

Федеральное государственное образовательное бюджетное учреждение
высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)**

Департамент анализа данных и машинного обучения

КУРСОВАЯ РАБОТА
по дисциплине «Машинное обучение»

на тему:

**«Предварительный анализ данных и построение признаков в задачах
выявления наличия соланина в клубнях картофеля.»**

Выполнил:

Никитин А. Д.
группа ПМ21-2

Научный руководитель:

К.Т.Н., доцент
Моисеев Г. В.

Москва, 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
Глава 1. Необходимые библиотеки.....	4
1.1 TensorFlow, Keras и YOLO.....	4
1.2 Какие методы будем использовать.	6
Глава 2. Предварительный анализ данных, сбор информации, а также построение и обучение модели нейронной сети.....	8
2.1 Подготовка данных и описание датасета	8
2.2 Создание нейронной сети	12
2.3 Оценка качества обучения нейронной сети.	15
2.4 Использование готовой нейронной сети для распознавания изображений.....	18
2.5 Анализ результатов.....	19
Заключение	23
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	23
ПРИЛОЖЕНИЯ	24

ВВЕДЕНИЕ

Способности и возможности людей обрабатывать информацию ограничены, особенно в условиях возрастающих объёмов информации, поэтому появилась потребность использовать способы хранения, обработки и передачи информации (информационные технологии), отчуждённые от человека. Исходя из этого с каждым годом популярность информационных технологий растёт и затрагивает различные сферы жизни человека.

В 1959 году была создана первая программа по игре в шашки, которая умела играть сама с собой и обучаться самостоятельно. С этого момента и началась история машинного обучения. В 2011 году Google основал Google Brain — подразделение, занимавшееся проектами в области ИИ. Спустя ещё три года свои платформы по машинному обучению появились у Amazon и Microsoft, а Facebook внедрила в работу DeepFace — алгоритм, умеющий распознавать человеческие лица.

Итак, машинное обучение — это раздел искусственного интеллекта (ИИ) и информатики, который помогает компьютеру обучаться без непосредственных инструкций. Машинное обучение является важным компонентом растущей области науки о данных. Благодаря машинному обучению компьютеры учатся распознавать на фотографиях и рисунках не только лица, но и пейзажи, предметы, текст и цифры. Более того, уже существует программное обеспечение, способное без участия человека писать новостные статьи (на тему экономики и, к примеру, спорта).

Задачи машинного обучения делятся на 2 вида:

- обучение без учителя.

Пример обучения без учителя - задача кластеризации.

- обучение с учителем, примером которого является задача классификации изображений. Данный метод предполагает, что модели будет известен правильный ответ.

Постановка задачи:

- 1) Предварительный сбор данных (составление датасета) из изображений, разделение их по видам.
- 2) Создание набора данных для тестирования.
- 3) Создание нейронной сети.
- 4) Компиляция модели и ее обучение.
- 5) Оценка качества обучения и проверка.

Для своих целей я буду использовать библиотеку TensorFlow языка Python и пакет, работающий с данной библиотекой Keras.

Актуальность данной работы на сегодняшний день достаточно высока. Ввиду сложности ручной оценки качества картофеля, куда легче автоматизировать данный процесс, например на конвейере после сбора урожая, поэтому данная нейронная сеть может стать незаменимым помощником на фермах, выращивающих картофель.

Данная идея не нова и уже используется на многих фермах и производствах, реализующих производство или подготовку к продаже картофеля и не только.

На практике данный метод может показывать очень хороший результат и с вероятностью вплоть до 99% определять ядовитый картофель.

Глава 1. Необходимые библиотеки

1.1 TensorFlow, Keras и YOLO

На сегодняшний день большинство задач по распознаванию изображений используют либо TensorFlow в связке с Keras или используют YOLO. Так как YOLO написана на языке C++ и для работы требует наличие в графическом процессоре Nvidia Cuda ядра, работа с ней требует специфических условий и оборудования, это и стало для меня причиной отказа

от ее использования. Из преимуществ YOLO можно отметить ее быстрое действие в реальном времени.

В отличие от YOLO, TensorFlow – встроенная в Python библиотека и, так как обучение в нашем вузе происходит именно на нем, выбор стал очевиден.

TensorFlow — это библиотека для машинного обучения, группы технологий, которая позволяет обучать искусственный интеллект решению разных задач. Библиотека изначально разработана для Python и чаще всего используется с ним.

Существуют реализации TensorFlow для других языков: C#, C++, Go, Java, Swift и так далее. Они используются реже основной — главным образом для написания кода под специфичные платформы. Сама библиотека написана на языке Python с использованием быстрого и производительного C++ для решения математических задач. Поэтому она эффективно работает со сложными вычислениями.

Библиотека разработана Google как продолжение внутренней библиотеки компании. TensorFlow бесплатна, у нее открытый исходный код, который можно посмотреть на GitHub.

Сама библиотека включает в себя множество инструментов для разных направлений ML, но чаще всего используется для работы с нейронными сетями. Это структуры, вдохновленные устройством сетей нейронов в человеческой нервной системе. Нейронные сети состоят из программных элементов-«нейронов» и связей между ними, и такое устройство позволяет им обучаться. TensorFlow работает с обычными и глубокими нейронными сетями разных типов: рекуррентными, сверточными и так далее. Также она используется для машинного и глубокого обучения.

Примеры использования технологий — распознавание естественного языка, изображений и рукописных текстов, разнообразные задачи классификации или кластеризации, обработка больших данных.

Keras — это библиотека для языка программирования Python, которая предназначена для глубокого машинного обучения. Она позволяет быстрее создавать и настраивать модели — схемы, по которым распространяется и подсчитывается информация при обучении. Но сложных математических вычислений Keras не выполняет и используется как надстройка над другими библиотеками.

Keras с версии 2.3 — это надстройка над библиотекой TensorFlow, которая нужна для машинного обучения. TensorFlow выполняет все низкоуровневые вычисления и преобразования и служит своеобразным движком, математическим ядром. Keras же управляет моделями, по которым проходят вычисления.

До версии 2.3 Keras мог использовать в качестве движка вычислительные библиотеки Theano или CNTK. Но в новых версиях поддержка прекратилась, теперь библиотека работает только с TensorFlow.

1.2 Какие методы будем использовать.

В процессе работы мы создадим и обучим сверточную нейронную сеть. Convolutional neural network (CNN, ConvNet), или Сверточная нейронная сеть — класс глубоких нейронных сетей, часто применяемый в анализе визуальных образов. Сверточные нейронные сети являются разновидностью многослойного персептрона с использованием операций свёртки. Они нашли применение в распознавании изображений и видео, рекомендательных системах, классификации изображений, NLP (natural language processing) и анализе временных рядов.

Принцип работы операции свертки:

Операцию свёртки можно представить следующим алгоритмом:

Скользящее окно, называемое фильтром, с размером (n,n) двигается по входному признаку. Количество движений определяется заданным количеством фильтров.

Каждый полученный шаблон имеет форму (n,n,d) , где d — глубина входного признака.

Каждый шаблон умножается на своё ядро свёртки, в результате, формируется выходная карта признаков. Полученная выходная карта признаков имеет форму (h,w,N) , где h и w — длина и ширина, полученные в результате отсечения, а N — количество фильтров.

Количество фильтров — гиперпараметр, поэтому выбирается самостоятельно. Обычно его подбирают как степень двойки с увеличением количества фильтров по мере увеличения глубины архитектуры. А ядра свёртки являются обучаемыми параметрами.

