso.ai/computer-vision/image-recognition/), [обнаружением объектов](https://viso.ai/deep-learning/object-detection/), [распознаванием лиц](https://viso.ai/deep-learning/deep-face-recognition/), [классификацией изображений](https://viso.ai/computer-vision/image-classification/)и т. Д.

В этой связи [особого упоминания заслуживает внедрение глубоких сверточных нейронных сетей или CNNS](https://viso.ai/deep-learning/deep-neural-network-three-popular-types/). Эти сети широко используются для анализа визуальных изображений с замечательной точностью.

Но, хотя это дает нам возможность добавлять больше слоев в CNNS для решения более сложных задач в области компьютерного зрения, он имеет свой собственный набор проблем. Было замечено, что обучение нейронных сетей становится более сложным с увеличением количества добавленных слоев, а в некоторых случаях и точность снижается.

Именно здесь использование ResNet приобретает важное значение. Более глубокие нейронные сети сложнее обучать. С Resnet становится возможным превзойти трудности обучения очень глубоких нейронных сетей.

* 1. Что такое ResNet-50

ResNet имеет много вариантов, которые работают на одной и той же концепции, но имеют разное количество слоев. Resnet50 используется для обозначения варианта, который может работать с 50 слоями нейронной сети.

Глубокие нейронные сети в компьютерном зрении

При работе с глубокими сверточными нейронными сетями для решения проблемы, связанной с компьютерным зрением, специалисты по машинному обучению занимаются укладкой большего количества слоев. Эти дополнительные слои помогают решать сложные задачи более эффективно, поскольку различные слои могут быть обучены для различных задач, чтобы получить высокоточные результаты.

В то время как количество сложенных слоев может обогатить особенности модели, более глубокая сеть может показать проблему деградации. Другими словами, по мере увеличения количества слоев [нейронной сети](https://viso.ai/deep-learning/deep-neural-network-three-popular-types/) уровни точности могут насыщаться и медленно ухудшаться после точки. В результате производительность модели ухудшается как на обучающих, так и на тестовых данных.

Эта деградация не является результатом переобучения. Вместо этого это может быть результатом инициализации сети, функции оптимизации или, что более важно, проблемы исчезновения или взрыва градиентов.

Для чего используется глубокое остаточное обучение

ResNet был создан с целью решения именно этой проблемы. Глубокие остаточные сети используют остаточные блоки для повышения точности моделей. Концепция “пропущенных соединений”, лежащая в основе остаточных блоков, является сильной стороной этого типа нейронной сети.

* 1. Архитектура ResNet-34

Первой архитектурой ResNet была Resnet-34 (найти [исследовательскую работу здесь](https://arxiv.org/pdf/1512.03385.pdf)), которая включала в себя вставку коротких соединений при превращении простой сети в ее остаточный сетевой аналог. В этом случае простая сеть была вдохновлена нейронными сетями VGG (VGG-16, VGG-19), а сверточные сети имели фильтры 3×3. Однако, по сравнению с [VGGNets](https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/), ResNets имеют меньше фильтров и более низкую сложность. 34-слойная ResNet достигает производительности 3,6 млрд флопов по сравнению с 1,8 млрд флопов меньших 18-слойных ResNet.

Он также следовал двум простым правилам проектирования – слои имели одинаковое количество фильтров для одного и того же выходного размера карты объектов, и количество фильтров удваивалось в случае, если размер карты объектов уменьшался вдвое, чтобы сохранить временную сложность для каждого слоя. Он состоял из 34 взвешенных слоев.

В эту обычную сеть были добавлены соединения быстрого доступа. Хотя входные и выходные измерения были одинаковыми, ярлыки идентификаторов использовались непосредственно. С увеличением размеров необходимо было рассмотреть два варианта. Во-первых, ярлык по-прежнему будет выполнять сопоставление идентификаторов, в то время как дополнительные нулевые записи будут дополнены для увеличения измерений. Другой вариант состоял в том, чтобы использовать ярлык проекции для сопоставления размеров.

