JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 20 (2021) 230–236
Received: 22 June 2021
Accepted: 2 July 2021

Performance analysis of machine learning libraries

Analiza wydajności bibliotek uczenia maszynowego

Ewa Justyna Kędziora*, Grzegorz Krzysztof Maksim*

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The paper presents results of performance analysis of machine learning libraries. The research was based on ML.NET and TensorFlow tools. The analysis was based on a comparison of running time of the libraries, during detection of objects on sets of images, using hardware with different parameters. The library, consuming fewer hardware resources, turned out to be TensorFlow. The choice of hardware platform and the possibility of using graphic cores, affecting the increase in computational efficiency, turned out to be not without significance.

Keywords: machine learning; performance; TensorFlow; ML.NET

Streszczenie

W artykule zaprezentowane zostały wyniki analizy wydajności bibliotek uczenia maszynowego. Badania oparte zostały na narzędziach ML.NET i TensorFlow. Przeprowadzona analiza bazowała na porównaniu czasu działania bibliotek podczas wykrywania obiektów na zbiorach zdjęć, przy użyciu sprzętu o różnych parametrach. Biblioteką, zużywającą mniejsze zasoby sprzętowe, okazała się TensorFlow. Nie bez znaczenia okazał się wybór platformy sprzętowej oraz możliwość użycia rdzeni graficznych, mających wpływ na zwiększenie wydajności obliczeń.

Słowa kluczowe: machine learning; wydajność; TensorFlow; ML.NET

*Corresponding authors

Email address: ewa.kedziora@pollub.edu.pl (E. J. Kędziora), grzegorz.maksim@pollub.edu.pl (G. K. Maksim)

1. Wstęp

We współczesnym świecie wykorzystanie technik uczenia maszynowego stało się niezwykle powszechne. Zgodnie z jednym z wielu przeprowadzonych na ten temat sondaży, już w 2019 roku stwierdzono, że prawie połowa szybko rozwijających się przedsiębiorstw ma w planach wdrożenie sztucznej inteligencji w przeciągu roku [1].

Nierozerwalną koncepcją mieszczącą się w ramach sztucznej inteligencji jest uczenie maszynowe (ang. machine learning), gdzie komputer wykrywa jak ma zostać wykonane zadanie dzięki określonym wzorcom i przekazanemu zbiorowi danych. Powstało wiele narzędzi oraz bibliotek wspierających programistów w tworzeniu i trenowaniu modeli, a wraz z rosnącym zapotrzebowaniem zwiększa się liczba dostępnych rozwiązań, które w znacznej części są typu open source.

Algorytmy uczenia maszynowego korzystają z parametrów, opartych na danych szkoleniowych. Dostosowanie danych w taki sposób, aby w jak najlepszy sposób reprezentowały rzeczywistość, pozwala na uzyskanie dokładniejszych wyników i użytecznych analiz, których osiągnięcie nie byłoby możliwe do zbadania przez człowieka lub wymagałoby więcej nakładów pracy [2].

W celu lepszego poznania zastosowań uczenia maszynowego i wcześniejszych badań związanych z wydajnością bibliotek uczenia maszynowego przeprowadzono analizę artykułów o podobnym temacie.

W artykule „Review and comparative analysis of machine learning libraries for machine learning” porównano dokładność czasów rozpoznawania cyfr pisanych pismem odręcznym przy pomocy bibliotek TensorFlow, PyTorch, Theano, Keras oraz SciKit-Learn. Wydajność była mierzona na podstawie czasu treningu sieci neuronowej oraz dokładności rozpoznawania danych pobranych z bazy MNIST. Czas uczenia biblioteki TensorFlow był bardzo podobny do narzędzi Scikit-learn oraz Keras i wynosił od 1 do 3 sekund, co zdecydowanie pokonywało bibliotekę PyTorch, osiągającą czas treningu na poziomie 25 sekund [3].