Преобразование выходной карты признаков convolutional neural network (сверточные нейронные сети)

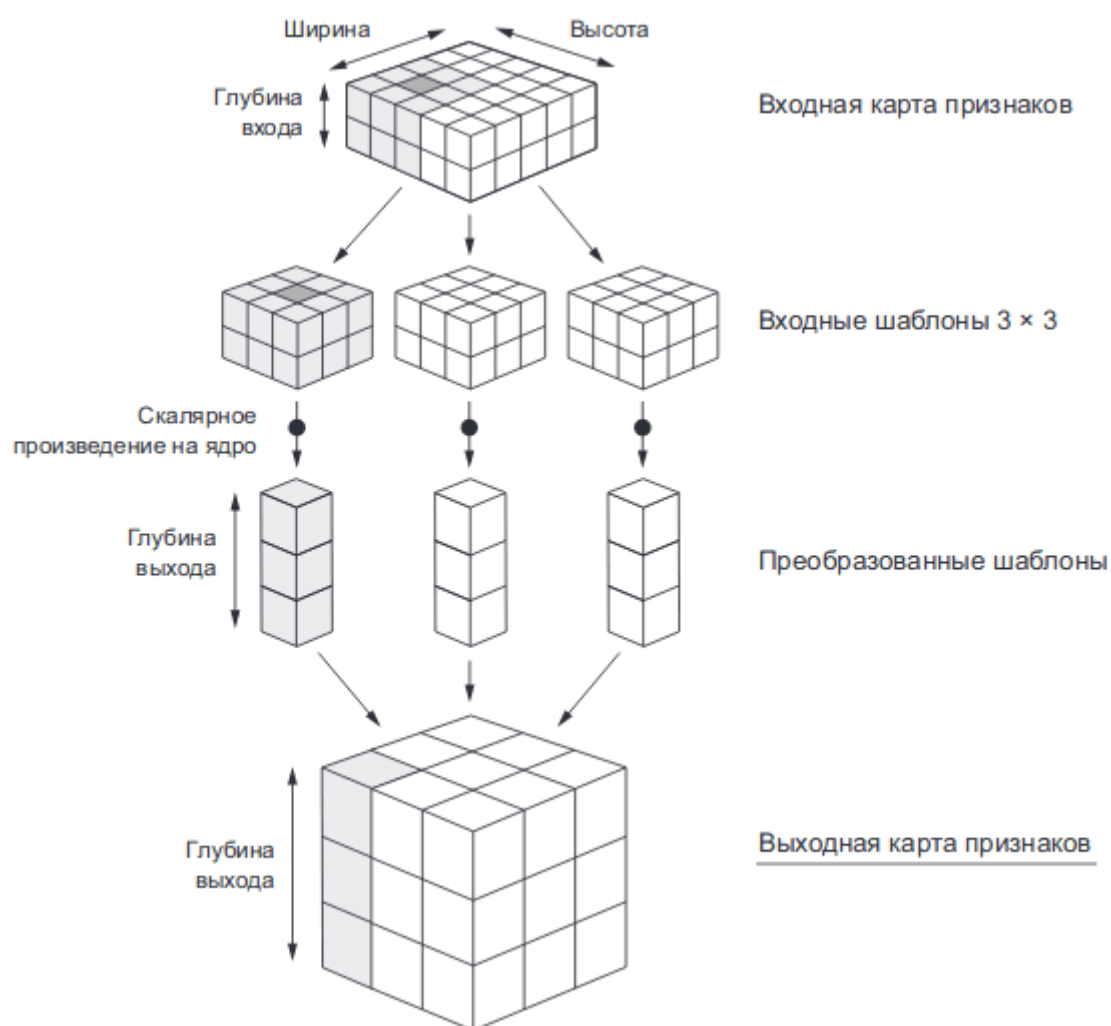


Рисунок 1.1 Принцип действия операция свёртки.

На сегодняшний день практически все задачи по распознаванию изображений используют именно этот метод.

Глава 2. Предварительный анализ данных, сбор информации, а также построение и обучение модели нейронной сети.

2.1 Подготовка данных и описание датасета

Цель данной курсовой работы является собрать достаточное количество фотографий клубней картофеля с соланином и без и обучить машину понимать, здоровая ли перед ней картошка или ядовитая.

Данная нейронная сеть могла бы очень помочь на ферме, где после сбора урожая требуется отделить ядовитые клубни от здоровых.

Первое, что необходимо сделать – это найти достаточное количество фотографий (не менее ста экземпляров каждого типа).

Для сбора фотографий я воспользовался поисковиком Yandex и скачал по соответствующим запросам по 108 картинок картофеля с соланином и без. Итоговые виды датасетов выглядят так:

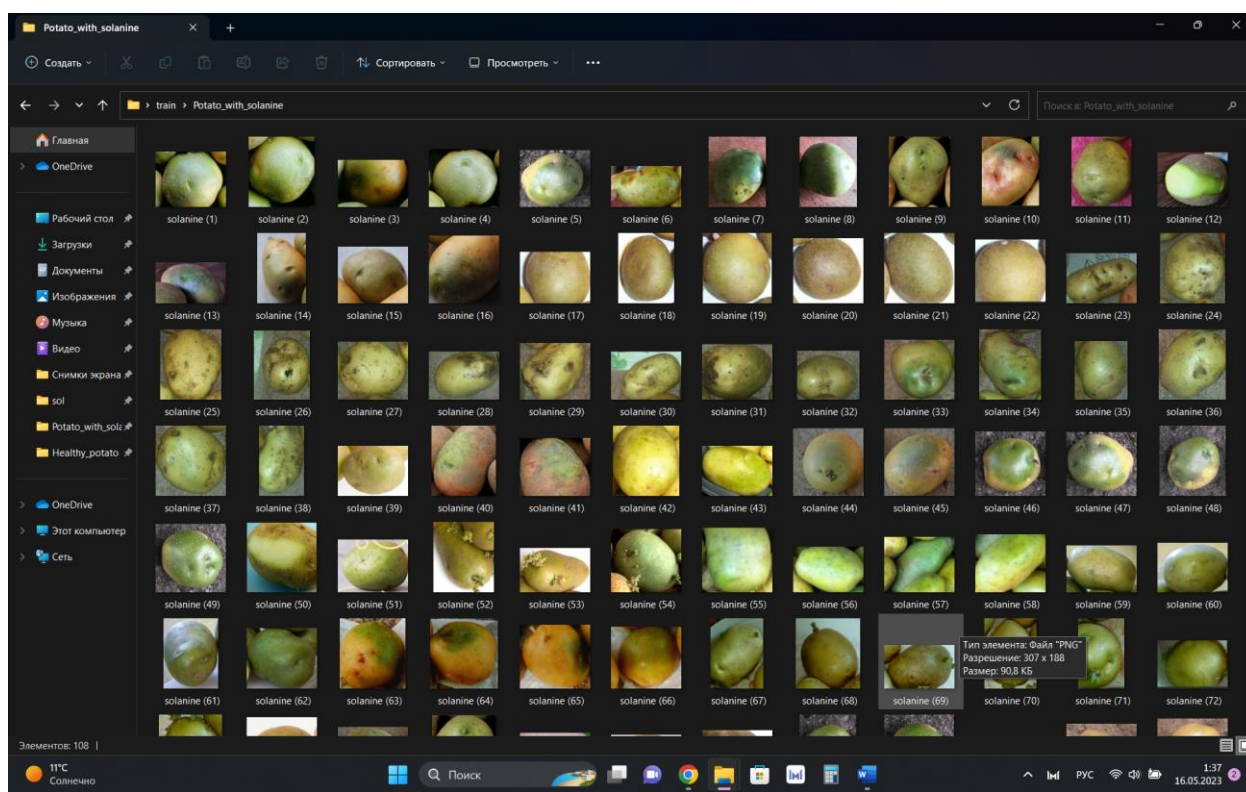


Рисунок 2.1 Вид датасета картофеля с соланином.

