Проект System.AI: полностью управляемый стек машинного обучения и анализа данных для экосистемы .NET

Г.С. Брыкин

Московский Государственный Технический Университет им. Н.Э. Баумана,
г. Москва, Россия

Аннотация. В последние годы технологии машинного обучения становятся всё более распространёнными в таких известных задачах как стилизация изображений, окрашивание чёрно-белых изображений, супер-разрешение изображений, поиск поддельных данных, распознавание голоса и изображений. В связи с этим возникает необходимость в реализации набора инструментов для интеграции систем искусственного интеллекта в приложения для мобильных устройств, устройств умного дома и домашних ПК. Статья посвящена решению, позволяющему разработчикам интегрировать системы анализа данных и искусственного интеллекта непосредственно в приложение, что позволит получить легковесный, портативный, кроссплатформенный монолитный программный продукт, что зачастую невозможно с использованием существующих решений. Основными особенностями предлагаемого решения являются нацеленность на экосистему Microsoft .NET [1], а также использование только стандартных возможностей BCL и языка C#. Реализованный пакет инструментов является исключительно кроссплатформенным и аппаратнонезависимым. API во многом совпадает с аналогичными решениями для Python, что позволяет быстро перенести коды на Python в проект для .NET.

Ключевые слова: *машинное обучение, Анализ данных, .NET Framework, Mono [2], Xamarin [3], .NET Core [1], .NET Standard [1], Управляемый код.*

DOI: 10.14357/20790279230108

Введение

Платформа Microsoft .NET Framework и ее кроссплатформенные реализации, такие как Mono, .NET Core и Xamarin, предлагают разработчику широкий спектр инструментов для разработки мобильных, настольных и сетевых приложений; баз данных. Основной задачей проекта System.AI является внедрение в экосистему .NET стека технологий машинного обучения и анализа данных с сохранением при этом возможностей и синтаксиса аналогичных решений для Python или JavaScript, если это возможно. System.AI на данный момент включает в себя реализации библиотек *imageio* [4], *PyTorch* [5] и *convnet.js* [6], а также множество расширений стандартных классов .NET, которые обеспечивают гибкое взаимодействие между *System.AI* и .NET. Основными отличиями от предыдущих работ являются минимализм используемых инструментов из стандартного пакета .NET, язык C# 5, и полностью управляемый код, что делает *System.AI* по-настоящему кроссплатформенным программным обеспечением.

1. Предыдущие работы

Решения для **Python (PyTorch, TensorFlow, etc.).** Наиболее популярными фреймворками машинного обучения являются *PyTorch* от FaceBook и *TensorFlow* от Google [7]. Эти библиотеки для языка программирования Python обладают обширной функциональностью для реализации нейронных сетей для любой задачи, однако специфика языка Python крайне затрудняет его использование при разработке мобильных и настольных приложений. В частности, среда выполнения Python в настоящее время не имеет официальной поддержки для известной мобильной операционной системы Android. Кроме того, флагманские фреймворки машинного обучения не поддерживают 32-разрядные операционные системы, по-прежнему часто

использующиеся на домашних ПК. Наконец, интеграция среды выполнения Python, базовых библиотек и библиотек машинного обучения в конечный продукт значительно увеличивает объём занимаемого дискового пространства, что критично для мобильных.

convnet.js и ConvNetCS. *convnet.js* – это библиотека для обучения свёрточных нейронных сетей непосредственно в браузере, написанная на JavaScript. Фактически, это первая широко известная библиотека машинного обучения, предназначенная для использования на широком спектре устройств. Преимущества включают кроссплатформенность и совместимость со многими браузерами и операционными системами. Однако эта библиотека предназначена для демонстрационных целей и обладает довольно узким функционалом, которого недостаточно для реализации современных нейронных сетей. Кроме того, JavaScript является скриптовым языком для веб-сайтов и редко используется для разработки приложений.

ConvNetCS [8] – это порт *convnet.js* на C#. Структура библиотеки позволяет использовать её в настольных приложениях, но ограниченная функциональность делает это бессмысленным.