Autorka książki „Hands-On Machine Learning with SciKit-Learn and TensorFlow”, w swojej publikacji stwierdza, że jednym z większych problemów wydajności jest jej spadek, w przypadku dostarczenia błędnych danych. Jako rozwiązanie takiego problemu wskazane jest reagowanie na nieprawidłowości za pomocą algorytmu wykrywania anomalii, umożliwiającego śledzenie przychodzących danych [4].

Autorzy jednej z prac sugerowali skupienie największej uwagi w kierunku uczenia nadzorowanego, czyli np. na stworzeniu etykiet, prawidłowo opisujących grupę przykładowych obrazów, ponieważ niezawodność uczenia wrasta dzięki ingerencji człowieka i opisaniu danych [5]. Podobne wnioski uzyskali naukowcy realizujący doświadczenia na otwartoźródłowej bazie ImageNet stwierdzając, że rozwiązaniem sprzyjającym wydajności, jest zmierzenie dokładności etykiet w uczeniu nadzorowanym [6].

Algorytm jest nadzorowany przez człowieka aby w efekcie sam, po analizie szeregu przypadków, mógł zwracać poprawne odpowiedzi z jak największą dokładnością. To zagadnienie dotyczy nie tylko wykrywa-

nia obiektów na obrazach ale może służyć również do wielu zadań, w których szukany wynik jest liczbą. Na podstawie informacji statystycznych, warunków atmosferycznych i ograniczeń prędkości, istnieje możliwość określenia liczby możliwych wypadków na drodze lub określenia cen sprzedaży nieruchomości na podstawie wielkości, stanu lub lokalizacji [7]. Zagadnienia tego rodzaju mieszczą się w definicji techniki regresji.

Krokiem w kierunku niezawodności są techniki głębokiego uczenia (ang. deep learning) wykorzystujące sztuczne sieci neuronowe. Symulują one systemy, które odzwierciedlają aspekty działania prawdziwych neuronów. Są odpowiedzialne za wiele ostatnich postępów w uczeniu maszynowym jako całości, szczególnie w takich dziedzinach, jak rozpoznawanie obrazów i przetwarzanie języka naturalnego [8].

Architektura sieci neuronowej nadal znacznie odbiega swoją strukturą od złożonej sieci mózgowej składającej się z ogromnej ilości neuronów. Pozostaje jeszcze kwesta użycia zasobów sprzętowych oraz poboru mocy jaki jest potrzebny przy działaniu sztucznych sieci. Przy najbardziej zaawansowanych operacjach wymagane są duże zasoby energii oraz rozbudowane stacje, co powoduje kolejne problemy przy wykorzystaniu uczenia maszynowego [9].

W dokumentacji biblioteki TensorFlow, jako działanie sprzyjające doskonaleniu modelu, wskazywane jest rozwiązanie wąskich gardeł. Sprawdzenie czy model radzi sobie dobrze jest możliwe dzięki zaimplementowaniu go i profilowaniu, które pomaga w analizie zużycia zasobów sprzętowych- pamięci i czasu [10].

Rozwiązanie, które zaproponowała firma NIVIDA umożliwia wykorzystanie rdzeni graficznych w formie procesów ogólnego przeznaczenia. Narzędzia pozwalające na wykorzystanie kart graficznych pozwalają na przyspieszenie sprzętowe [11].

W artykule „Performance Analysis of Deep Learning Libraries: TensorFlow and PyTorch" porównano dwa narzędzia wykorzystujące rdzenie CUDA: PyTorch oraz TensorFlow. Po analizie wydajności tych bibliotek zauważono, że pierwsza z nich wykorzystuje mniejszy potencjał GPU niż TensorFlow. Podczas badań zwrócono uwagę również na to, że pomimo podobnego użycia jednostki CPU, przy TensorFlow temperatura procesora była kilka stopni wyższa niż przy wykorzystaniu konkurencyjnego rozwiązania. Narzędzie od Google jest również znacznie bardziej proste i intuicyjne. PyTorch natomiast wykorzystuje mniej zautomatyzowane funkcje, co skutkuje większym wkładem włożonym przez programistów [12].