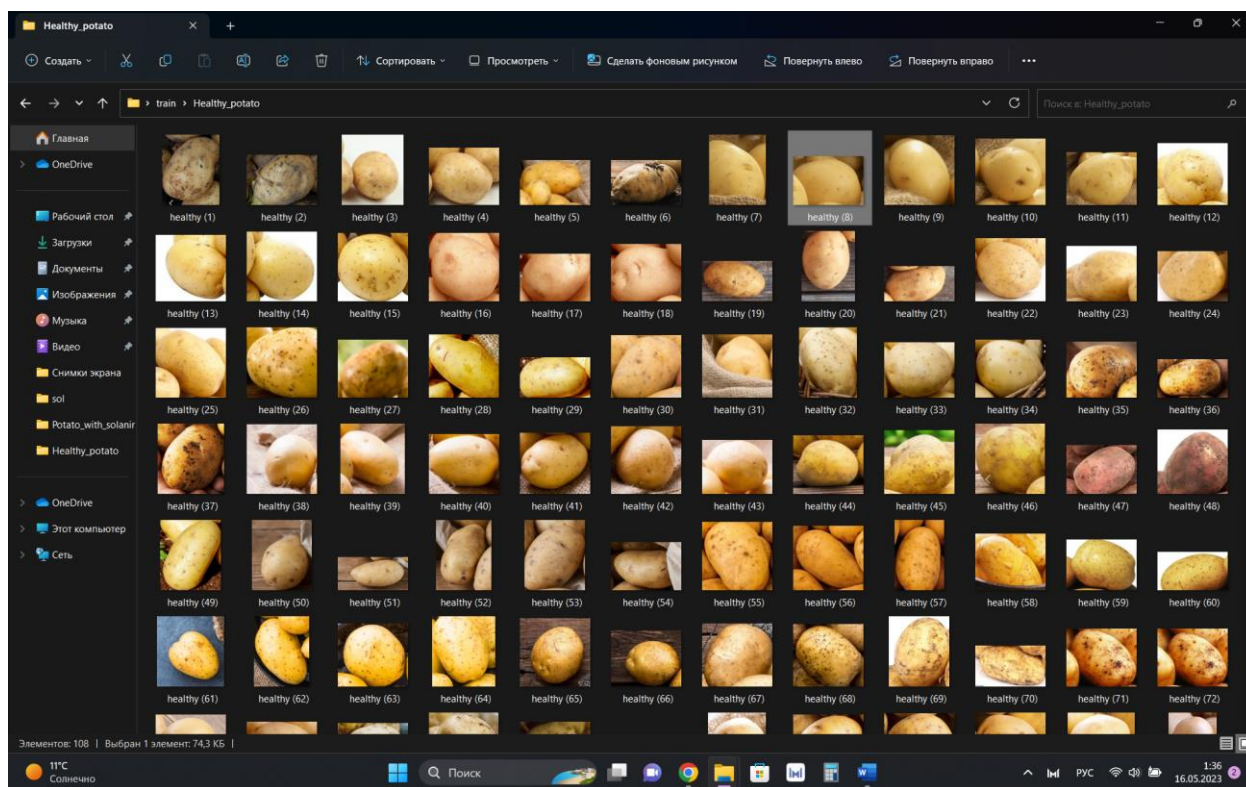


Рисунок 2.2 Вид датасета здорового картофеля.

Итого имеем:

108 изображений картофеля с соланином.

108 изображений картофеля без соланина.

Проведем подготовку датасета для обучения нейронной сети. Для начала установим константы `batch_size` – размер мини выборки в 16 единиц и `image_size` – размер изображений 100x100 пикселей. После чего подготовим изображения к работе, разбив их по директориям с выборкой для обучения и тестирования.

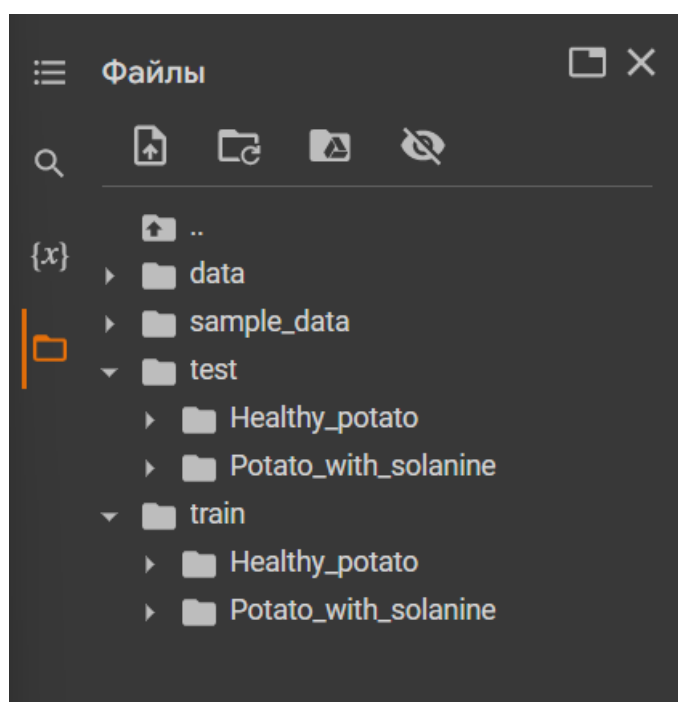


Рисунок 2.3 Итоговый вид файловой системы.

Для загрузки изображений в TensorFlow будем использовать утилиту `image_dataset_from_directory()`. Эта утилита создает TensorFlow датасет из каталога с изображениями, которые имеют специфическую структуру, где изображения одного класса находятся в одном каталоге. Наш набор данных устроен именно так:

```
Набор данных для обучения

✓ 1 [10] train_dataset = image_dataset_from_directory('train',
сек.      subset='training',
          seed=42,
          validation_split=0.1,
          batch_size=batch_size,
          image_size=image_size)

Found 180 files belonging to 2 classes.
Using 162 files for training.

Проверочный набор данных

✓ 0 [11] validation_dataset = image_dataset_from_directory('train',
сек.      subset='validation',
          seed=42,
          validation_split=0.1,
          batch_size=batch_size,
          image_size=image_size)

Found 180 files belonging to 2 classes.
Using 18 files for validation.

Названия классов в наборах данных.

✓ 0 [12] class_names = train_dataset.class_names
сек.      class_names

['Healthy_potato', 'Potato_with_solanine']
```

Рисунок 2.4 Создание набора данных для обучения и тестирования.

Нам для обучения нейронной сети нужно 3 набора данных: для обучения, для тестирования и для проверки, поэтому набор данных для обучения мы делим на 2 части. Для этого, когда мы вызываем утилиту `image_dataset_from_directory()`, мы указываем параметр `validation_split` именно он говорит о том, что 10 процентов набора данных для обучения будет использоваться в качестве проверочного набора данных. Указываем какой набор мы хотим создать (Для обучения “training”, для проверки “validation”), начальное значение `seed` для генератора случайных чисел, его обязательно использовать, если мы делим набор данных на две части для обучения и для проверки, так как набор данных перемешивается, нам необходимо выполнять операции одинаковым образом, когда мы вызываем утилиту `image_dataset_from_directory()` для создания обучающего и проверочного наборов данных. Указываем размер мини выборки `batch_size`. Наборы данных

в TensorFlow рассчитаны на работу с большим объемом данных, который не помещается в память, поэтому изображения не прочитываются в память целиком, а прочитается столько, сколько мы указали в размере мини выборки `batch_size`. В конце указываем размер изображений 100x100.