Tensorflow.js (Deeplearn.js). Представляет собой библиотеку машинного обучения и линейной алгебры для запуска в браузере [9]. Она использует высокооптимизированные алгоритмы и поддерживает вычисления на графическом процессоре. Функционал достаточен для разработки и запуска практически всех современных нейронных сетей. Недостатками являются зависимость от версии браузера и невозможность использования при разработке приложений.

AlbiruniML. Является *Tensorflow.js* - подобной библиотекой машинного обучения и линейной алгебры с поддержкой автоматического дифференцирования [10]. Позиционируется автором как реализация *Tensorflow.js* для .NET и является кроссплатформенным программным обеспечением. Библиотека обладает достаточным функционалом для полноценной работы с нейронными сетями, но очень низким уровнем оптимизации алгоритмов. Последняя версия библиотеки была выпущена в 2018 году.

SciSharp Stack. Это готовый к использованию высокооптимизированный стек машинного обучения и анализа данных для экосистемы .NET [11]. Он написан в основном на C# с использованием кроссплатформенных возможностей BCL, но многие из наиболее важных библиотек в *SciSharp* используют неуправляемый код (неуправляемые .dll или файлы .so), например *TensorFlow.NET*, или вызывают коды на Python, например *Torch.NET* или *Numpy.NET*. Многие библиотеки в пакете фактически являются API, но не реализациями. Это не устраняет трудностей, связанных с использованием классических фреймворков машинного обучения.

Demos. Суть проекта заключается в реализации популярных нейронных алгоритмов для окрашивания, стилизации и супер-разрешения изображений на чистом C# [12]. Каждый алгоритм в предлагаемом проекте представлен в виде самостоятельного приложения для операционной системы Windows с удобным графическим интерфейсом. Цель проекта - предоставить возможность быстро запускать нейронные алгоритмы на настольных компьютерах. Проект считается предыдущим этапом работы над проектом *System.AI*.

MyCaffe. MyCaffe [13] – реализация библиотеки машинного обучения *Caffe* [14] на C#. Это кроссплатформенное и хорошо оптимизированное программное обеспечение, поддерживающее, однако, вычисления только на GPU от NVIDIA. Архитектура *MyCaffe* идеально подходит для использования в составе классических приложений на .NET, однако невозможность использования центрального процессора, а также ограничение на поддерживаемые графические процессоры существенно сужают круг поддерживаемых устройств.

2. Предлагаемое решение

Обзор структуры**.** *System.AI* - это набор взаимосвязанных библиотек. Все библиотеки из состава *System.AI* можно разделить на 3 глобальных типа: библиотеки ввода-вывода, предназначенные, соответственно, для обеспечения ввода и вывода данных различной природы; библиотеки расширений, реализующие отсутствующие в .NET типы данных (такие как, например, *Half*, *Quarter*, комплексные числа различной точности и т.д.), и обеспечивающие взаимодействие между типами *System.AI* и стандартными типами .NET, а также содержащие методы расширения для классов BCL. Например, библиотека *DotnetExtensions* реализует метод *tobytes()* для стандартных массивов. Это позволяет использовать подобный *NumPy* синтаксис для сохранения массивов в двоичной форме. Последний класс библиотек *System.AI* - это библиотеки машинного обучения, предназначенные непосредственно для создания, обучения и запуска нейронных сетей.

Для реализации всех компонентов *System.AI* был использован язык программирования C# 5. Это позволяет использовать консольный компиля-

