CZU: 004.43 DOI: 10.46727/c.steam-2023.p260-271

ВВЕДЕНИЕ В МАШИННОЕ ОБУЧЕНИЕ НА PYTHON: ОСНОВЫ И ПРАКТИКА

Olga CERBU, dr. conf. universitar
https://orcid.org/0000-0002-6278-7115
Tatiana SESTACOVA, dr. conf universitar
https://orcid.org/0000-0002-6995-4254
Nichita MIRONOV
Государственный Университет Молдовы, Факультет Информатики
Технический университет Молдовы

Резюме. В этой статье представлен способ работы с машинным обучением. Машинное обучение (Machine Learning, ML) - это подраздел искусственного интеллекта (ИИ), который занимается разработкой алгоритмов и моделей, способных извлекать знания и делать прогнозы на основе данных. В статье показан способ работы в области машинного обучения с помощью инструментов Google Colab, Jupyter Notebook с реализацией на языке Python.
Ключевые слова: Google Colab, Jupyter Notebook, Python, TensorFlow, Machine Learning.

Abstract. This article presents a way to work with machine learning. Machine Learning (ML) is a subfield of artificial intelligence (AI) that develops algorithms and models that can extract knowledge and make predictions based on data. The article shows a way to work in the field of machine learning using Google Colab and Jupyter Notebook tools with implementation in Python.
Keywords: Google Colab, Jupyter Notebook, Python, TensorFlow, Machine Learning.

Введение

Машинное обучение (Machine Learning, ML) - это подраздел искусственного интеллекта (ИИ), который занимается разработкой алгоритмов и моделей, способных извлекать знания и делать прогнозы на основе данных. Основная идея машинного обучения заключается в том, что компьютерные системы могут обучаться и совершенствовать свои навыки, используя опыт, без явного программирования.

Некоторые ключевые аспекты машинного обучения:

1. **Автоматическое обучение**: В машинном обучении алгоритмы способны обучаться на данных и делать предсказания, без явного программирования. Это означает, что системы могут адаптироваться к изменяющейся среде и обновлять свои модели на основе новых данных.
2. **Обработка больших объемов данных**: Машинное обучение позволяет эффективно анализировать и извлекать информацию из больших объемов данных, что было бы практически невозможно для человека.

3. **Прогнозы и классификация**: Машинное обучение применяется для создания моделей, которые способны делать прогнозы, классифицировать данные и принимать решения на основе имеющейся информации. Это может быть полезно во многих областях, от финансов до медицины.
4. **Поддержка принятия решений**: ML может помочь улучшить процесс принятия решений, предоставляя аналитику и информацию, основанную на данных, что особенно важно для бизнеса.
5. **Автоматизация задач**: Машинное обучение позволяет автоматизировать задачи, которые раньше требовали человеческого участия, что может значительно увеличить эффективность и снизить затраты.

Важные аспекты машинного обучения

1. **Обработка и анализ данных**: Современный мир генерирует огромные объемы данных, и машинное обучение позволяет извлекать ценную информацию из этой данных.
2. **Предсказания и оптимизация**: ML помогает в создании моделей для прогнозирования будущих событий и оптимизации процессов в различных областях, от производства до медицины.
3. **Персонализация**: Машинное обучение используется для создания персонализированных рекомендаций и услуг, что улучшает опыт пользователей.
4. **Автоматизация и автономия**: ML играет ключевую роль в развитии автономных систем и робототехники, позволяя им адаптироваться к окружающей среде.
5. **Инновации**: Машинное обучение способствует созданию новых продуктов и услуг, а также приводит к новым открытиям и исследованиям в различных областях науки.

Python является одним из наиболее популярных языков программирования для машинного обучения по ряду важных причин:

- **Простота и читаемость кода**: Python имеет простой и понятный синтаксис, который делает код более читаемым и понятным для разработчиков. Это позволяет исследователям данных и инженерам быстро создавать, тестировать и поддерживать модели машинного обучения.
- **Богатый экосистема библиотек**: Python обладает богатым набором библиотек и фреймворков для машинного обучения, таких как NumPy, Pandas, scikit-learn, TensorFlow и PyTorch. Эти библиотеки предоставляют множество инструментов и функций, упрощающих разработку и обучение моделей.
- **Активное сообщество и поддержка**: Python имеет огромное и активное сообщество разработчиков. Это означает, что всегда есть множество ресурсов,

форумов и обучающих материалов, готовых помочь новичкам и специалистам в области машинного обучения.