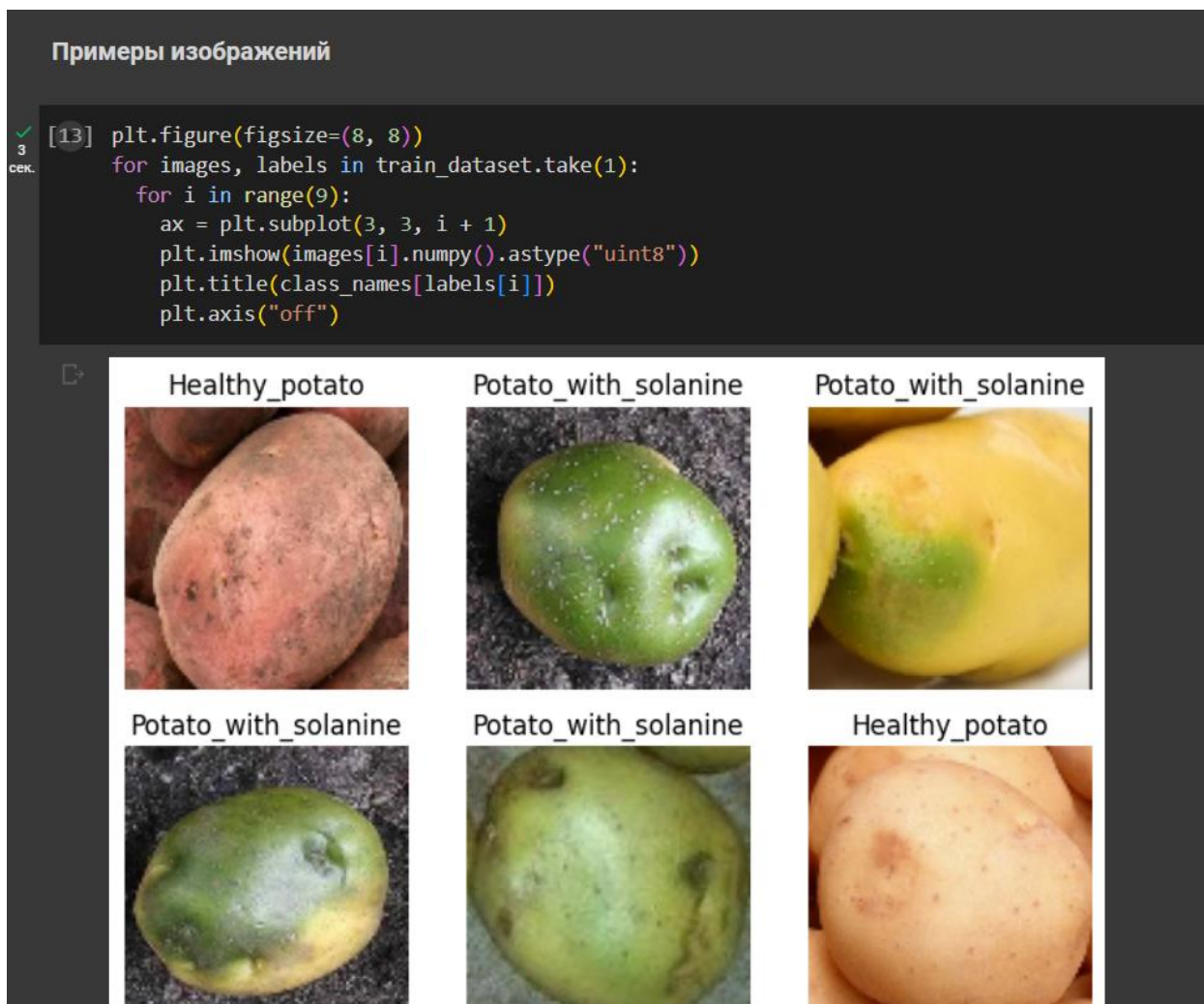


Рисунок 2.5 Примеры изображений из созданного датасета.

После создания обучающего и проверяющего датасетов создаем тестирующий датасет почти таким же образом, но без указания `seed`, `validation_split` и `subset`.

2.2 Создание нейронной сети

Для того, чтобы ускорить скорость обучения необходимо настроить производительность TensorFlow датасетов. Вызываем метод `prefetch`, это

означает, что будет выполнена предварительная загрузка нескольких мини выборок до того, как они понадобятся. Загрузка данных с диска – очень медленная операция, поэтому имеет смысл в то время, как графический процессор используется для обучения нейронной сети применять центральный процессор для того, чтобы загрузить данные с диска. Выполняем автоматическую настройку, теперь наши наборы данных готовы и настроены, и мы можем создавать нейронную сеть для распознавания картофеля с соланином из набора данных.

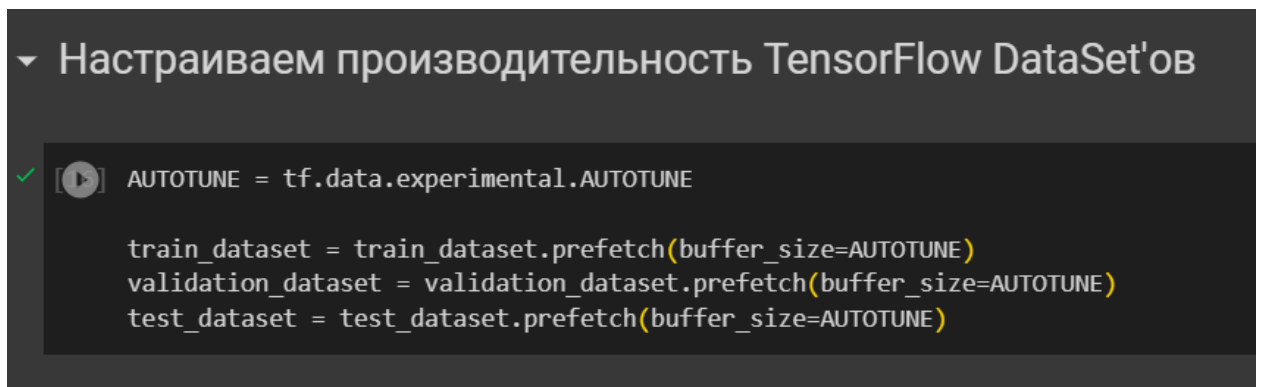


Рисунок 2.6 Настройка производительности TensorFlow датасетов.

Архитектуру нейронной сети я взял из статьи авторов, в которой описан набор данных, архитектура достаточно традиционная, в ней чередующиеся сверточные слои и слои подвыборки в размерах на свертке 5x5. В конце находится полносвязная часть для классификации и количество нейронов – 2 по количеству классов в исходном наборе данных.

▼ Создаем нейронную сеть

```
# Создаем последовательную модель
model = Sequential()
# Сверточный слой
model.add(Conv2D(16, (5, 5), padding='same',
                 input_shape=(100, 100, 3), activation='relu'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(128, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Полносвязная часть нейронной сети для классификации
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
# Выходной слой, 2 нейрона по количеству классов
model.add(Dense(2, activation='softmax'))
```

Рисунок 2.7 Создание нейронной сети.

После создания нейронной сети ее необходимо скомпилировать.

Компилируем модель

```
✓ [18] model.compile(loss='sparse_categorical_crossentropy',
0      optimizer="adam",
ек.    metrics=['accuracy'],
      run_eagerly=True)
```

Рисунок 2.8 Компиляция нейронной сети.