Kolejne powstające platformy mają zminimalizować problemy z wydajnością, unifikacją, rozszerzalnością i skalowalnością, umożliwiając jak najlepszą wydajność przy maksymalnym wykorzystaniu zasobów sprzętowych [13].

TensorFlow jest biblioteką obecną w wielu badaniach, natomiast materiałów dotyczących biblioteki ML.NET jest zdecydowanie mniej. Pierwsza z nich, w zestawieniach z innymi platformami, wielokrotnie osiągała zbliżone wyniki, będąc jednocześnie uzależnioną od parametrów sprzętowych. Analizowane pozycje literaturowe na temat badania wydajności, nie obejmowały porównania tych dwóch platform, co motywuje do przeprowadzenia eksperymentu obejmującego oba rozwiązania, umożliwiającego wybór narzędzia realizującego scenariusz wykrywania obiektów na obrazach w sposób optymalny. Celem autorów jest zbadanie wydajności bibliotek oferowanych przez firmy Microsoft i Google, pod względem szybkości badania i użycia zasobów sprzętowych.

2. Metodyka badań

2.1. Opis badania

W celu przeprowadzenia badania utworzono dwie aplikacje bazujące na algorytmach uczenia maszynowego:

1. Aplikacja konsolowa do badania wydajności biblioteki ML.NET
2. Aplikacja stworzona w środowisku Jupyter Notebook do badania czasu analizy obiektów przez TensorFlow.

Ich działanie sprowadza się do wprowadzania przez użytkownika danych w formie zbiorów obrazów. Następnie zbiór danych ulega przetworzeniu. Aplikacja wykrywa obiekty na obrazach i zapisuje rezultaty do osobnego katalogu. Wyniki z przeprowadzonej analizy są przedstawiane w postaci prawdopodobieństwa i ramek otaczających obiekty, co przedstawione jest na Rysunku 1. Podstawą takiego działania jest użycie prawidłowo wytrenowanego modelu.

Rysunek 1: Obraz po analizie z zaznaczonymi wykrytymi obiektami.

Badanie wydajności działania aplikacji polega na zmierzeniu czasów wykonywania tych samych operacji przez dwie różne biblioteki. W kodzie źródłowym dodano funkcję mierzenia czasu wykonania kodu. Po wykonaniu określonych zadań, czas był wyświetlany na ekranie.

W celu uzyskania jak najbardziej wiarygodnych wyników zostały użyte gotowe, wyuczone modele. Badanie zostało przeprowadzone na dwóch urządzeniach posiadających różną specyfikację, przedstawioną

w Tabeli 1, celem zweryfikowania, które z nich radzi sobie lepiej z narzuconym zadaniem i jakie parametry mają wpływ na wydajność działania bibliotek.

Tabela 1: Specyfikacja urządzeń użytych w badaniu

Urządzenie	Procesor			RAM	GPU
	Typ	Liczba rdzeni	Taktowanie (MHz)	(GB)	
Komputer stacjonarny	AMD Ryzen 7	8	od 3600 do 4400	16	GeForce RTX 3070
Laptop	Intel Core i5	2	od 1900 do 2051	8	-

Pierwsze stanowisko badawcze posiada dodatkowo kartę graficzną, która umożliwiła skorzystanie ze specjalnych rdzeni CUDA. CUDA jest zastrzeżoną platformą programistyczną dostosowaną do masowej równoległości zadań obliczeniowych, a posiadanie rdzeni w liczbie 5888, umożliwia równoległe przetwarzanie 5888 wektoryzowanych jednostek wykonawczych. Za ich pomocą, dzięki możliwościom oferowanym przez język Python, realne jest zbadanie czasu, jakiego potrzebuje sama karta graficzna.

Skorzystano z metody obserwacji polegającej na świadomym obserwowaniu funkcjonowania aplikacji podczas wykrywania obiektów na zbiorze obrazów.

2.2. Środowisko badawcze

W przeprowadzanym badaniu wykorzystane zostały dwie konkurencyjne biblioteki: TensorFlow-platforma oferowana przez Google Brain Team, najbardziej spopularyzowana wśród programistów zajmujących się uczeniem maszynowym i ML.NET-biblioteka od firmy Microsoft stworzona dla języków programowania C# oraz F#.