- **Поддержка для научных вычислений**: Python хорошо подходит для научных вычислений и анализа данных. Многие ученые и исследователи используют Python для обработки и анализа данных, что делает его естественным выбором для машинного обучения.
- **Кросс-платформенность**: Python поддерживается на большинстве операционных систем, что делает его универсальным языком для разработки машинных обучающих моделей, независимо от платформы.
- **Интеграция с другими языками**: Python легко интегрируется с другими языками программирования, что позволяет использовать специализированные библиотеки и инструменты, написанные на других языках, в экосистеме Python.
- **Продвинутые инструменты визуализации**: Библиотеки вроде Matplotlib и Seaborn позволяют разработчикам визуализировать данные и результаты моделей, что важно для анализа и интерпретации результатов машинного обучения.
- **Поддержка большинства областей машинного обучения**: Python подходит для широкого спектра задач машинного обучения, включая классификацию, регрессию, кластеризацию, обработку естественного языка и многое другое.

Все эти факторы делают Python привлекательным и эффективным выбором для разработки и исследования в области машинного обучения, и именно поэтому он так популярен среди специалистов в этой области.

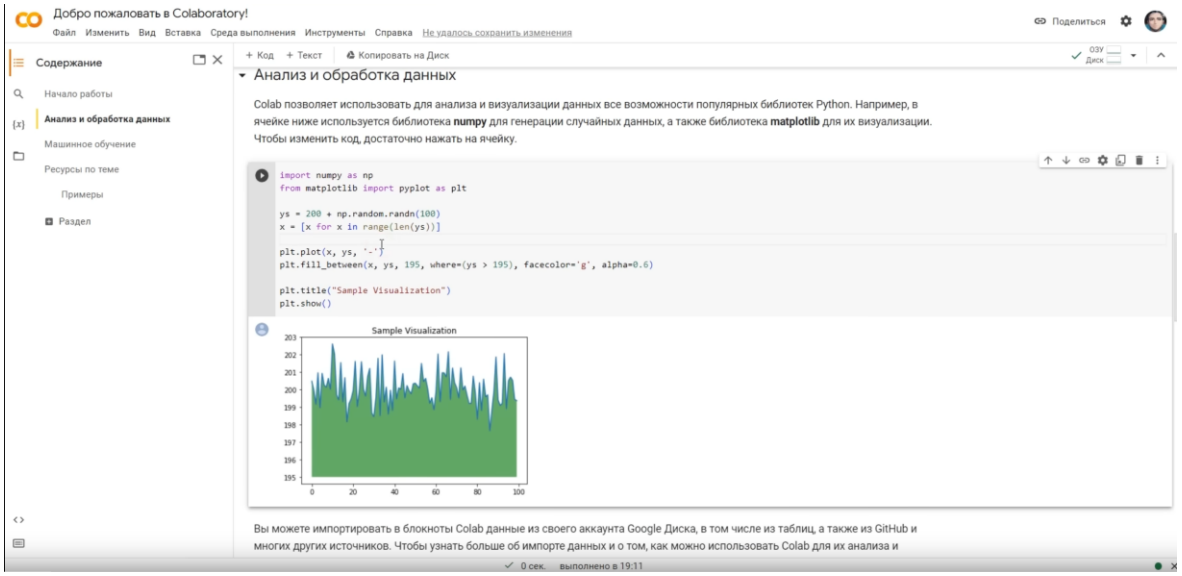
Среда Google Colab

Google Colab предоставляет доступ к среде Jupyter Notebook, а также к вычислительным ресурсам GPU и TPU. Вам не нужно устанавливать Python или библиотеки, так как они уже предустановлены.

Чтобы приступить к работе с выше перечисленными инструментами, необходимо следовать в следующим порядке:

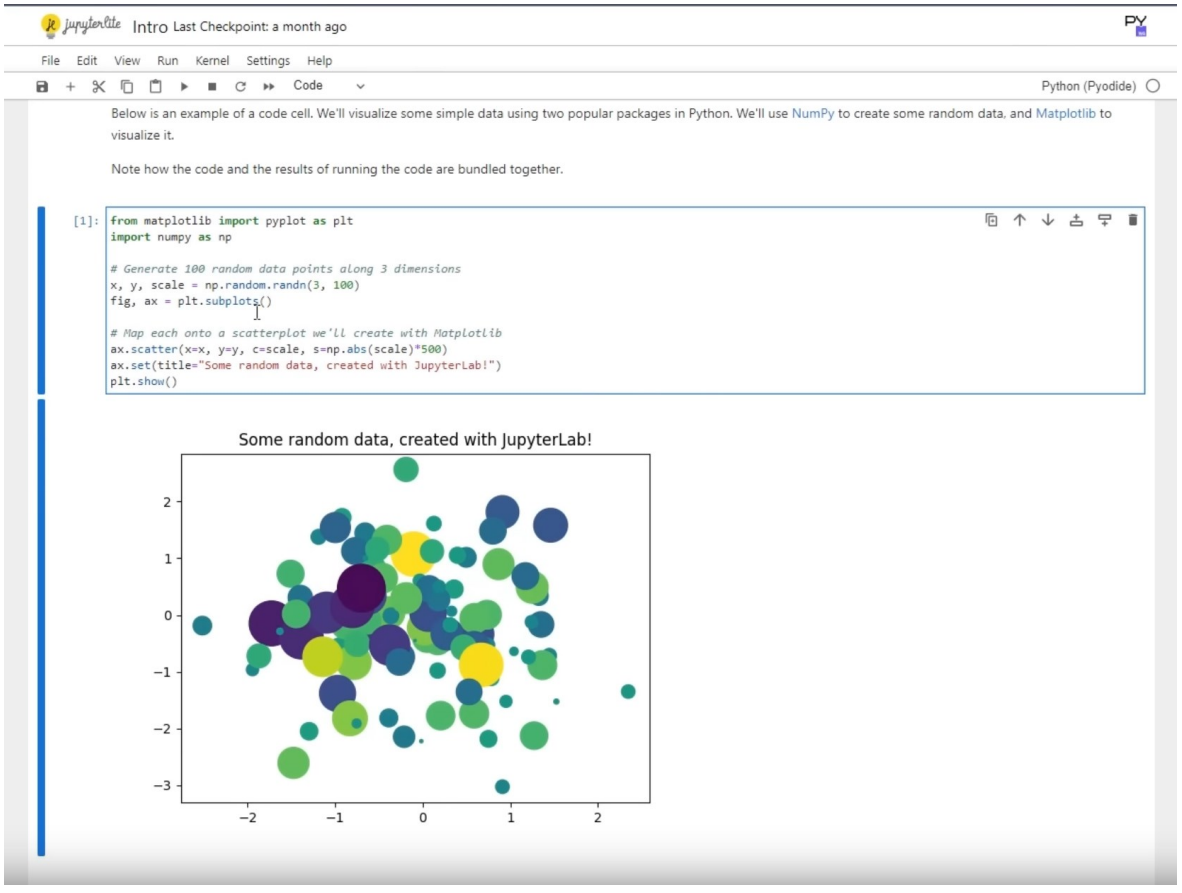
1. Перейдите на веб-сайт Google Colab
2. Войдите в свой аккаунт Google, если вы не вошли ранее.
3. Создайте новую «тетрадь» или загрузите существующие.

Затем вы можете начать писать код Python в ячейках тетради и запускать их. Google Colab также предоставляет доступ к файловой системе, что делает удобным импорт и экспорт данных и моделей.



Фигура 1. Пример проекта в Google Colab

Jupyter Notebook



Фигура 2. Среда программирования Jupyter

Jupyter Notebook - это отдельная среда для разработки, которую вы можете установить локально на своем компьютере. Чтобы начать работать с Jupyter Notebook:

1. Убедитесь, что у вас установлен Python. Вы можете проверить его, выполнив команду python --version или python3 --version.
2. Установите Jupyter Notebook, если у вас его еще нет. Это можно сделать с помощью pip, выполнив команду:

>pip install jupyter

3. Запустите Jupyter Notebook, выполнив команду:

>jupyter notebook

Ваш веб-браузер откроется с панелью управления Jupyter. Вы можете создать новую тетрадь или открыть существующую.

Классификация изображений при помощи TensorFlow

Распознавание названия цветка на основе изображений - это тип задачи классификации в машинном обучении, где модель должна определить, к какому виду цветка относится изображение. Это важная задача с множеством практических применений, от анализа растительного мира до разработки приложений для садоводов и ботаников.