И теперь мы готовы к обучению нейронной сети. В метод `fit()` мы передаем набор данных для обучения `train_dataset`. Здесь один датасет содержит как изображения, так и правильные ответы. В качестве проверочного набора данных `validation_data` передаем проверочный набор данных `validation_dataset`, который так же содержит как изображения из проверочного набора данных, так и правильные ответы. Количество эпох обучения `epochs` равно 15. Размер мини выборки мы не указываем, так как его мы указали при создании наборов данных. Запускаем обучение.

```
[19] history = model.fit(train_dataset,
                        validation_data=validation_dataset,
                        epochs=15,
                        verbose=2)

Epoch 1/15
WARNING:tensorflow:5 out of the last 5 calls to <function BaseOptimizer.update_step_xla at 0x7f6f182df250> triggered tf.function retracing. Tracing is expensive and the excessive num
WARNING:tensorflow:6 out of the last 6 calls to <function BaseOptimizer.update_step_xla at 0x7f6f182df250> triggered tf.function retracing. Tracing is expensive and the excessive num
11/11 - 15s - loss: 47.1287 - accuracy: 0.4815 - val_loss: 0.6013 - val_accuracy: 0.7222 - 15s/epoch - 1s/step
Epoch 2/15
11/11 - 1s - loss: 0.7657 - accuracy: 0.5617 - val_loss: 0.8441 - val_accuracy: 0.3889 - 1s/epoch - 97ms/step
Epoch 3/15
11/11 - 2s - loss: 0.6787 - accuracy: 0.5494 - val_loss: 0.6572 - val_accuracy: 0.6667 - 2s/epoch - 141ms/step
Epoch 4/15
11/11 - 1s - loss: 0.6794 - accuracy: 0.5247 - val_loss: 0.6791 - val_accuracy: 0.3889 - 1s/epoch - 95ms/step
Epoch 5/15
11/11 - 1s - loss: 0.6561 - accuracy: 0.6235 - val_loss: 0.6197 - val_accuracy: 0.7222 - 1s/epoch - 100ms/step
Epoch 6/15
11/11 - 1s - loss: 0.6139 - accuracy: 0.6790 - val_loss: 0.6079 - val_accuracy: 0.5000 - 1s/epoch - 95ms/step
Epoch 7/15
11/11 - 1s - loss: 0.5755 - accuracy: 0.7654 - val_loss: 0.4293 - val_accuracy: 0.8889 - 1s/epoch - 98ms/step
Epoch 8/15
11/11 - 1s - loss: 0.5583 - accuracy: 0.6852 - val_loss: 0.5687 - val_accuracy: 0.7222 - 1s/epoch - 93ms/step
Epoch 9/15
11/11 - 1s - loss: 0.4039 - accuracy: 0.8580 - val_loss: 0.4182 - val_accuracy: 0.7778 - 1s/epoch - 92ms/step
Epoch 10/15
11/11 - 1s - loss: 0.3267 - accuracy: 0.8827 - val_loss: 0.4552 - val_accuracy: 0.8333 - 1s/epoch - 93ms/step
Epoch 11/15
11/11 - 1s - loss: 0.3073 - accuracy: 0.8951 - val_loss: 0.3724 - val_accuracy: 0.8333 - 1s/epoch - 96ms/step
Epoch 12/15
11/11 - 1s - loss: 0.3548 - accuracy: 0.8827 - val_loss: 0.2897 - val_accuracy: 0.7778 - 1s/epoch - 98ms/step
Epoch 13/15
11/11 - 2s - loss: 0.3102 - accuracy: 0.8889 - val_loss: 0.6032 - val_accuracy: 0.8889 - 2s/epoch - 159ms/step
Epoch 14/15
11/11 - 1s - loss: 0.2135 - accuracy: 0.9136 - val_loss: 0.8150 - val_accuracy: 0.6667 - 1s/epoch - 121ms/step
Epoch 15/15
11/11 - 1s - loss: 0.3985 - accuracy: 0.8642 - val_loss: 0.2598 - val_accuracy: 0.8889 - 1s/epoch - 96ms/step
```

Рисунок 2.9 Обучение нейронной сети.

После обучения нейронной сети видно, что доля правильных ответов `val_accuracy` равна 0.8889.

2.3 Оценка качества обучения нейронной сети.

Для оценки качества работы нейросети на тестовом наборе данных вызываем метод `model.evaluate()` и передаем в него тестовый набор данных. Тестовый набор данных содержит изображения и правильные ответы так же как наборы данных для обучения и проверки. Итак мы можем видеть, что доля правильных ответов на тестовом наборе данных равна 88.23%, что весьма неплохо для столь малого изначального датасета.

```
Оцениваем качество обучения сети

[20] # Оцениваем качество обучения модели на тестовых данных
      scores = model.evaluate(test_dataset, verbose=1)

      3/3 [=====] - 0s 23ms/step - loss: 0.3347 - accuracy: 0.8824

[21] print("Доля верных ответов на тестовых данных, в процентах:", round(scores[1] * 100, 4))

      Доля верных ответов на тестовых данных, в процентах: 88.2353
```

Рисунок 2.10 Оценка качества обучения.

Для наглядности построим графики правильных ответов и ошибки на обучающем и проверочном наборе данных.

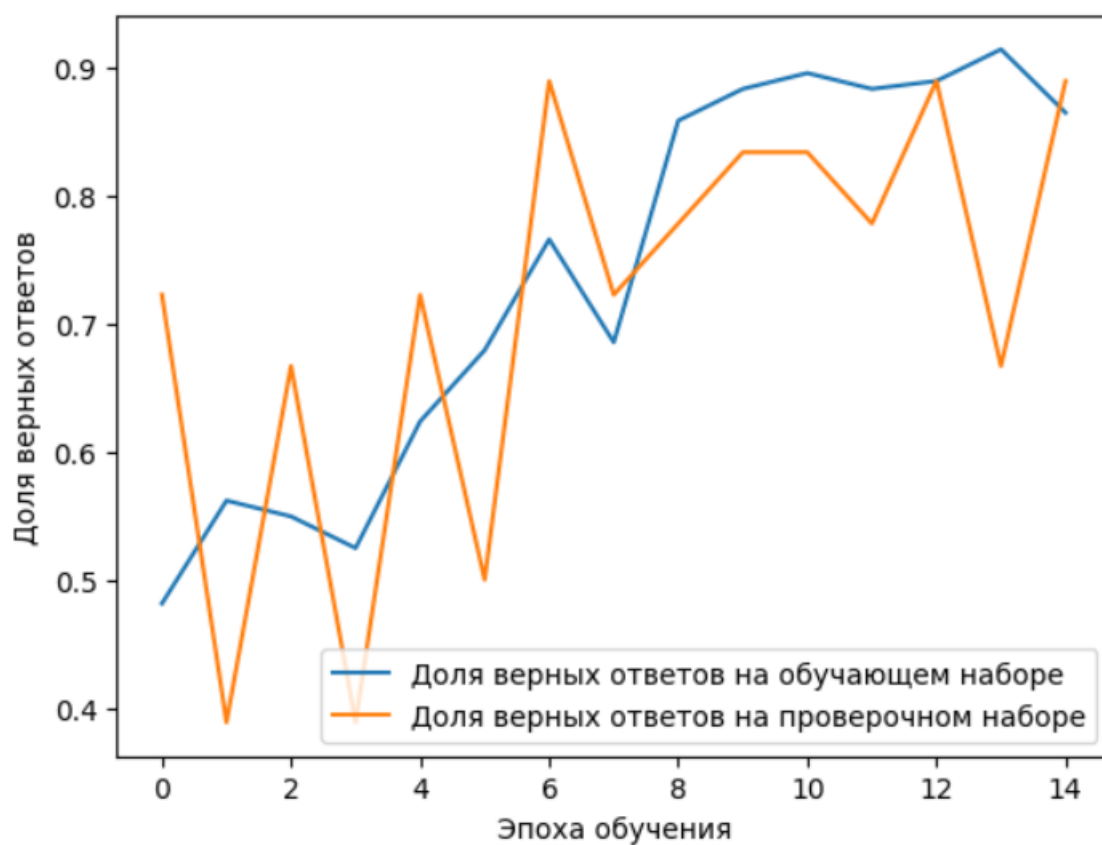


Рисунок 2.11 Доля верных ответов.