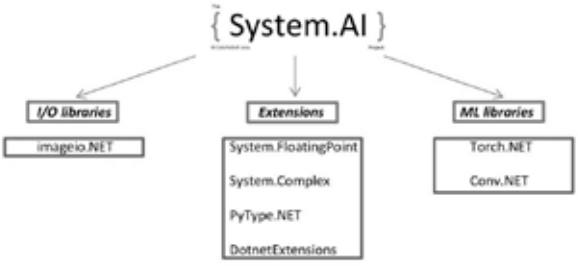

Рис. 1. Структура System.AI

тор C#, встроенный в Microsoft .NET Framework в ОС Windows, без необходимости установки стороннего программного обеспечения. Учитывая то, что требуемая минимальная версия (4.0) Microsoft .NET Framework является частью операционной системы, начиная с Windows 8, а также может быть установлена в операционных системах Windows XP, Windows Vista и Windows 7, мы можем предположить, что *System.AI* может быть скомпилирован из исходных кодов и использован с использованием стандартных инструментов самой распространённой настольной ОС – Windows. Кроме того, *System.AI* использует ограниченный набор исключительно кроссплатформенных возможностей BCL (Библиотеки базовых классов .NET), таких как многопоточность через *System.Threading.Tasks*, вывод на консоль через *System.Console* и работа с файлами и потоками через *System.IO*. Это позволяет запускать приложения с *System.AI* на любом устройстве, поддерживающем Microsoft .NET Framework или его реализацию, независимо от операционной системы или архитектуры процессора. Трассировки кода, выполняемого из *System.AI* и аналогичных решений представлены на рис. 2. Мы можем видеть, что код, основанный на *System.AI*, напрямую взаимодействует с операционной системой и аппаратным обеспечением, в то время как использование аналогичных решений предполагает один или несколько дополнительных уровней. Среда выполнения .NET содержит JIT-компилятор, который генерирует высокопроизводительный код для текущей архитектуры процессора. Таким образом, повышается производительность системы. IL-код *System.AI* и IL-код приложения почти так же эффективны, как машинный код. В настоящее время *System.AI* успешно собран и запущен на MIUI (Android) и облачной платформе Google Colab [15] (Ubuntu) с помощью Mono, а также на Windows (7, 10) с помощью .NET Framework.

В табл. 1 показано сравнение *System.AI* с аналогичными библиотеками.

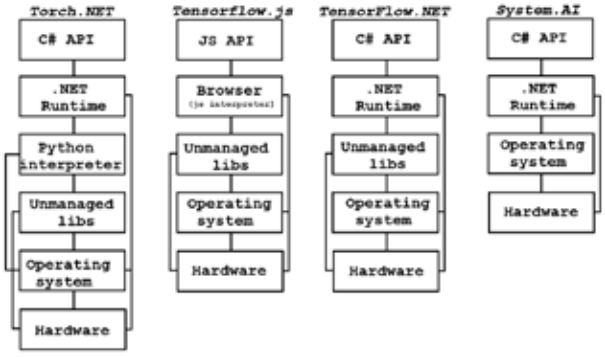

Рис. 2. Трассировка кода *System.AI* и предшествующих решений

imageio.NET. *imageio.NET* - это библиотека для ввода/вывода изображений. Она позволяет читать файлы в форматах bmp, gif, jpeg, png, psd и tga, возвращая изображение в виде трехмерного массива байт. Кроме того, библиотека позволяет сохранять изображение, представленное трехмерным массивом простого числового типа данных, в формах bmp, jpg, png, hdr и tga. В качестве ядра используются переработанные библиотеки *StbImageSharp* и *StbImageWriteSharp*. *imageio.NET* позволяет выполнять обработку изображений на любом устройстве, поддерживающем .NET или любую из его реализаций. Пример кода на PascalABC.NET для чтения изображения из файла в массив приведен ниже.

Табл. 1

Различия между *System.AI* и другими решениями. **Ведётся работа*.

	convnet.js	*ConvNetCS*	*tensorflow.js*	*AlbiruniML*	*SciSharp*	***System.AI***
Оптимизирован	Ч	Ч	√	Ч	√	√
Кросс-платформенность	√	√	~	√	~	√
Аппаратная независимость	√	√	√	√	~	√
Готов к использованию	Ч	Ч	√	Ч	√	**~***
Для приложений	Ч	√	Ч	√	√	√

```
{$reference 'imageio.dll'}
uses System;
uses System.IO;
begin
  Console.WriteLine(imageio.__version__);  // current imageio.NET version
  var im1 := imageio.imread('Tuebingen_Neckarfront.jpg'); // read image as .NET byte array
  Console.ReadKey(true);
end.
```

System.FloatingPoint. Данная библиотека содержит типы данных *Quarter* (8-битный float), *Half* (16-битный float) и *BFloat16* (16-битный float от Google Brain). Реализованные типы показаны на рис. 3, 4 и 5.