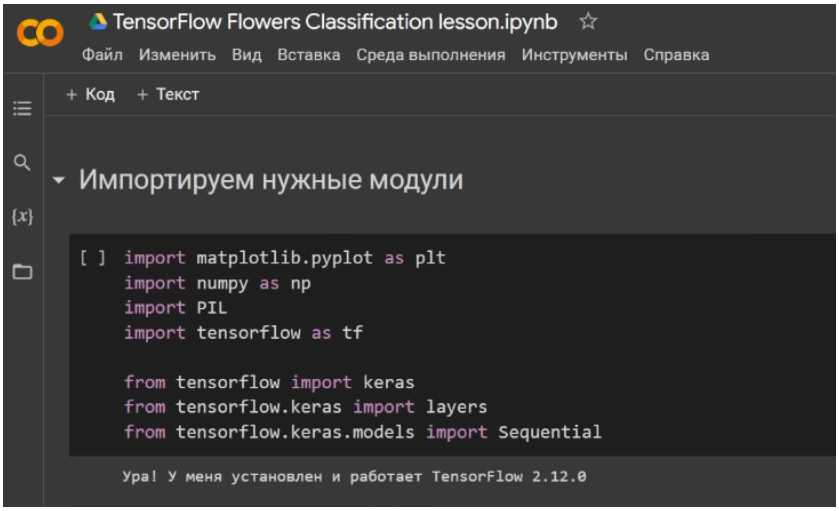
Для распознавания названия цветка на основе изображений мы будем использовать нейронные сети глубокого обучения (deep learning). В частности, мы будем использовать сверточные нейронные сети (Convolutional Neural Networks, CNN), которые специально разработаны для анализа изображений.

Подготовка данных

1. Для начала необходимо собрать набор данных, содержащий изображения цветков разных видов, а также метки, указывающие, к какому виду цветка относится каждое изображение. Вы можете использовать открытые источники данных, такие как "Flower Recognition Dataset" или же взять готовый пример в интернете

Фигура 3. Пример набора данных

2. В Google Colab создаем новый файл для работы и импортируем наш набор данных.



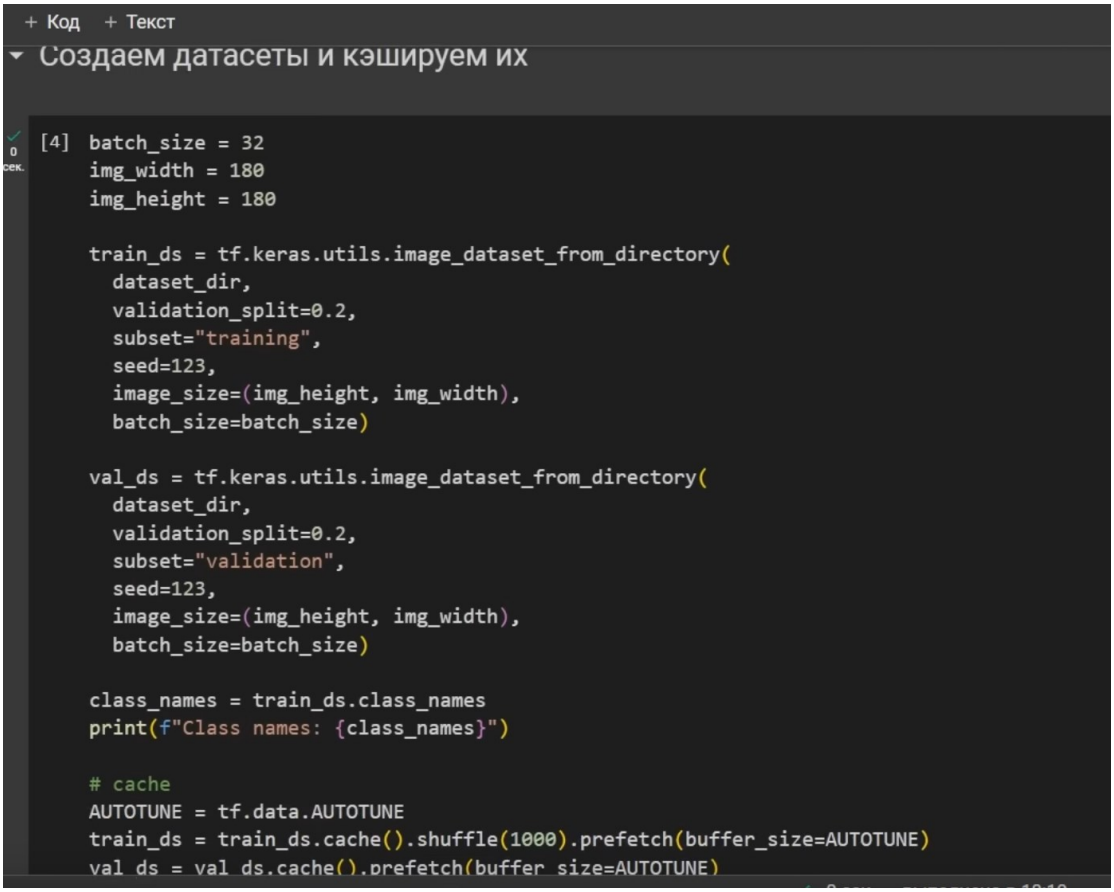
Фигура 4. Инициация проекта в Google Colab

Фигура 5. Инструкция по импортированию датасетов

3. Далее необходимо разделить данные на обучающий и тестовый наборы. Обычно 80% данных используется для обучения, а 20% для тестирования.