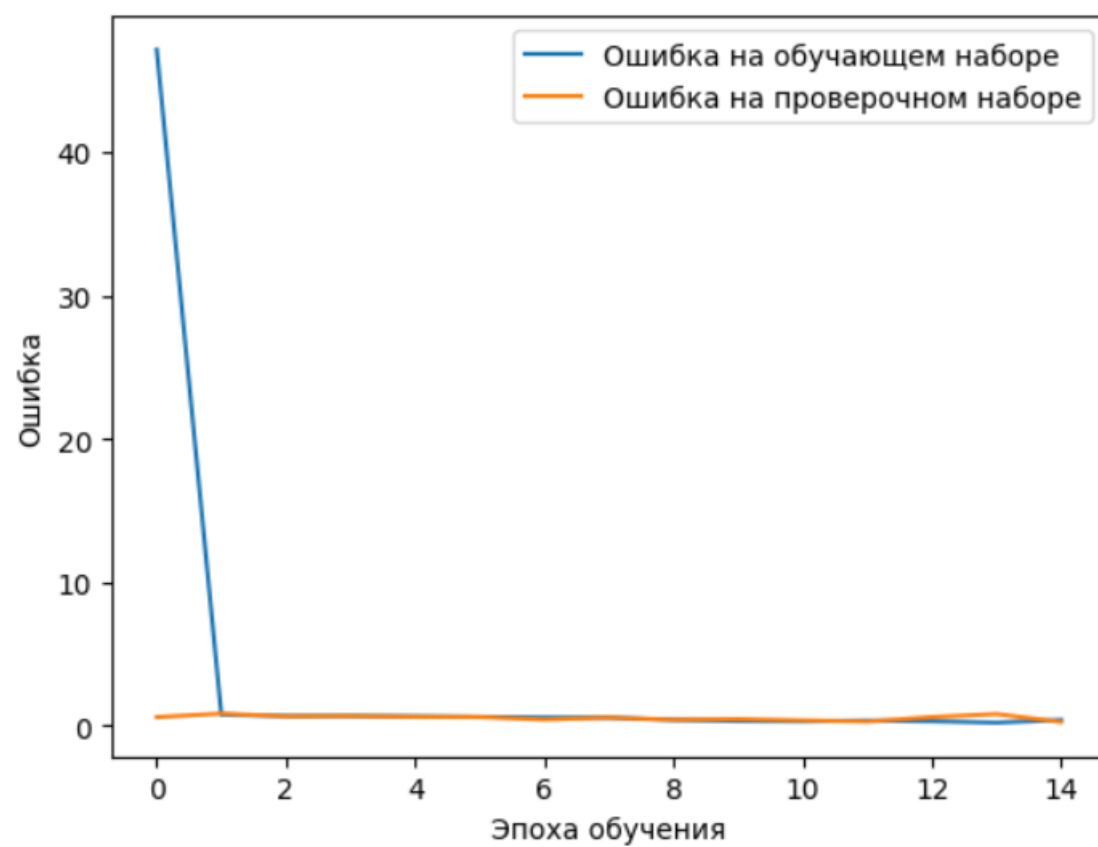


Рисунок 2.12 Доля ошибочных ответов.

2.4 Использование готовой нейронной сети для распознавания изображений.

Для начала распознаем изображения из тестового набора данных, используя метод `model.predict()`.

```
[26] prediction = model.predict(test_dataset)
3/3 [=====] - 0s 13ms/step
```

Рисунок 2.13.1 Распознавание изображений из тестового набора данных.

```
[28] for elem in prediction:
    predictionz = np.argmax(elem)
    print("Номер класса:", predictionz)
    print("Название класса:", classes[predictionz])

Номер класса: 1
Название класса: Potato_with_solanine
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 1
Название класса: Potato_with_solanine
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 1
Название класса: Potato_with_solanine
Номер класса: 1
Название класса: Potato_with_solanine
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 1
Название класса: Potato_with_solanine
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 1
Название класса: Potato_with_solanine
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 1
Название класса: Potato_with_solanine
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 0
Название класса: Healthy_potato
Номер класса: 1
Название класса: Potato_with_solanine
```

Рисунок 2.13.2 Распознавание изображений из тестового набора данных.

Видно как нейросеть распознает последовательно каждое изображение. Теперь выберем случайное одно изображение из тестового набора данных и

распознаем его, после чего выведем это изображение на экран для проверки.

```
Выберем случайное изображение и проверим

[29] data_sample = next(iter(test_dataset))
     sample_image = data_sample[0].numpy()[0]
     sample_label = classes[data_sample[1].numpy()[0]]
     prediction = np.argmax(model.predict(sample_image.reshape(-1, *sample_image.shape)))[0]
     print("Номер класса:", prediction)
     print("Название класса:", classes[prediction])

1/1 [=====] - 0s 158ms/step
Номер класса: 1
Название класса: Potato_with_solanine

plt.figure(figsize=(4, 4))
plt.imshow(data_sample[0].numpy()[0].astype("uint8"))
plt.title(classes[prediction])
plt.axis("off")

(-0.5, 99.5, 99.5, -0.5)
Potato_with_solanine
```

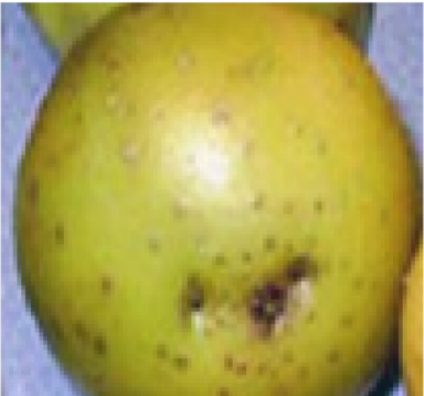


Рисунок 2.14 Распознавание случайного изображения.

Из результатов мы видим, что нейросеть справилась на отлично и определила правильно, так как на изображение и правда зеленый картофель с соланином.

2.5 Анализ результатов.

Чтобы проанализировать работу нейросети на нестандартных данных подготовим 2 тестовых датасета, состоящих из 20 изображений каждый:

1. Картофель без соланина с частичным его перекрытием зеленым листом. В качестве картофеля с соланином возьмем обычный картофель с соланином.
2. Картофель без соланина, покрытый грязью. В качестве картофеля с соланином возьмем обычный картофель с соланином.

