# Машинное обучение на C# ML.NET

ML.NET позволяет добавлять в приложения .NET возможности машинного обучения в автономном и подключенном режимах. Используя эту функцию, вы сможете получать автоматические прогнозы на основе данных, доступных вашему приложению. Приложения машинного обучения используют для прогнозирования закономерности, найденные в данных, не будучи явно запрограммированными.

В основе ML.NET лежит модель машинного обучения. Эта модель определяет шаги, которые необходимо выполнить для получения прогнозов на основе входных данных. С помощью ML.NET вы можете обучить пользовательскую модель, указав соответствующий алгоритм, а также импортировать предварительно обученные модели TensorFlow и ONNX.

Созданную модель можно добавить в приложение и использовать ее для получения прогнозов.

ML.NET работает в Windows, Linux и macOS с .NET Core или в Windows с .NET Framework. 64-разрядная версия поддерживается на всех платформах. 32-разрядная версия поддерживается в Windows, за исключением функций, связанных с TensorFlow, LightGBM и ONNX.

С помощью ML.NET можно получать прогнозы следующих типов.

**Классификация/категоризация**

Автоматическое разделение отзывов клиентов на положительные и отрицательные

**Регрессия/прогнозирование непрерывных значений**

Прогнозирование стоимости домов на основе размера и расположения

**обнаружение аномалий;**

Обнаружение мошеннических банковских транзакций

**Рекомендации**

Предложение продуктов, которые могут понравиться покупателям Интернет-магазина, на основе их предыдущих покупок

**Временные ряды/последовательности**

Прогнозы погоды и объемов продаж

**Классификация изображений**

Классификация патологий на медицинских изображениях

**Классификация текста**

Классифицируйте документы по их содержимому.

**Сходство предложений**

Измерьте, насколько похожи два предложения.

# Используемые библиотеки

1. System.Drawing;
2. System.Drawing.Drawing2D;
3. ObjectDetection.YoloParser;
4. ObjectDetection.DataStructures;

# Принцип работы

[Класс MLContext](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.ml.mlcontext) является отправной точкой для любых операций ML.NET. В результате инициализации класса mlContext создается среда ML.NET, которая может использоваться всеми объектами в рамках процесса создания модели.

Модель разделяет изображение на сетку 13 x 13, где каждая ячейка сетки — это 32px x 32px. Каждая ячейка сетки содержит 5 возможных ограничивающих прямоугольников объекта. Ограничивающий прямоугольник содержит 25 элементов:

• x — координата по оси X для центра ограничивающего прямоугольника относительно ячейки сетки, с которой он связан.

• y — координата по оси Y для центра ограничивающего прямоугольника относительно ячейки сетки, с которой он связан.

• w — ширина ограничивающего прямоугольника.

• h — высота ограничивающего прямоугольника.

• o — значение достоверности того, что объект существует в пределах ограничивающего прямоугольника, также известный как оценка объекта.

• p1-p20 — вероятности для каждого из 20 классов, прогнозируемых моделью.

В итоге 25 элементов, описывающих каждый из пяти ограничивающих прямоугольников, составляют 125 элементов, содержащихся в каждой ячейке сетки.

Выходные данные, формируемые предварительно обученной моделью ONNX, представляют собой массив длины 21125, представляющий тензорные элементы с 125 x 13 x 13 измерениями. Чтобы преобразовать прогнозы, созданные моделью, в тензоры, необходимо выполнить некоторые действия для постобработки. Для этого создайте набор классов для упрощения анализа выходных данных.

# Описание проекта

# Проект консольного приложения ObjectDetectionможно использовать для идентификации объектов в образцах изображений на основе модели Tiny Yolo2 ONNX .

# Используется только предварительно обученная модель ONNX с API ML.NET. Поэтому он не обучает какую-либо модель ML.NET. В настоящее время ML.NET поддерживает только оценку/обнаружение с помощью существующих обученных моделей ONNX.

# ML.NET: настройка модели

# файл изображения, используемый для загрузки изображений, имеет два столбца: первый определяется как, ImagePathа второй Labelсоответствует изображению.

# Важно подчеркнуть, что класс Labelв ImageNetDataдействительности не используется при подсчете очков с помощью модели Tiny Yolo2 Onnx. Используется при печати этикеток на консоли.

# Второй шаг — определить конвейер оценщика. Обычно при работе с глубокими нейронными сетями вы должны адаптировать изображения к формату, ожидаемому сетью. По этой причине размеры изображений изменяются, а затем трансформируются (в основном значения пикселей нормализуются по всем каналам R, G, B).

# 

# Также нужно проверить нейросеть, и проверить имена входных/выходных узлов. Для проверки модели можно использовать такие инструменты, как [Netron](https://github.com/lutzroeder/netron) , который автоматически устанавливается вместе с [Visual Studio Tools for AI](https://visualstudio.microsoft.com/downloads/ai-tools-vs/) . Эти имена используются позже в определении конвейера оценки: в случае начальной сети входной тензор называется «изображение», а выходной — «сетка».

# Обнаружение объектов на изображении:

# После того, как модель настроена, нам нужно передать изображение в модель для обнаружения объектов. При получении прогноза мы получаем массив поплавков в свойстве PredictedLabels. Массив представляет собой массив с плавающей запятой размером **21125** . Это выход модели i,e 125x13x13, как обсуждалось ранее. Этот вывод интерпретируется YoloOutputParserклассом и возвращает количество ограничительных рамок для каждого изображения. Опять же, эти прямоугольники фильтруются, так что мы извлекаем только 5 ограничивающих прямоугольников, которые имеют большую достоверность (насколько уверены, что прямоугольник содержит объект) для каждого объекта изображения.

# Результаты