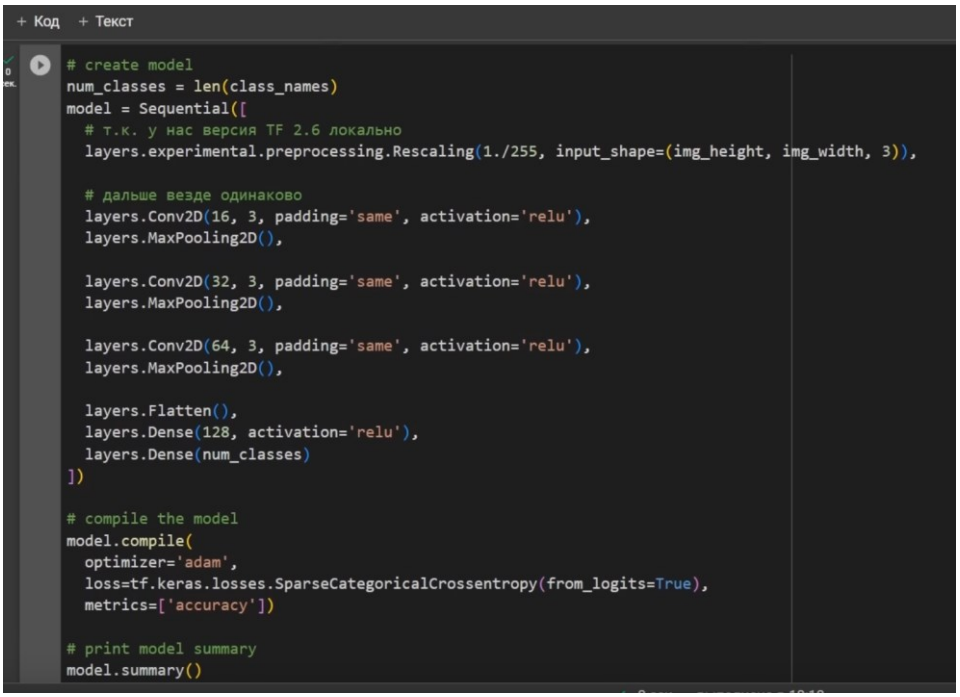
TensorFlow рекомендует перед началом машинного обучение совершить кэширование набора данных.

И для этого тут также есть готовые утилиты и функции. Потом нужно разметить структуру модели и для нашей текущей задачи хорошо подойдет последовательная архитектура нейронной сети, состоящая из трех блоков свертки с максимальным объединяющим слоем в каждом из них и с полно связанным слоем на 128 единиц поверх этого всего вместе с нелинейной активационной функции *relu*.

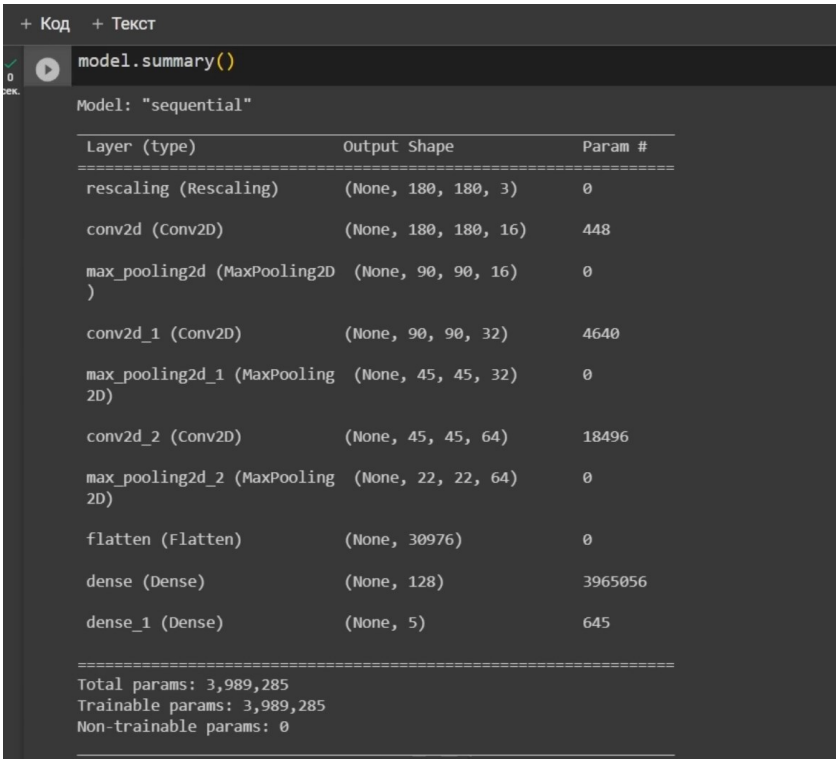


Фигура 6. Инструкция по кэшированию набора данных

4. Далее, создаем модель, компилируем её и выводим результаты.