Итоговый вид датасетов:

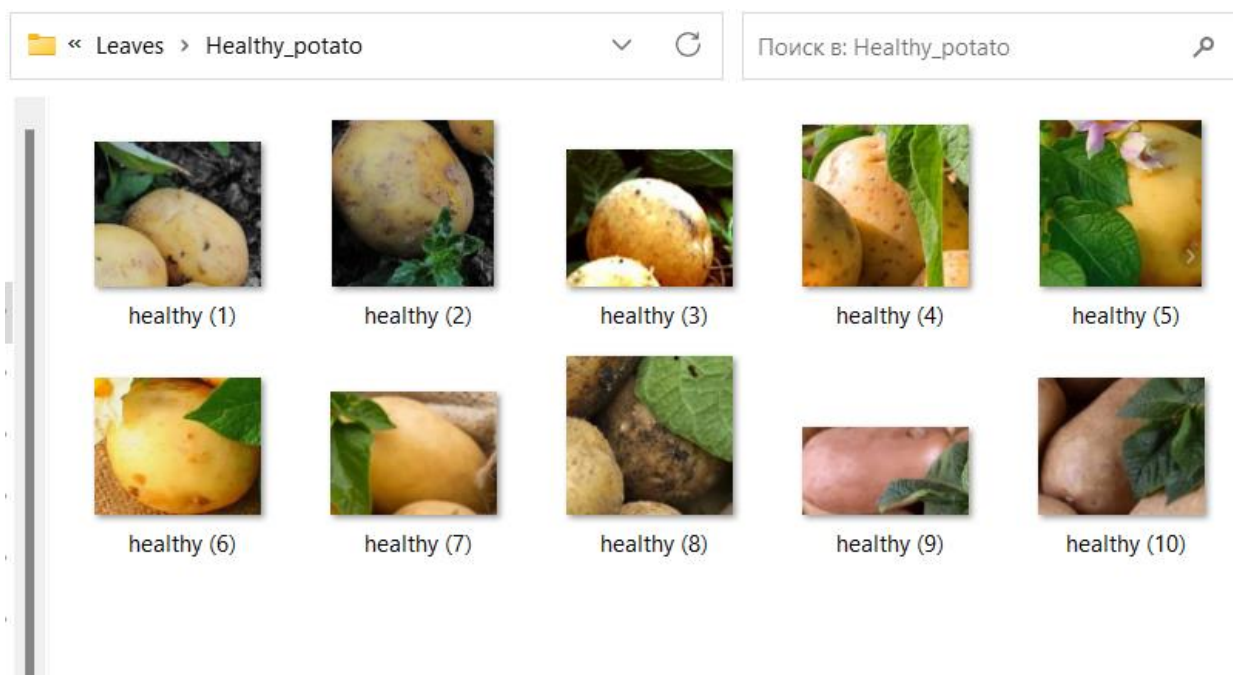


Рисунок 2.15 Вид датасета с перекрытием листвой.

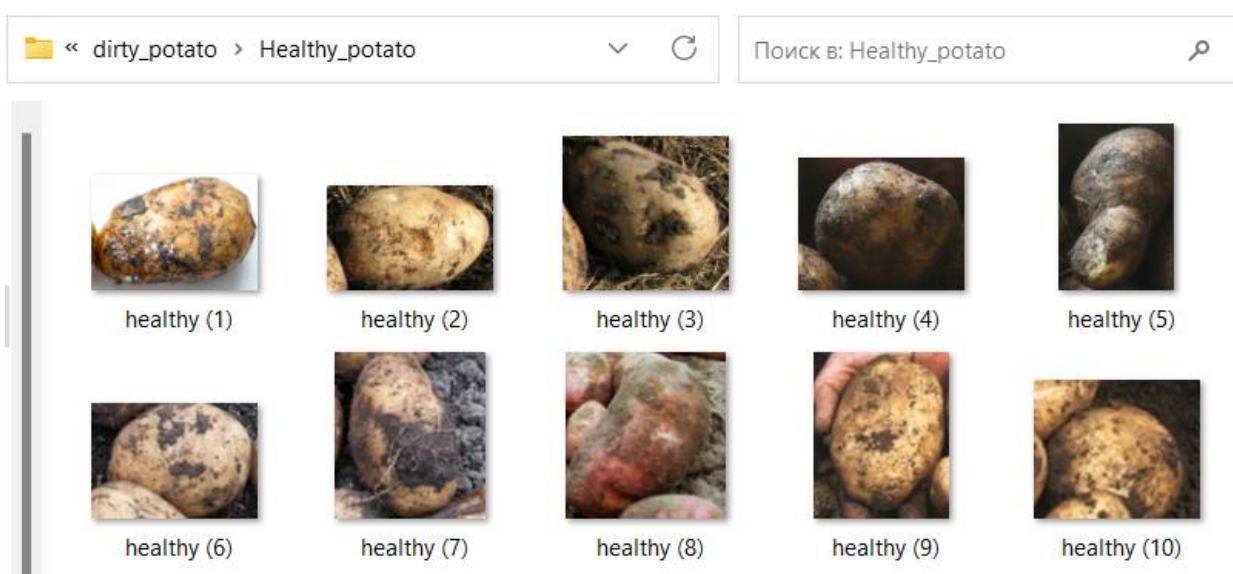


Рисунок 2.16 Вид датасета покрытого грязью картофеля.

Теперь оценим качество работы нейросети на данных созданных датасетов.

Оценка качества работы нейронной сети на разных условиях

Частичное перекрытие здорового растения листьями

```
[34] leaves_dataset = image_dataset_from_directory('leaves_test',
                                                    batch_size=2,
                                                    image_size=image_size)

Found 20 files belonging to 2 classes.

[35] #create_directory('leaves_test')

# Оцениваем качество обучения модели на тестовых данных
scores = model.evaluate(leaves_dataset, verbose=1)

10/10 [=====] - 1s 72ms/step - loss: 0.6303 - accuracy: 0.8500

[37] print("Доля верных ответов на тестовых данных, в процентах:", round(scores[1] * 100, 4))

Доля верных ответов на тестовых данных, в процентах: 85.0
```

Рисунок 2.15.1 Оценка качества на датасете с перекрытием листвой.

```
[38] plt.figure(figsize=(8, 8))
for i in range(9):
    data_sample = next(iter(leaves_dataset))
    sample_image = data_sample[0].numpy()[0]
    sample_label = classes[data_sample[1].numpy()[0]]
    prediction = np.argmax(model.predict(sample_image.reshape(-1, *sample_image.shape))[0])
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(data_sample[0].numpy()[0].astype("uint8"))
    plt.title(classes[prediction])
    plt.axis("off")
    #print("Номер класса:", prediction)
    #print("Название класса:", classes[prediction])

1/1 [=====] - 0s 64ms/step
1/1 [=====] - 0s 63ms/step
1/1 [=====] - 0s 65ms/step
1/1 [=====] - 0s 101ms/step
1/1 [=====] - 0s 112ms/step
1/1 [=====] - 0s 143ms/step
1/1 [=====] - 0s 97ms/step
1/1 [=====] - 0s 42ms/step
1/1 [=====] - 0s 44ms/step
```




Healthy_potato	Potato_with_solanine	Potato_with_solanine
		

Рисунок 2.15.2 Оценка качества на датасете с перекрытием листвой.

Оценка качества работы нейронной сети на разных условиях

Частичное перекрытие здорового растения листьями

```
[31] create_directory('dirt_test')

[33] dirt_dataset = image_dataset_from_directory('dirt_test',
                                                batch_size=2,
                                                image_size=image_size)

Found 20 files belonging to 2 classes.

[34] # Оцениваем качество обучения модели на тестовых данных
scores = model.evaluate(dirt_dataset, verbose=1)

10/10 [=====] - 1s 54ms/step - loss: 0.9090 - accuracy: 0.7500

[35] print("Доля верных ответов на тестовых данных, в процентах:", round(scores[1] * 100, 4))


Доля верных ответов на тестовых данных, в процентах: 75.0
```

Рисунок 2.16.1 Оценка качества на датасете с грязным картофелем.


```
plt.figure(figsize=(8, 8))
for i in range(9):
    data_sample = next(iter(dirt_dataset))
    sample_image = data_sample[0].numpy()[0]
    sample_label = classes[data_sample[1].numpy()[0]]
    prediction = np.argmax(model.predict(sample_image.reshape(-1, *sample_image.shape))[0])
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(data_sample[0].numpy()[0].astype("uint8"))
    plt.title(classes[prediction])
    plt.axis("off")

1/1 [=====] - 0s 66ms/step
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 68ms/step
1/1 [=====] - 0s 63ms/step
1/1 [=====] - 0s 55ms/step
1/1 [=====] - 0s 55ms/step
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 72ms/step
1/1 [=====] - 0s 65ms/step
```

Potato_with_solanine



Healthy_potato



Potato_with_solanine




Рисунок 2.16.2 Оценка качества на датасете с грязным картофелем.