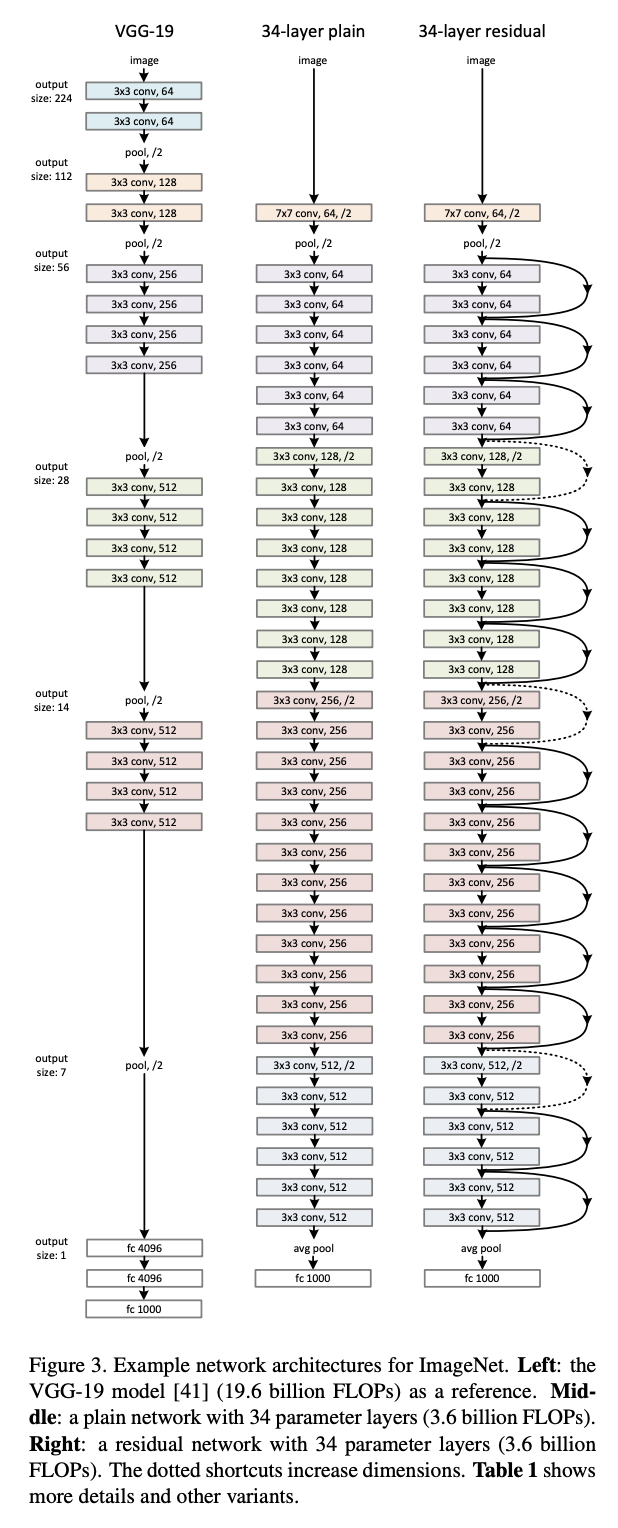


Рисунок – 1 Остаточные сети или реснеты

* 1. Архитектура ResNet-50

Хотя архитектура Resnet50 основана на вышеупомянутой модели, есть одно существенное различие. В этом случае строительный блок был модифицирован в конструкцию узкого места из-за опасений по поводу времени, затраченного на обучение слоев. Это использовало стек из 3 слоев вместо более ранних 2.

Поэтому каждый из 2-слойных блоков в Resnet34 был заменен 3-слойным блоком узких мест, образуя архитектуру Resnet 50. Это имеет гораздо более высокую точность, чем 34-слойная модель ResNet. 50-слойная ResNet достигает производительности 3,8 млрд флопов.

1. 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 при помощи фильтров

Первый слой нейронной сети принимает все пиксели в изображении. После того, как все данные введены в сеть, к изображению применяются различные фильтры, которые формируют понимание различных частей изображения. Это извлечение признаков, которое создает «карты признаков»

Этот процесс извлечения признаков из изображения выполняется с помощью «сверточного слоя», и свертка просто формирует представление части изображения. Именно из этой концепции свертки мы получаем термин “Сверточная нейронная сеть” (Convolutional Neural Network, CNN) — тип нейронной сети, наиболее часто используемый в классификации и распознавании изображений [4].