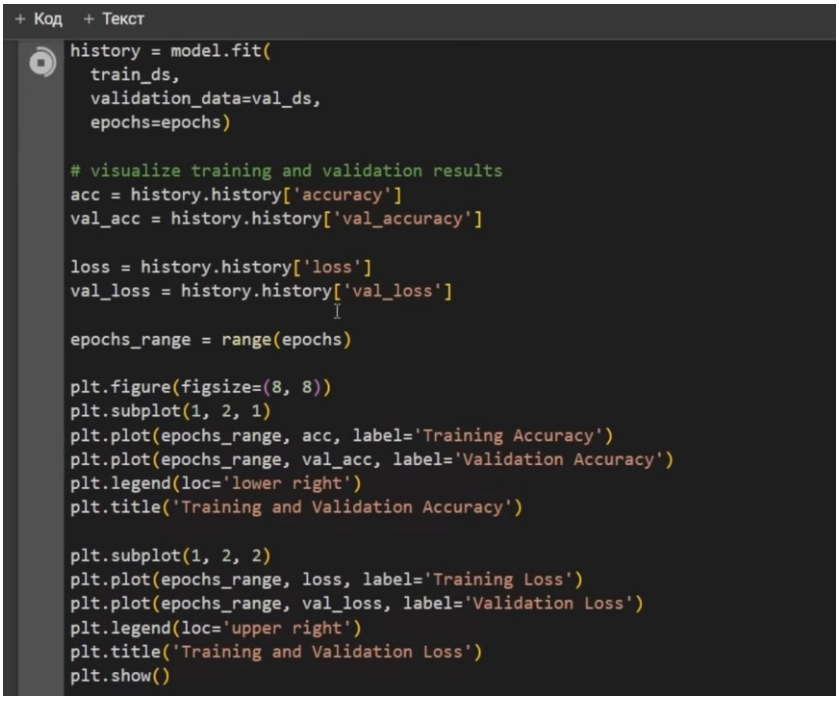


Фигура 7. Пример кода для вывода компилируемых результатов



Фигура 8. Вывод результата компилирования

И теперь мы уже можем начать обучение нашей нейронной сети для этого нужно указать количество эпох обучения и вызвать метод fit у нашей модели.



Фигура 9. Пример программы машинного обучения

Код с документации по TensorFlow который использует библиотеку Matplotlib позволит нам совершить статистическую оценку данных. А после обучения

вывести наглядный статистические данные в виде графиков отображающий два важных показателя: Точность (Accuracy) и потери (Loss)

```
Epoch 1/10
92/92 [==============================] - 133s 1s/step - loss: 1.4290 - accuracy: 0.3832 - val_loss: 1.1002 - val_accuracy: 0.5422
Epoch 2/10
92/92 [==============================] - 119s 1s/step - loss: 1.0710 - accuracy: 0.5661 - val_loss: 1.0545 - val_accuracy: 0.5695
Epoch 3/10
92/92 [==============================] - 119s 1s/step - loss: 0.8900 - accuracy: 0.6567 - val_loss: 0.9460 - val_accuracy: 0.6267
Epoch 4/10
92/92 [==============================] - 117s 1s/step - loss: 0.7224 - accuracy: 0.7285 - val_loss: 0.9375 - val_accuracy: 0.6349
Epoch 5/10
92/92 [==============================] - 120s 1s/step - loss: 0.5300 - accuracy: 0.8007 - val_loss: 0.8824 - val_accuracy: 0.6785
Epoch 6/10
92/92 [==============================] - 125s 1s/step - loss: 0.3284 - accuracy: 0.8893 - val_loss: 0.9843 - val_accuracy: 0.6771
Epoch 7/10
92/92 [==============================] - 120s 1s/step - loss: 0.1871 - accuracy: 0.9414 - val_loss: 1.4537 - val_accuracy: 0.6185
Epoch 8/10
92/92 [==============================] - 122s 1s/step - loss: 0.1261 - accuracy: 0.9615 - val_loss: 1.4179 - val_accuracy: 0.6267
Epoch 9/10
92/92 [==============================] - 115s 1s/step - loss: 0.0686 - accuracy: 0.9799 - val_loss: 1.7592 - val_accuracy: 0.6362
```

Фигура 10. Вывод статистических данных

Уровень отклонённых данных - это способ измерения того насколько далеко предсказуемое значение находится от желаемого выходного значения. Чем выше показатель потерь тем хуже нейросеть справляется со своей задачей.