2.3. Model

Celem implementacji, skorzystano z modelu w formacie ONNX (ang. Open Neural Network Exchange), wytrenowanego na podobnym problemie. Został on pobrany ze zbioru ONNX Model Zoo. Operatory w ONNX są podzielone na zestawy funkcji i prymitywnych operatorów. Funkcja to operator, którego obliczenia mogą być wyrażone jako podgraf innych operatorów. Do ich opisu używany jest graf. W nim znajdują się listy węzłów, wejść, wyjść oraz wartości stałe. Dzięki użyciu rozszerzalnego modelu grafu obliczeniowego możliwe jest użycie go przy wykorzystaniu wielu różnych platform [14].

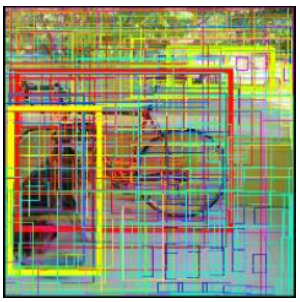
W początkowej fazie działania algorytmu uczenia maszynowego wymagane są obiekty typu DataView. Każdy model zawiera definicję danych wejściowych oraz wyjściowych, zawierającą nazwy, typy i rozmiary danych. Jeżeli któraś z danych jest niepoprawnie zdefiniowana bądź źle zaimplementowana, wówczas w trakcie działania programu mogą wystąpić wyjątki.

Obraz zostaje podzielony przez sieć neuronową na poszczególne regiony, tworząc siatkę o rozmiarach 13x13 (Rysunek 2). Następnie jest szacowane prawdopodobieństwo dla każdego regionu i na jego podstawie przewidywane są obwiednie (Rysunek 3, Rysunek 4).

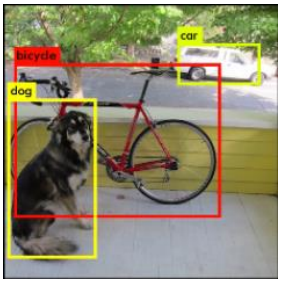
Przy trenowaniu modelu istotne jest najpierw zgromadzenie odpowiednich danych, a następnie ich załadowanie. W kolejnym etapie tworzone są potoki, które model jest w stanie przetworzyć. Następnym krokiem jest trenowanie modelu i jego ocena. Jeżeli model nie jest jeszcze odpowiednio wytrenowany, należy powtórzyć wszystkie czynności do uzyskania odpowiedniego efektu. Jeżeli model zostanie wytrenowany do odpowiedniego poziomu, zostaje zapisany. Proces uczenia go od podstaw pochłania dużo zasobów obliczeniowych i wiąże się z ogromną liczbą parametrów do skonfigurowania, dlatego też tak użyteczne są projekty społecznościowe jak ONNX, gdy samo wytrenowanie modelu nie jest tak istotne dla zakresu badań.

Rodzaje modeli ONNX można podzielić na dwa rodzaje: ONNX skupiony na sieciach neuronowych, który rozpoznaje tensory jako typy wejściowe i wyjściowe oraz ONNX-ML, pozwalający na rozpoznawanie sekwencji i map. Drugi typ wykorzystuje modele, które nie zostały stworzone w oparciu o sieci neuronowe.

Rysunek 2: Schemat działania algorytmu wykrywania obiektów na obrazie, etap 1 [15].



Rysunek 3: Schemat działania algorytmu wykrywania obiektów na obrazie, etap 2 [15].



Rysunek 4: Schemat działania algorytmu wykrywania obiektów na obrazie, etap 3 [15].