Рис. 3. Структура *Quarter*

Рис. 4. Структура *Half*

Рис. 5. Структура *BFloat16*

Типы данных *Half* и *BFloat16* активно используются в машинном обучении, и их поддержка должна быть доступна, по крайней мере, на уровне преобразований между встроенными типами .NET. Пример кода на VB.NET для работы с *System.FloatingPoint* предлагается ниже.

```
// Reference System.FloatingPoint.dll or System.AI.dll
Module Program
  Sub Main()
    Dim a As Quarter = Single.Parse(Console.ReadLine())
    Dim b As Half = Single.Parse(Console.ReadLine())
    Dim c As BFloat16 = Single.Parse(Console.ReadLine())
    Dim e As Object = a * b + c
    Console.WriteLine(String.Format("{0} * {1} + {2} = {3}, type = {4}", a, b, c, e, e.GetType))
    Console.ReadKey(True)
  End Sub
End Module
```

System.Complex. Библиотека реализует типы комплексных вещественных чисел, в которых действительная и мнимая части представлены значениями типа *Quarter*, *Half*, *BFloat16*, *Single* или *Double*. То есть каждому простому вещественному типу соответствует комплексный тип. Комплексные числа используются в машинном обучении и анализе сигналов.

PyType.NET. Эта библиотека реализует некоторые типы данных языка Python, которые позволяют сократить код. Одним из таких типов является шаблонный класс объединения (*Union*), который позволяет передавать значение одного из указанных типов данных. Он активно используется, когда необходимо передать одно число или пару чисел (например, в качестве параметров двумерного слоя свертки).

warnings.NET. Библиотека является аналогом библиотеки предупреждений стандартного пакета Python и предназначена для вывода предупреждений о некритических ситуациях, возникших во время выполнения программы.

DotnetExtensions. Библиотека содержит методы расширения для стандартных классов .NET. Задачи обработки данных часто требуют простых вспомогательных функций, применяемых, например, к числовым массивам. Такие функции могут быть использованы для отладки, ввода/вывода массивов данных и т.д. Подобный функционал также встроен в пакет *NumPy*, который является стандартным для работы с массивами в экосистеме Python. *DotnetExtensions* расширяет некоторые классы BCL функциями, аналогичными тем, которые содержатся в пакетах Python. В качестве примера приведём код на C# для сохранения числового массива в файл с использованием методов расширения *System.Array.tobytes()* и *System.IO.Stream.Write()*.

```
// string fname - file name
// arr - (multidimentional) array of basic .NET type (such as float or int)
using(var f = File.Create(fname))
{
 f.Write(arr.tobytes());
}
```

Torch.NET. *Torch.NET* - это библиотека машинного обучения и линейной алгебры с поддержкой автоматического дифференцирования. *Torch.NET* является прямым аналогом библиотеки *PyTorch* экосистемы Python. *Torch.NET*, как и другие библиотеки в составе *System.AI*, во многом копирует синтаксис своего аналога, однако, возможности *Torch.NET* всё же отличаются от возможностей *PyTorch*. Система типов *Torch.NET* значительно богаче, чем у *PyTorch*: дополнительно поддерживаются типы данных *Quarter*, *CQuarter*, *CBFloat16*, *UInt16*, *UInt32*, *UInt64*. Квантование на данный момент не поддерживается. Важное отличие *Torch.NET* от *PyTorch* заключается в том, что

все операции со всеми типами данных реализованы для центрального процессора. На данный момент библиотека *Torch.NET* активно развивается.