Размер фильтра влияет на то, сколько пикселей проверяется за один раз. Общий размер фильтра, используемого в CNN, равен 3, и он охватывает как высоту, так и ширину, поэтому фильтр проверяет область пикселей 3 x 3.

В то время как размер фильтра покрывает высоту и ширину фильтра, глубина фильтра также должна быть указана [4].

«глубина» ‑‑ это количество цветовых каналов, которые имеет изображение. Изображения в градациях серого (не цветные) имеют только 1 цветной канал, в то время как цветные изображения имеют глубину в 3 канала.

Все это означает, что для фильтра размером в 3, примененного к полноцветному изображению, итоговые размеры этого фильтра будут 3 x 3 x 3. Для каждого пикселя, охватываемого этим фильтром, сеть умножает значения фильтра на значения самих пикселей, чтобы получить числовое представление этого пикселя. Затем этот процесс выполняется для всего изображения, чтобы получить полное представление. Фильтр перемещается по остальной части изображения в соответствии с параметром, называемым «шаг», который определяет, на сколько пикселей должен быть перемещен фильтр после того, как он вычислит значение в своей текущей позиции. Обычный размер шага для CNN ‑‑ 2.

Конечным результатом всех этих расчетов является карта признаков. Этот процесс обычно выполняется с несколькими фильтрами, которые помогают сохранить сложность изображения.

Функции активации

После того, как карта признаков изображения была создана, значения, представляющие изображение, передаются через функцию активации или слой активации. Функция активации принимает эти значения, которые благодаря сверточному слою находятся в линейной форме (то есть просто список чисел) и увеличивает их нелинейность, поскольку сами изображения являются нелинейными [4].

Объединение слоёв

После активации данные отправляются через объединяющий слой. Объединение «упрощает» изображение: берёт информацию, которая представляет изображение, и сжимает её. Процесс объединения в пул делает сеть более гибкой и способной лучше распознавать объекты и изображения на основе соответствующих функций [4].

Когда мы смотрим на изображение, нас, как правило, волнует не вся информация (например, что на заднем плане изображения), а только признаки, которые нас интересуют — люди, животные и т. д.

Аналогично, объединяющий слой в CNN избавится от ненужных частей изображения, оставив только те части, которые он считает релевантными, в зависимости от заданного размера объединяющего слоя.

Поскольку сеть должна принимать решения относительно наиболее важных частей изображения, расчёт идёт на то, что она изучит только те части изображения, которые действительно представляют суть рассматриваемого объекта. Это помогает предотвратить “переобучение” — когда сеть слишком хорошо изучает все аспекты учебного примера и уже не может обобщать новые данные, поскольку учитывает нерелевантные отличия.

Существуют различные способы объединения значений, но чаще всего используется максимальное объединение. Максимальное объединение подразумевает взятие максимального значения среди пикселей в пределах одного фильтра (в пределах одного фрагмента изображения). Это отсеивает 3/4 информации, при условии использования фильтра размером 2 x 2.

Максимальные значения пикселей используются для того, чтобы учесть возможные искажения изображения, а количество параметров (размер изображения) уменьшены, чтобы контролировать переобучение. Существуют и другие принципы объединения, такие как среднее или суммарное объединение, но они используются не так часто, поскольку максимальное объединение даёт большую точность [4].

Сжатие

Последние слои нашей CNN — плотно связанные слои — требуют, чтобы данные были представлены в форме вектора для дальнейшей обработки. По этой причине данные необходимо «свести воедино». Для этого значения сжимаются в длинный вектор или столбец последовательно упорядоченных чисел [4].

Полностью связанный слой

Конечные слои CNN представляют собой плотно связанные слои или искусственную нейронную сеть (Artificial neural networks (ANN)). Основной функцией ANN является анализ входных признаков и объединение их в различные атрибуты, которые помогут в классификации. Эти слои образуют наборы нейронов, которые представляют различные части рассматриваемого объекта, а набор нейронов может представлять собой, например, висячие уши собаки или красноту яблока. Когда достаточное количество этих нейронов активируется в ответ на входное изображение, то оно будет классифицировано как объект [4].

Для решения поставленной задачи была построена свёрточная нейронная сеть, которая была обучена на 6231 изображений, принадлежащих к 16 видам фруктов и овощей. Сеть была обученна для того, чтобы предсказывать по изображению название фрукта или овоща.