Wykorzystany w badaniach model to Tiny YOLO2. Jest to sieć neuronowa, pozwalająca na wykrywanie obiektów w czasie rzeczywistym [15][16]. W czasie testu jest analizowana cała fotografia, więc prognozy bazują na globalnym kontekście obrazu. Modele YOLO charakteryzują się bardzo dużą szybkością wykrywania obiektów oraz wysoką dokładnością. Dzięki wytrenowaniu modeli na procesorach GPU, modele te uzyskały ogromną dokładność i bardzo dobrą szybkość. Rozmiary modeli jednak są dosyć duże i często przy środowiskach z ograniczoną pamięcią nie są one w stanie pracować w zadowalający sposób [17].

2.4. Implementacja biblioteki TensorFlow

Za pośrednictwem biblioteki TensorFlow, dane są przetwarzane w postaci grafu. Działanie to wymaga przekształcenia obiektów do postaci tensora. Tensorami są krawędzie grafu łączące wierzchołki, które z kolei odzwierciedlają działania matematyczne. Podczas przetwarzania obrazu odbywa się konwersja, co skutkuje powstaniem tablicy liczb, która reprezentuje poddawany analizie obraz i na tym etapie tablica może być weryfikowana przez model. Niezbędnym krokiem jest określenie, które dane mają być podane na wyjściu modelu i przetworzone, ponieważ sam model składa się z danych wejściowych i wyjściowych. Powyższe działania zostały przeprowadzone z użyciem platformy opensource TensorFlow Object Detection API.

Implementacja tej biblioteki przeprowadzona została w środowisku do wykonywania aktywnego kodu krok po kroku tj. Jupyter Notebook w języku Python. TensorFlow jest narzędziem wszechobecnym we współczesnej technologii, dlatego też wdrożenie aplikacji, bazującej na tej platformie można wykonać na większości urządzeń m.in. komputerach lokalnych, procesorach i procesorach graficznych, klastrach, oraz na urządzeniach z systemem Android i iOS.

Celem prawidłowego przebiegu eksperymentu zostały użyte dodatkowe pakiety:

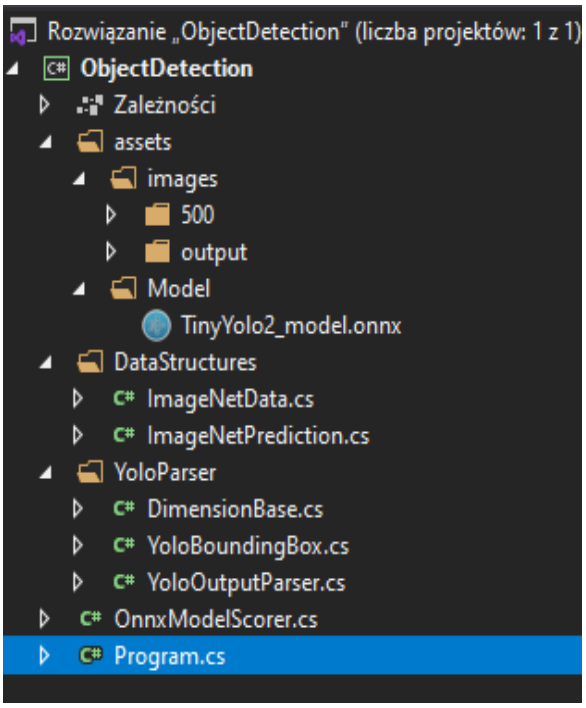
1. Pakiet pillow-umożliwiający przetworzenie obrazów do języka akceptowalnego przez język Python,
2. Pakiet jupyter-powiązany z możliwością tworzenia arkuszy do wykonywania kodu w Pythonie.
3. Pakiet lxml-umożliwiająca przetwarzanie plików w formacie xml.

Baza obrazów wykorzystanych do przeanalizowania przez algorytmy w eksperymencie badawczym to COCO. Dodatkowo użyto pakietu API umożliwiającego wykorzystanie powyższej bazy w projekcie.

2.5. Implementacja biblioteki ML.NET

Gotowa struktura katalogów projektu została przedstawiona na rysunku 5. Folder Model, zawiera wcześniej wspomniany model TinyYolo2_model.onnx. Katalog assets zawiera testowane obrazy, natomiast zdjęcia przetworzone z wykrytymi obiektami mieszczą się w katalogu output.