Torchvision.NET. *Torchvision.NET* - это аналог модуля *torchvision*, входящего в состав *PyTorch*. *Torchvision.NET* содержит предобученные нейронные сети, методы чтения и дополнения данных, а также инструменты для работы с датасетами. Пример кода, который выполняет классификацию изображений с использованием *SqueezeNet*, показан ниже (Применены библиотеки *Torch.NET*, *Torchvision.NET*, *imageio.NET*).

```
using System;
using System.AI;
using System.Linq;
using System.Collections.Generic;
using models = System.AI.torchvision.models;
namespace Test
{
  public static class Program
   {
    public static torch.Tensor load(string uri)
     {
      return torch.tensor(imageio.imread(uri)).
      transpose(0,2).transpose(1,2).unsqueeze(0).@
      float() / 255f;
     }
     #region imagenet_classes
     public static string[] imagenet_classes = new
     string[]
     {
      <imagenet classes>
     };
     #endregion
     public static Dictionary<int, float> get_
     top5(float[] p)
     {
      <get top5 code>
     }
     public static void Main(string[] args)
     {
      var m = models.squeezenet1_1(true);
      var y = m.forward(load(args[0]));
      var pred = get_top5(y.squeeze(0).dotnet() as
      float[]);
      foreach(var p in pred)
      {
       Con-sole.WriteLine(imagenet_classes[p.
       Key]);
      }
      Console.ReadKey(true);
     }
   }
}
```

Conv.NET. *Conv.NET* - это реализация *convnet.js* для платформы .NET. Несмотря на то, что возможностей *Conv.NET* недостаточно для реализации сложных современных архитектур нейронных сетей, данная библиотека может быть использована в качестве учебной.

3. Методы

Ключевую роль в успехе и применимости фреймворка машинного обучения, как и любого другого программного продукта, играет эффективность кода. Реализация компонентов нейронной сети в управляемом коде представляет особый интерес, поскольку требует использования редко используемых специфических особенностей языка и платформы. Рассмотрим подход, используемый для реализации компонентов *System.AI*, на примере матричного умножения.

- Оптимизация алгоритма с учетом архитектуры вычислительной системы: минимизация непоследовательных обращений к оперативной памяти.
- Многопоточность.
- Прямой доступ к элементам матриц через указатели.
- Применение арифметики указателей.
- Векторизация.

Последовательное выполнение упомянутых выше оптимизаций позволяет получить прирост производительности более чем в 33 раза по сравнению с классическим (наивным) алгоритмом. Измерения проводились в 64-битном режиме для матриц 1000x1000.

Табл. 2

Влияние различных подходов к оптимизации кода на время его выполнения

Метод	Время, мс	Производительность
Наивный	3867	517,2 МФлопс
+ Кэш	2633	759,6 МФлопс
+ Многопоточность	903	2.2 ГФлопс
+ Указатели	**531**	**3.8 ГФлопс**
+ Арифметика	**261**	**7.6 ГФлопс**
+ Векторизация	**117**	**17 ГФлопс**

Существует несколько способов дальнейшего повышения производительности, одним из которых является использование .NET 5 или .NET Core, что позволит более эффективно, чем в .NET 4.8 с *System.Numerics.Vector4*, использовать векторные инструкции процессора.

При разработке *System.AI* используются компромиссные решения, позволяющие добиться максимальной эффективности по памяти и времени.

Зачастую разрабатываются новые алгоритмы и подходы. Например, в слоях свёртки используется модифицированный алгоритм im2col: вместо извлечения всех необходимых для свёртки участков входного изображения в матрицу с последующим умножением этой матрицы на матрицу ядер свёртки, как это происходит в известных фреймворках глубокого обучения, в *Torch.NET* каждый отдельный участок изображения преобразуется в вектор, который немедленно умножается на матрицу весов. Это решение позволяет радикально сократить объём занимаемой дополнительной памяти для вычисления свёртки, а также демонстрирует уровень производительности, сравнимый с уровнем классического im2col. Использование таких трюков возможно благодаря тому, что *System.AI* не полагается на готовые математические библиотеки, как это делают многие подобные продукты, вместо этого весь необходимый код пишется с нуля на C#.