Для обучения использовался набор данных Fruits 360 Dataset [5]. Набор данных представляет собой цветные изображения в формате JPEG.

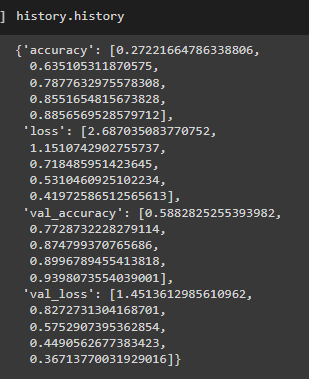
В особенности метода обучения входит: архитектура сети (ResNet50), оптимизатор градиентного спуска (Adam), функция потерь градиентного спуска (категориальная перекрёстная энтропия), метрика (точность), количество эпох (5).

1. Вычислительные эксперименты

Вычислительные эксперименты заключались в обучении нейроной сети с архитектурой – ResNet50. Целью обучения было ؘ– обучить сеть предсказывать что это за фрукт или овощ по изображению, всего 40 классов. Метод обучения сетей описан в пункте 3.

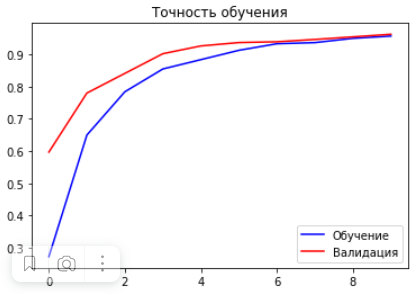
На рисунке 2 представлена история обучения сети с архитектурой ResNet50 на наборе данных Fruits-360.

Рисунок 2 – История обучения сети



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

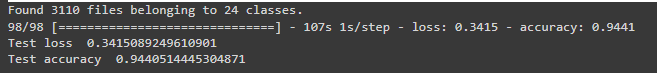
Рисунок 3 – Графики зависимости точности от эпохи для сети ResNet50



1. Результаты работы

Проверка обученной сети была на тестовых данных и на рисунке 4 можно увидеть какая точность у модели и какие потери.

Рисунок 4 – Точность и потери сети на тестовых данных



Проверка прогнозов

Давайте посмотрим на 12-е изображение, прогноз к нему и массив прогнозов. Метки правильных прогнозов выделены синим цветом, а метки неправильных прогнозов — красным. Число указывает процент (из 100) предсказанной метки.

На рисунке 5 изображен прогноз к 12 му изображению. Также, на рисунке 6 изображен массив прогнозов



Рисунок 5 – Прогноз на 12-е изображение

Используйте обученную модель

Наконец, используем обученную модель, чтобы сделать прогноз для случайных изображений из сети интернет. На рисунках 7-9 изображены прогнозы объектов.