Между эпохами обучения эти значения будут меняться, а график позволит нам увидеть как именно они меняются:

Фигура 11. График отображающий точность данных и уровень отклонения

Этот график наглядно демонстрирует проблему которая в нейросетях называется переобучение или же Overfitting. Это ситуация, когда нейросеть настолько долго обучается на тренировочном датасете, что начинает правильно распознавать только изображение только из самого датасета.

Мы видим что в левом графике показатель Training Accuracy только растет, в то время как Validation Accuracy где-то с четвёртой-пятой эпохи стагнирует. Также посмотрим на правый график, мы видим что Training Loss стремительно падает в то время как Validation растёт.

То есть нейросеть к десятой эпохе подходит к тому, что начинает очень точно распознавать тренировочный датасет и плохо распознавать валидационный. При этом очень сильно растут потери в валидационном датасете, у тренировочного потери только падают. Это и есть **переобучение.**

Один из наиболее эффективных способов борьбы с переобучением - это увеличение размера обучающей выборки. Чем больше данных у вас есть, тем лучше модель может обобщать. Если нет возможности собрать больше данных, можно воспользоваться аугментацией данных, что позволяет создавать дополнительные обучающие примеры путем их трансформации.

Добавление **регуляризации** к модели помогает уменьшить переобучение. L1 и L2 регуляризация добавляют штрафы за большие веса модели. Это может помочь уменьшить чрезмерное настроение на обучающие данные.

Аугментация - это приём который позволяет дополнить набор данных в несколько раз путем несложных манипуляций с изображениями: поворачивают, отражают, искажают картинки в наборе данных таким образом что в итоге из одного изображения можно получить сразу 5-7 штук, а набор данных на 3 тысячи изображений растянуть до 15-20 тысяч.

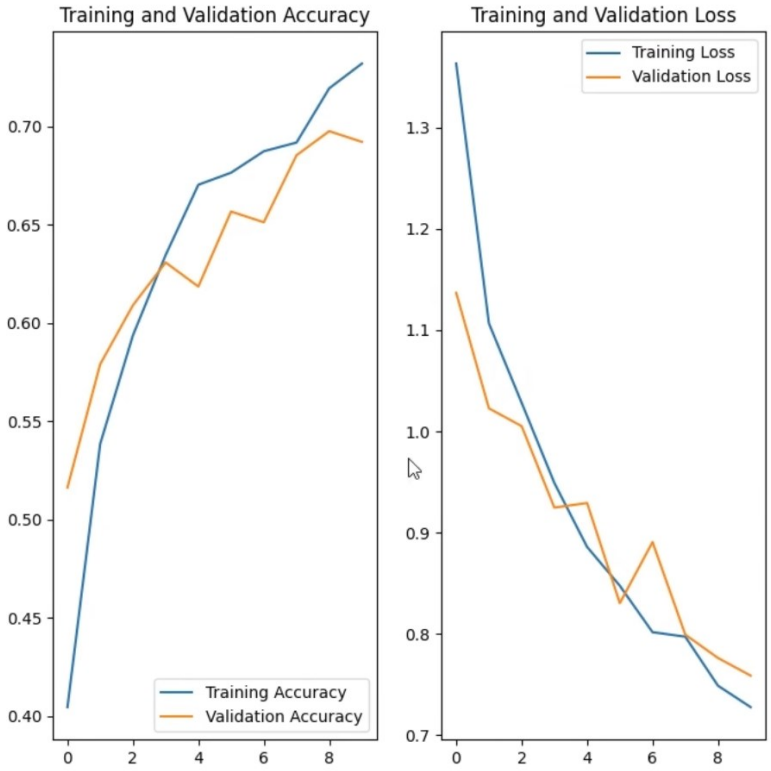
Фигура 12. пример аугментации данных

Применив ниже перечисленные методы, можем достичь аугментации данных, с разбросом данных в несколько тысяч эллементов.