4. Алгоритмы

Важным условием достижения высокой производительности программного обеспечения являются хорошие алгоритмы. При разработке *System.AI* выбираются компромиссные решения для достижения высокой эффективности по времени и памяти. Особое внимание уделяется алгоритмам свёрточных слоёв. В частности, был разработан новый алгоритм patch2vec, который представляет собой комбинацию наивного алгоритма свёртки и матричного im2col. Смысл большинства быстрых алгоритмов свёртки, таких как im2col или im2row, заключается в приведении свёртки к матричному умножению, что позволяет оптимизировать операции доступа к памяти за счёт использования кэша процессора. Однако такие методы требуют буфер для временных элементов, что крайне затратно. Предлагаемый метод (patch2vec) «на лету» разворачивает каждый участок входного изображения в вектор, а затем применяет к нему все фильтры свёртки. Этот алгоритм не уступает по эффективности классическим решениям вроде im2col, а на практике даже превосходит их в некоторых случаях. Буфер для этого алгоритма будет иметь размер , что много меньше, чем в случае аналогичных методов. Более того, patch2vec не накладывает ограничений на параметры свёртки, в отличие, например, от метода Шмуэля Винограда. Предлагаемый алгоритм трудно вписать в классические фреймворки машинного обучения из-за того, что они ориентированы на использование GEMM в качестве вычислительного ядра. Чистые реализации на C# или других языках программирования позволяют легко это сделать. В результате анализа существующих статей на тему быстрых свёрток [16] [17] было установлено, что предлагаемый алгоритм является новым и ранее не был представлен.

Patch2Vec.

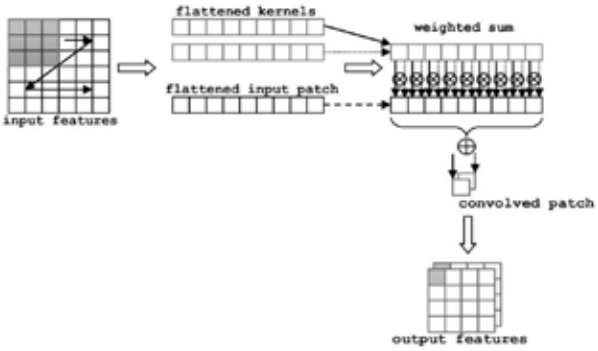

Рис. 6. Иллюстрация операции двумерной свёртки методом patch2vec.

На рис. 6 показана иллюстрация двумерной свёртки с помощью алгоритма patch2vec. Рассматривается случай, когда ядро свёртки квадратное и имеет размер 3x3, шаг симметричен и равен 1, расширение ядра отсутствует (=1), дополнение нулями отсутствует, количество групп равно единице. Тем не менее, алгоритм работает с любыми параметрами, разрешенными для операции свёртки. Кроме того, этот алгоритм также может быть применен для одномерных и трёхмерных свёрток.

Идея patch2vec состоит в том, чтобы избавиться от многократных (равных числу ядер свёртки) обращений к одним и тем же элементам входного тензора на каждом шаге свёртки. Вместо многократного доступа к одним и тем же данным, расположенным непоследовательно в памяти, предлагаемый алгоритм извлекает значения входного тензора, необходимые для применения ядра свёртки в данной точке, и записывает их в буфер (данные расположены последовательно в памяти), после чего ядра свёртки применяются не к области входного тензора, а к значениям, записанным в буфер. В этом случае все операции доступа к памяти при вычислении взвешенной суммы будут последовательными. Использование patch2vec уменьшает количество непоследовательных обращений к памяти в k раз (где k - количество ядер свёртки). Таким образом, алгоритм особенно эффективен при большом количестве ядер.

Vec2Patch. Vec2patch - это алгоритм, который является противоположностью patch2vec и предназначен для эффективного вычисления транспонированных свёрток и обратного распространения ошибки через обычную свёртку. Эффективная многопоточная реализация vec2patch сложна, по-

скольку требует синхронизации доступа к памяти при проецировании результирующего вектора на область результирующего тензора.