Zdefiniowanie zmiennych, które reprezentują lokalizację folderów z obrazami wraz z zawierającym model katalogiem zostały uwzględnione w klasie Main.



Rysunek 5: Struktura katalogów projektu w środowisku Visual Studio.

Tworzenie obiektów z klas wcześniej przedstawionych, zawarte jest w bloku try-catch. To w nim znajduje się m.in. obiekt, który przechowuje informację o prawdopodobieństwie, zadeklarowana zostaje instancja obwiedni, a także wraz z głównym kodem aplikacji następuje obliczanie czasu wykonania kodu.

Jednym z głównych obiektów tworzonych w tym miejscu jest także model punktowy dla modelu w formacie ONNX.

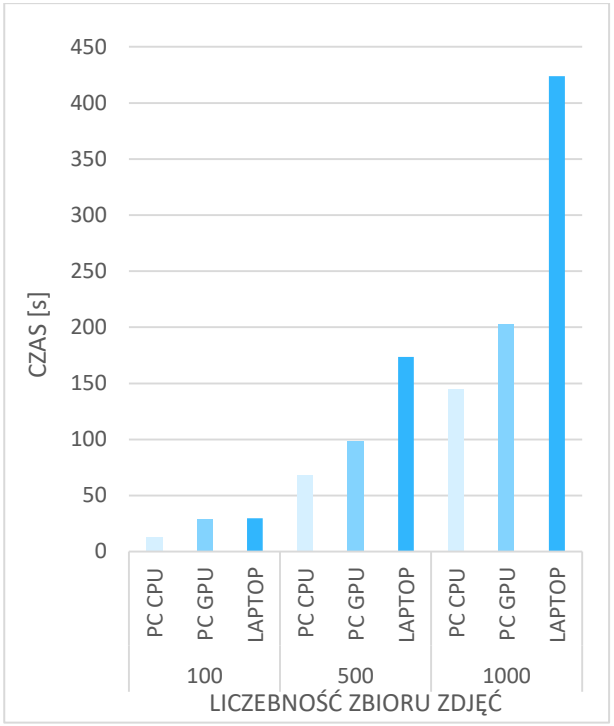
3. Wyniki badań

Metodyka badawcza, polegała na porównaniu trafności procesu wykrywania obiektów na obrazach, przez wybrane, konkurencyjne biblioteki oraz na zweryfikowaniu szybkości działania poszczególnych bibliotek na urządzeniach posiadających odmienne parametry.

Uzyskane wyniki podzielone zostały pod kątem liczebności zdjęć oraz urządzeń, na którym badanie zostało przeprowadzone. Podział ten, umożliwiał przedstawienie różnic w działaniach aplikacji i przede wszystkim w funkcjonowaniu algorytmów.

W przypadku biblioteki TensorFlow zdjęcia podzielone zostały na trzy zbiory zawierające 100, 500 i 1000 obrazów. Dla każdego ze zbiorów wyniki, wyrażone w sekundach, prezentowały ten sam trend. Czas przetwarzania obrazów w każdej z badanych grup, był najkrótszy dla CPU, wynosząc kolejno, 13, 68 i 145 sekund. Analiza obrazów z użyciem karty graficznej i wspomnianych w rozdziale 3 rdzeni CUDA wyniosła 29, 99 i 203 sekundy.

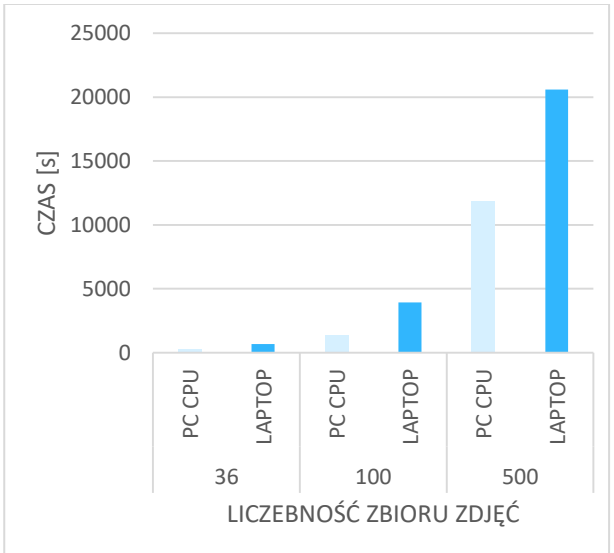
Wyniki na laptopie z wbudowaną kartą graficzną wyniosły 30, 174 i 424 sekundy. Zauważalny jest dwukrotny wzrost czasu pomiędzy GPU i laptopem na korzyść karty graficznej (Rysunek 6).