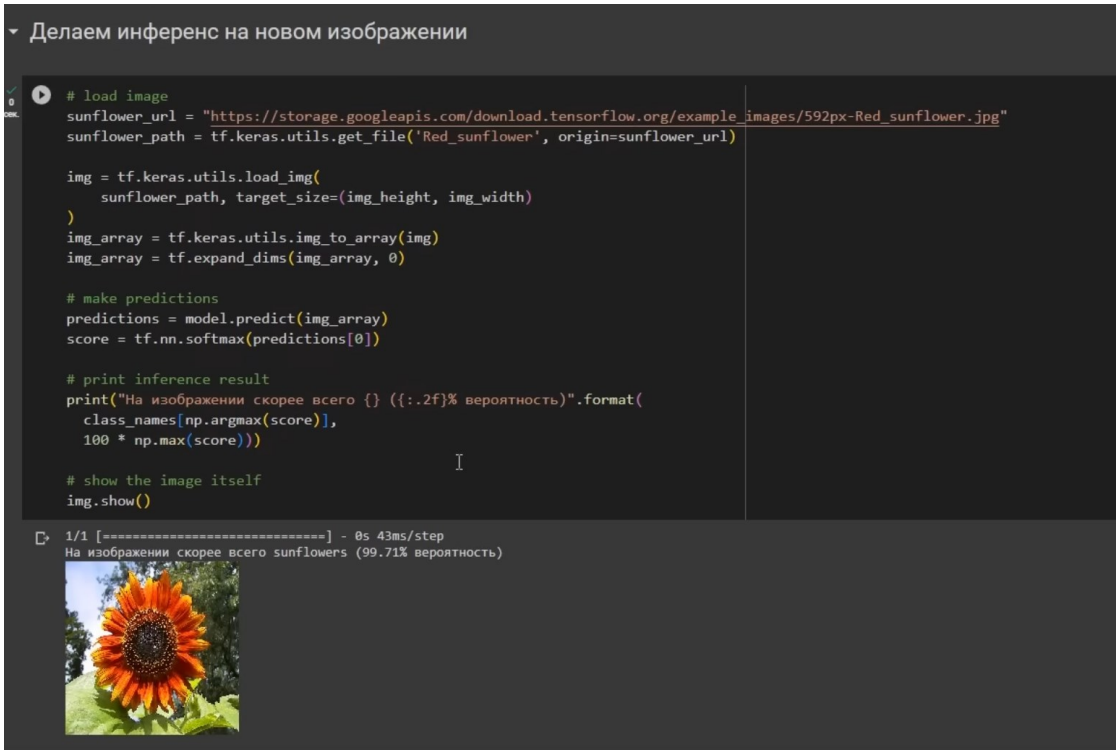
```
# аугментация
layers.experimental.preprocessing.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),
layers.experimental.preprocessing.RandomRotation(0.1),
layers.experimental.preprocessing.RandomZoom(0.1),
layers.experimental.preprocessing.RandomContrast(0.2),
```

Фигура 13. Методы используемые для аугментации данных

Можем заметить что после процесса аугментации данных, выводимые в график результаты значительно улучшились.



Фигура 14. Графический вывод данных после совершения аугментации набора данных



Фигура 15. Пример внедрения финального кода с учётом процесса аугментации набора данных

Заключение

Таким образом, мы написали пример создания и обучения модели нейронной сети с использованием TensorFlow для классификации изображений цветов. Наша модель включает в себя несколько важных компонентов, таких как аугментация данных, регуляризация и обработка изображений.

Выводы о проделанной работе:

Аугментация данных: использовали аугментацию данных, такую как случайное отражение, поворот, увеличение масштаба и контраста. Это помогает модели обучаться на более разнообразных данных, что может повысить ее способность обобщать.

Архитектура модели: модель состоит из сверточных слоев, слоев подвыборки и полносвязных слоев. Эта архитектура обеспечивает хороший баланс между изучением признаков на разных уровнях и обобщением.

Регуляризация: применили регуляризацию с помощью слоя Dropout, что помогает предотвратить переобучение и улучшить производительность модели на новых данных.

Компиляция и обучение модели: использовали оптимизатор Adam и функцию потерь Sparse Categorical Crossentropy.

Интерпретация результатов: После обучения модели, мы загрузили изображение и получили прогнозы от модели. Модель определила класс изображения (в данном случае, вид цветка) с вероятностью. Это важный шаг для понимания того, как работает модель и какие выводы она делает.

Отображение изображения: В завершение, отобразили само изображение для наглядности.

Библиография

1. https://www.tensorflow.org/
2. Курс по машинному обучению
3. https://www.deeplearningbook.org/
4. https://github.com/tensorflow/tensorflow
5. https://colab.research.google.com/
6. https://jupyter.org/
7. https://realpython.com/tutorials/machine-learning/