Заключение и дальнейшая работа

На данный момент основной задачей является реализация полного функционала *PyTorch* в библиотеке *Torch.NET*. Бета-версия *Torch.NET* будет опубликована в ближайшее время. Ещё одной важной задачей является добавление поддержки вычислений на графическом процессоре. Для этого планируется использовать технологию OpenCL - кроссплатформенный стандарт, поддерживаемый большинством современных видеокарт.

Исходные коды, двоичные файлы и документация, связанные с этим проектом, доступны на GitHub под лицензией Apache-2.0: https://github.com/ColorfulSoft/System.AI

Литература

1. .NET homepage. Available at: https://dotnet.microsoft.com (accessed November 22, 2022)
2. Mono homepage. Available at: https://www.mono-project.com (accessed November 22, 2022)
3. Xamarin homepage. Available at: https://dotnet.microsoft.com/apps/xamarin (accessed November 22, 2022)
4. Imageio homepage. Available at: https://github.com/imageio/imageio (accessed November 22, 2022)
5. *Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala.* 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703. Available at: https://arxiv.org/abs/1912.01703 (accessed November 22, 2022)
6. convent.js homepage. Available at: https://cs.stanford.edu/people/karpathy/convnetjs (accessed November 22, 2022)
7. *Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng.* 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467. Available at: https://arxiv.org/abs/1603.04467 (accessed November 22, 2022)
8. ConvNetCS GitHub repository. Available at: https://github.com/mashmawy/ConvNetCS (accessed November 22, 2022)
9. *Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, Stan Bileschi, Michael Terry, Charles Nicholson, Sandeep N. Gupta, Sarah Sirajuddin, D. Sculley, Rajat Monga, Greg Corrado, Fernanda B. Viégas, Martin Wattenberg.* 2019. TensorFlow.js: Machine Learning for the Web and Beyond. arXiv:1901.05350. Available at: https://arxiv.org/abs/1901.05350 (accessed November 22, 2022)
10. AlbiruniML GitHub repository. Available at: https://github.com/mashmawy/AlbiruniML (accessed November 22, 2022)
11. SciSharp STACK homepage. Available at: https://scisharp.github.io/SciSharp/ (accessed November 22, 2022)
12. Demos GitHub repository. Available at: https://github.com/ColorfulSoft/StyleTransfer-Colorization-SuperResolution (accessed November 22, 2022)
13. *David W. Brown.* 2018. MyCaffe: A Complete C# Re-Write of Caffe with Reinforcement Learning. arXiv:1810.02272. Available at: https://arxiv.org/abs/1810.02272 (accessed November 22, 2022)
14. *Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell.* 2014. Caffe: Convolution Architecture for Fast Feature Embedding. arXiv:1408.5093. Available at: https://arxiv.org/abs/1408.5093 (accessed November 22, 2022)
15. Google Colab homepage. Available at: https://colab.research.google.com (accessed November 22, 2022)
16. *Lavin, Scott Gray.* 2015. Fast Algorithms for Convolutional Neural Networks. arXiv:1509.09308. Available at: https://arxiv.org/abs/1509.09308 (accessed November 22, 2022)
17. *Anton V. Trusov, Elena E. Limonova, Dmitry P. Nikolaev and Vladimir V. Arlazarov.* 2021. p-im2col: Simple Yet Efficient Convolution Algorithm With Flexibly Controlled Memory Overhead. IEEE Access PP(99):1-1 (2021)

Брыкин Глеб Сергеевич. Московский Государственный Технический Университет им. Н.Э. Баумана, г. Москва, Россия. Количество печатных работ: 1. Область научных интересов: глубокое обучение, электроника. E-mail: glebbrykin@colorfulsoft.ru (ответственный за переписку).