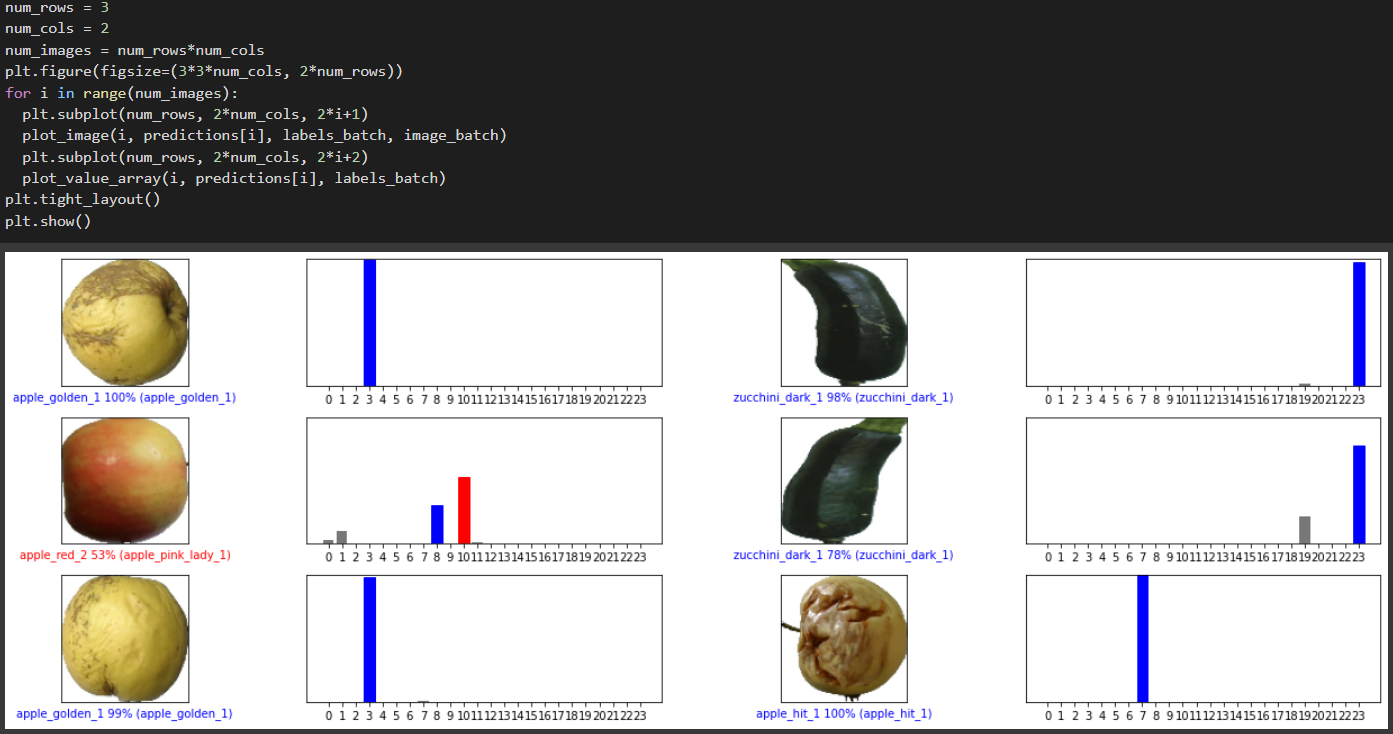


Рисунок 6 – Массив прогнозов



Рисунок 7 – Прогноз случайного объекта №1

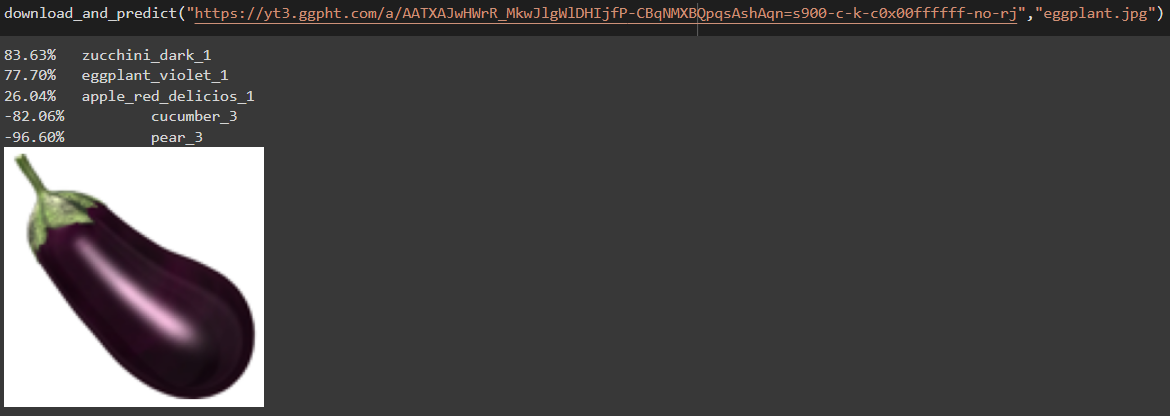


Рисунок 8 – Прогноз случайного объекта №2



Рисунок 9 – Прогноз случайного объекта №3

Исходный код лабораторной работы представлен в приложении А и доступен по ссылке – https://colab.research.google.com/drive/1rzI8POzNnvRMM9t1VxJp\_9ykGau1xsj4?usp=sharing

1. Вывод

В лабораторной работе была реализована нейронная сеть при помощи фреймворка TensorFlow и библиотеки Numpy для классификации изображений фруктов и овощей, а также, были изучены: метод обучения, параметры обучения, параметры нейронной сети.