Rysunek 6: Czas analizy zbiorów zdjęć wykonanej na Laptopie, GPU i CPU z wykorzystaniem biblioteki TensorFlow.

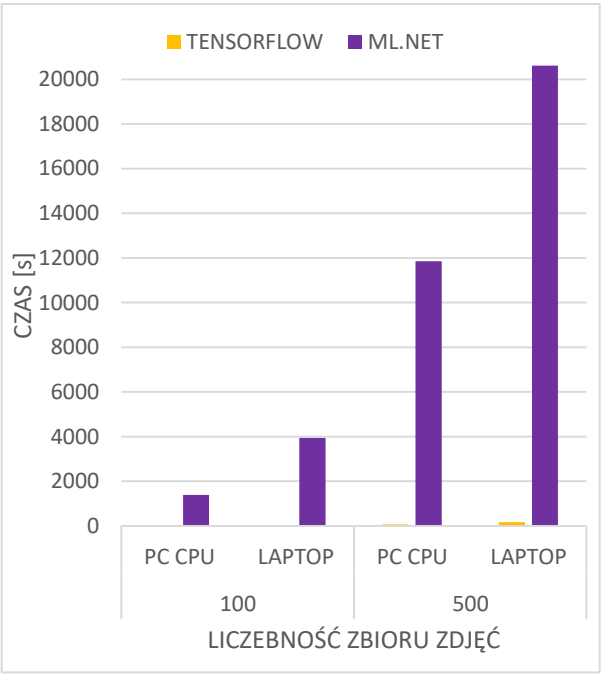
Podczas analizy biblioteki ML.NET zdjęcia podzielone zostały na trzy zbiory: 36, 100 i 500 obrazów. W czasie trwania eksperymentu prowadzono próby przeprowadzenia go na liczebniejszych zbiorach, jednak takie działanie było ograniczane przez możliwą do wykorzystania pamięć RAM.

Czas, poświęcony przez program na przetworzenie fotografii i wykrycie obiektów, cechował się podobną tendencją co poprzednia biblioteka. Najkrótsze wyniki zostały osiągnięte za pośrednictwem CPU, wynosząc kolejno: 250, 1393 oraz 11846 sekund. Mniej korzystne czasy zostały osiągnięte na laptopie, tj. drugim badanym w przypadku ML.NET urządzeniu, wynosząc 671, 3941 i 20605 sekund (Rysunek 7).



Rysunek 7: Czas analizy zbiorów zdjęć wykonanej na Laptopie i CPU z wykorzystaniem biblioteki ML.NET.

Zestawienie wyników uzyskanych z obu aplikacji wykazało ponad 100 razy dłuższy czas poświęcony na wyznaczone zadanie w badaniu przeprowadzonym za pomocą ML.NET (Rysunek 8).