The System.AI Project: Fully Managed Cross-Platform Machine Learning and Data Analysis Stack for .NET Ecosystem

G.S. Brykin

Bauman Moscow State Technical University, Moscow, Russia

Abstract. In recent years, machine learning technologies have become increasingly popular in widespread tasks such as image stylization, black-and-white image coloring, super-resolution of images, fake data searching, voice and image recognition. In this regard, there is a need to implement a set of tools for integrating artificial intelligence systems into applications for mobile devices, smart home devices, and home PCs. The paper describes a solution that allows developers to integrate data analysis and machine learning systems directly into a user application, which will allow to produce a lightweight, portable, and cross-platform monolithic application, which is often not possible with existing solutions. The main features of the proposed solution are the focus on the Microsoft .NET [1] ecosystem and the use of exclusively standard features of BCL and C# programming language. The implemented package of tools is completely cross-platform and hardware independent. The API is similar in many ways to its Python counterparts, which allows to quickly migrate Python codes into a .NET project.

Keywords: *machine Learning, Data Analysis, .NET Framework, Mono [2], Xamarin [3], .NET Core [1], .NET Standard [1], Managed Code.*

DOI: 10.14357/20790279230108

References

1. .NET homepage. Available at: https://dotnet.microsoft.com (accessed November 22, 2022)
2. Mono homepage. Available at: https://www.mono-project.com (accessed November 22, 2022)
3. Xamarin homepage. Available at: https://dotnet.microsoft.com/apps/xamarin (accessed November 22, 2022)
4. Imageio homepage. Available at: https://github.com/imageio/imageio (accessed November 22, 2022)
5. *Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala.* 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703. Available at: https://arxiv.org/abs/1912.01703 (accessed November 22, 2022)
6. convent.js homepage. Available at: https://cs.stanford.edu/people/karpathy/convnetjs (accessed November 22, 2022)
7. *Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng.* 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467. Available at: https://arxiv.org/abs/1603.04467 (accessed November 22, 2022)
8. ConvNetCS GitHub repository. Available at: https://github.com/mashmawy/ConvNetCS (accessed November 22, 2022)
9. *Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, Stan Bileschi, Michael Terry, Charles Nicholson, Sandeep N. Gupta, Sarah Sirajuddin, D. Sculley, Rajat Monga, Greg Corrado, Fernanda B. Viégas, Martin Wattenberg.* 2019. TensorFlow.js: Machine Learning for the Web and Beyond. arXiv:1901.05350. Available at: https://arxiv.org/abs/1901.05350 (accessed November 22, 2022)
10. AlbiruniML GitHub repository. Available at: https://github.com/mashmawy/AlbiruniML (accessed November 22, 2022)

11. SciSharp STACK homepage. Available at: https://scisharp.github.io/SciSharp/ (accessed November 22, 2022)
12. Demos GitHub repository. Available at: https://github.com/ColorfulSoft/StyleTransfer-Colorization-SuperResolution (accessed November 22, 2022)
13. *David W. Brown.* 2018. MyCaffe: A Complete C# Re-Write of Caffe with Reinforcement Learning. arXiv:1810.02272. Available at: https://arxiv.org/abs/1810.02272 (accessed November 22, 2022)
14. *Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell.* 2014. Caffe: Convolution Architecture for Fast Feature Embedding. arXiv:1408.5093. Available at: https://arxiv.org/abs/1408.5093 (accessed November 22, 2022)
15. Google Colab homepage. Available at: https://colab.research.google.com (accessed November 22, 2022)
16. *Lavin, Scott Gray.* 2015. Fast Algorithms for Convolutional Neural Networks. arXiv:1509.09308. Available at: https://arxiv.org/abs/1509.09308 (accessed November 22, 2022)
17. *Anton V. Trusov, Elena E. Limonova, Dmitry P. Nikolaev and Vladimir V. Arlazarov.* 2021. p-im2col: Simple Yet Efficient Convolution Algorithm With Flexibly Controlled Memory Overhead. IEEE Access PP(99):1-1 (2021)

Gleb S. Brykin. Bauman Moscow State Technical University, ul. Baumanskaya 2-ya, 5/1, Moscow, 105005, Russia, e-mail: glebbrykin@colorfulsoft.ru