**Список использованных источников**

1. Учебник по нейронной сети TensorFlow [Электронный ресурс]. URL: <https://pythobyte.com/tensorflow-neural-network-tutorial-15134/> (дата обращения: 24.03.2022)
2. NumPy. Матричные вычисления [Электронный ресурс]. URL: <https://physics.susu.ru/vorontsov/language/numpy.html> (дата обращения: 24.03.2022)
3. Глубокие остаточные сети (ResNet, ResNet 50) [Электронный ресурс]. URL: https://viso.ai/deep-learning/resnet-residual-neural-network/ (дата обращения: 24.03.2022).
4. Распознавание изображений на Python с помощью TensorFlow и Keras [Электронный ресурс]. URL: https://evileg.com/ru/post/619/ (дата обращения 24.03.2022)
5. Kaggle – Fruits 360 [Электронный ресурс]. URL: https://www.kaggle.com/moltean/fruits (дата обращения: 24.03.2022).

Приложение А

Исходный код

import numpy as np

import matplotlib.pyplot as plt

import os

import pathlib

from zipfile import ZipFile

import PIL

import tensorflow as tf

from google.colab import drive

drive.mount('/content/gdrive', force\_remount=True)

!cp /content/gdrive/'My Drive'/fruits-360.zip .

!unzip /content/fruits-360.zip

train\_directory = pathlib.Path("/content/fruits-360-original-size/fruits-360-original-size/Training")

test\_directory = pathlib.Path("/content/fruits-360-original-size/fruits-360-original-size/Test")

image\_count = len(list(train\_directory.glob('\*/\*.jpg')))

image\_count

batch\_size = 32

img\_height = 100

img\_width = 100

train\_ds = tf.keras.preprocessing.image\_dataset\_from\_directory(

    train\_directory,

    validation\_split=0.2,

    subset='training',

    seed=42,

    image\_size=(img\_height, img\_width),

    batch\_size=batch\_size

)

val\_ds = tf.keras.preprocessing.image\_dataset\_from\_directory(

    train\_directory,

    validation\_split=0.2,

    subset='validation',

    seed=42,

    image\_size=(img\_height, img\_width),

    batch\_size=batch\_size

)

class\_names = train\_ds.class\_names

num\_classes = len(class\_names)

for image\_batch, labels\_batch in train\_ds.take(1):

    print(image\_batch.shape)

    print(labels\_batch.shape)

#Создаем модель

AUTOTUNE = tf.data.experimental.AUTOTUNE

train\_ds = train\_ds.cache().shuffle(1000).prefetch(buffer\_size=AUTOTUNE)

val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)

data\_augmentation = tf.keras.Sequential([

    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),

    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2)

])

preprocess\_input = tf.keras.applications.resnet.preprocess\_input

base\_model = tf.keras.applications.resnet.ResNet50(

    input\_shape=(img\_height, img\_width, 3),

    include\_top=False,

    weights='imagenet'

)

base\_model.trainable = False

global\_average\_layer = tf.keras.layers.GlobalAveragePooling2D()

prediction\_layer = tf.keras.layers.Dense(num\_classes)

inputs = tf.keras.Input(shape=(100, 100, 3))

x = data\_augmentation(inputs)

x = preprocess\_input(x)

x = base\_model(x, training=False)

x = global\_average\_layer(x)

x = tf.keras.layers.Dropout(0.2)(x)

outputs = prediction\_layer(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

optimizer = tf.keras.optimizers.Adam(learning\_rate=0.0001)

model.compile(

    optimizer=optimizer,

    loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

    metrics=['accuracy']

)

model.summary()

#Тренируем модель

epochs = 5

history = model.fit(

    train\_ds,

    epochs=epochs,

    validation\_data=val\_ds

)

train\_loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range (epochs)

plt.figure(figsize=(12, 10))

plt.plot(epochs\_range, train\_loss, label="Training Loss")

plt.plot(epochs\_range, val\_loss, label="Validation Loss")

plt.legend(loc='upper left')

plt.title('Training and Validation Loss')

plt.show()

train\_loss

val\_loss

np.argmin(val\_loss)

history.history

test\_ds = tf.keras.preprocessing.image\_dataset\_from\_directory(

    test\_dir,

    image\_size=(img\_height, img\_width),

    batch\_size=batch\_size

)

class\_names\_test = test\_ds.class\_names

num\_classes = len(class\_names)

scores = model.evaluate(test\_ds)

print('Test loss ', scores[0])

print('Test accuracy ', scores[1])