Из результатов мы видим, что нейросеть на датасете с перекрытием листвой дает правильный ответ в 85% случаях, а на датасете с грязным картофелем дает верный ответ в 75% случаях.

Данный результат далек от идеала, но данную проблему можно решить, включив в датасет для обучения соответствующие фотографии.

Итого имеем:

1. Для правильных ответов на тестовом датасете: 88%
2. Для правильных ответов на датасете с перекрытием листвой: 85%
3. Для правильных ответов на датасете с грязным картофелем: 75%

Заключение

В результате проделанной работы был подготовлен набор данных, на которых была обучена нейросеть для распознавания картофеля с соланином и без. Для решения поставленной задачи была использована библиотека TensorFlow и ее пакет Keras. На выходе нейросеть показала достаточно неплохой результат в 88 процентов правильных ответов для своего относительно небольшого набора данных в 216 изображений картофеля с соланином и без. Была произведена проверка работы нейросети на примере.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Автор: Andrey Sozykin. Канал на YouTube курс “Программирование глубоких нейронных сетей”, URL: <https://www.youtube.com/@AndreySozykin>
2. Автор: К ВВ. Как создать классификатор изображений на Python с помощью Tensorflow 2 и Keras, URL: waksoft.susu.ru
3. Автор: Adrian Rosebrock, September 10, 2018. Keras Tutorial: How to get started with Keras, Deep Learning, and Python, URL: <https://www.reg.ru/blog/keras/amp/>
4. GitHub, URL: https://github.com/sozykin/dlpython_course
5. Автор: Evgenii Legotckoi, 13 мая 2020. Распознавание изображений на Python с помощью TensorFlow и Keras, URL: <https://evileg.com/ru/post/619/>
6. Автор: bredd_owen, 14 фев 2017. Создаём нейронную сеть InceptionV3 для распознавания изображений URL: <https://habr.com/ru/articles/321834/>

ПРИЛОЖЕНИЯ

Код работы.

```
"""Курсовая_Работа_Никитин_А.Д..ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1xF0YfF8HkeqP9-5Ma3vhAFsMmuGHHc0p
"""

# Commented out IPython magic to ensure Python compatibility.
import os
import numpy as np
import tensorflow as tf
```



```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,
Dropout
from tensorflow.keras import utils
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing import image_dataset_from_directory
import matplotlib.pyplot as plt
from google.colab import files
# %matplotlib inline

"""## Создаем Tensorflow Dataset'ы """

batch_size=16
image_size=(100, 100)

"""Подготовка изображений"""

import shutil
import os

# Каталог с набором данных
data_dir = 'data'
# Каталог с данными для обучения
train_dir = 'train'
# Каталог с данными для тестирования
test_dir = 'test'
# Часть набора данных для тестирования
test_data_portion = 0.15
# Часть набора данных для проверки
val_data_portion = 0.15
# Количество элементов данных в одном классе
nb_images = 108

def create_directory(dir_name):
    if os.path.exists(dir_name):
        shutil.rmtree(dir_name)
    os.makedirs(dir_name)
    os.makedirs(os.path.join(dir_name, "Potato_with_solanine"))
    os.makedirs(os.path.join(dir_name, "Healthy_potato"))

create_directory(train_dir)
create_directory(test_dir)

def copy_images(start_index, end_index, source_dir, dest_dir):
    for i in range(start_index, end_index):
        shutil.copy2(os.path.join(source_dir, "solanine (" + str(i) +
        ").png"),
                    os.path.join(dest_dir, "Potato_with_solanine"))
        shutil.copy2(os.path.join(source_dir, "healthy (" + str(i) +
        ").png"),
                    os.path.join(dest_dir, "Healthy_potato"))

start_test_data_idx = int(nb_images * (1 - test_data_portion))
print(start_test_data_idx)

copy_images(1, start_test_data_idx, data_dir, train_dir)
copy_images(start_test_data_idx, nb_images, data_dir, test_dir)

"""**Набор данных для обучения**"""

train_dataset = image_dataset_from_directory('train',
                                             subset='training',
                                             seed=42,

```

```

validation_split=0.1,
batch_size=batch_size,
image_size=image_size)

"""**Проверочный набор данных**"""

validation_dataset = image_dataset_from_directory('train',
                                                    subset='validation',
                                                    seed=42,
                                                    validation_split=0.1,
                                                    batch_size=batch_size,
                                                    image_size=image_size)

"""Названия классов в наборах данных."""

class_names = train_dataset.class_names
class_names

"""**Примеры изображений**"""

plt.figure(figsize=(8, 8))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")

"""**Набор данных для тестирования**"""

test_dataset = image_dataset_from_directory('test',
                                             batch_size=batch_size,
                                             image_size=image_size)

classes = test_dataset.class_names; classes

"""## Настраиваем производительность TensorFlow DataSet'ов"""

AUTOTUNE = tf.data.experimental.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

"""## Создаем нейронную сеть"""

# Создаем последовательную модель
model = Sequential()
# Сверточный слой
model.add(Conv2D(16, (5, 5), padding='same',
                 input_shape=(100, 100, 3), activation='relu'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(128, (5, 5), activation='relu', padding='same'))

```

```

# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Полносвязная часть нейронной сети для классификации
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
# Выходной слой, 131 нейрон по количеству классов
model.add(Dense(2, activation='softmax'))

"""**Компилируем модель**"""

model.compile(loss='sparse_categorical_crossentropy',
              optimizer="adam",
              metrics=['accuracy'],
              run_eagerly=True)

"""## Обучаем нейронную сеть"""

history = model.fit(train_dataset,
                    validation_data=validation_dataset,
                    epochs=15,
                    verbose=2)

"""## Оцениваем качество обучения сети"""

# Оцениваем качество обучения модели на тестовых данных
scores = model.evaluate(test_dataset, verbose=1)

print("Доля верных ответов на тестовых данных, в процентах:", round(scores[1]
* 100, 4))

plt.plot(history.history['accuracy'],
         label='Доля верных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'],
         label='Доля верных ответов на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Доля верных ответов')
plt.legend()
plt.show()

plt.plot(history.history['loss'],
         label='Ошибка на обучающем наборе')
plt.plot(history.history['val_loss'],
         label='Ошибка на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Ошибка')
plt.legend()
plt.show()

"""## Сохраняем обученную нейронную сеть"""

model.save("Potato_quality.h5")

files.download("Potato_quality.h5")

"""## Используем сеть для распознавания наличия соланина в клубне картофеля"""

prediction = model.predict(test_dataset)

prediction

```

```

for elem in prediction:
    predictionz = np.argmax(elem)
    print("Номер класса:", predictionz)
    print("Название класса:", classes[predictionz])

"""Выберем случайное изображение и проверим"""

data_sample = next(iter(test_dataset))
sample_image = data_sample[0].numpy()[0]
sample_label = classes[data_sample[1].numpy()[0]]
prediction = np.argmax(model.predict(sample_image.reshape(-1,
*sample_image.shape))[0])
print("Номер класса:", prediction)
print("Название класса:", classes[prediction])

plt.figure(figsize=(4, 4))
plt.imshow(data_sample[0].numpy()[0].astype("uint8"))
plt.title(classes[prediction])
plt.axis("off")

```