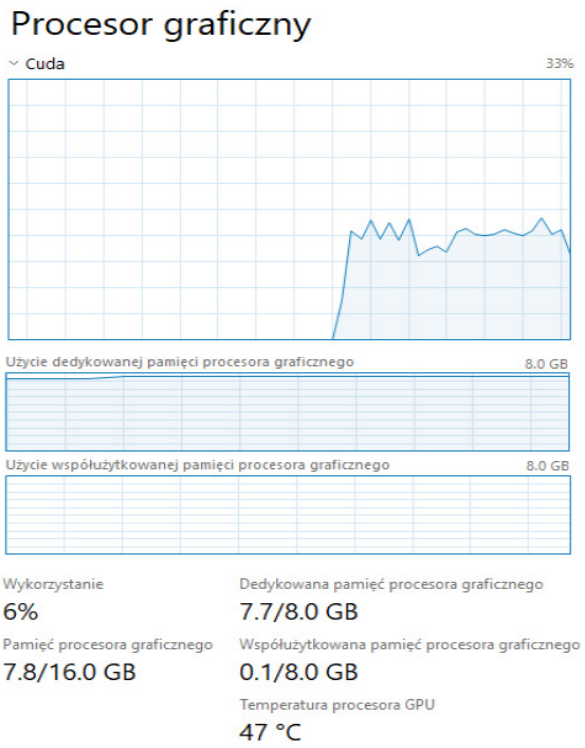
Rysunek 8: Graficzne porównanie czasu wykrycia obiektów na obrazach przez biblioteki TensorFlow i ML.NET dla różnych środowisk badawczych.

Przy próbie analizy zbioru zdjęć składającego się z 1000 zdjęć, został otrzymany komunikat o braku zasobów pamięciowych. Podobna sytuacja, jednak przy zbiorze dwukrotnie większym, nastąpiła również dla biblioteki TensorFlow. Użycie procesora było w tym przypadku znacznie niższe i tym samym nie miało to wpływu na otrzymany czas przetwarzania obrazów.

Tabela 2: Porównanie czasu wykrycia obiektów na obrazach przez biblio-teki TensorFlow i ML.NET dla różnych środowisk badawczych

Biblioteka	Środowisko badawcze	Liczba zdjęć	Czas przetwarzania (s)
TensorFlow	PC CPU	100	13
		500	68
	Laptop	100	30
		500	174
ML.NET	PC CPU	100	1393
		500	11846
	Laptop	100	3941
		500	20605

W przypadku użycia w aplikacji możliwości wykorzystania rdzeni graficznych, zaobserwowano wzrost użycia rdzeni CUDA oraz znaczne wykorzystanie dedykowanej pamięci procesora graficznego. Użycie procesora nie zmieniło się znacząco, natomiast czas działania uległ nieznacznemu wydłużeniu (Rysunek 9).



Rysunek 9: Zużycie procesora graficznego podczas analizy obrazów za pomocą TensorFlow, na komputerze stacjonarnym.

4. Wnioski

W tym artykule zostały porównane dwie biblioteki uczenia maszynowego pod kątem czasu poświęconego na wykrycie obiektu znajdującego się na obrazie, oraz zużycia zasobów sprzętowych urządzeń o odmiennej specyfikacji.

Przeprowadzona analiza wyników pozwoliła zauważyć, że biblioteka TensorFlow zdecydowanie lepiej sprawdza się w scenariuszu obejmującym wykrywanie obiektów na zdjęciach niż biblioteka ML.NET.

Na podstawie przeprowadzonych obserwacji potwierdzono tezę, że działanie biblioteki TensorFlow jest uzależnione od platformy sprzętowej. Wykorzystanie rdzenia graficznego i rdzeni CUDA pozwala na zwiększenie wydajności obliczeń. Liczba przetwarzanych grafik nie wpłynęła znacząco na długość wykonywania algorytmu rosnąc proporcjonalnie.

W badaniach zrealizowanych za pomocą biblioteki ML.NET analiza fotografii była bardzo powolna, pomimo maksymalnego wykorzystania procesora.

Zdecydowanie widoczna przewaga TensorFlow może wynikać z większej liczby użytkowników, którzy swoje działania związane z zagadnieniem uczenia maszynowego opierają na tej platformie. ML.NET pomimo gorszych osiąganych czasów w przeprowadzonym badaniu, nadal dociera do dużego grona użytkowników, z powodu bazowania na językach C# i F#.

Literatura

[1] Leading business in the age of AI, https://news.microsoft.com/europe/features/leaders-look-to-embrace-ai-and-high-growth-companies-are-seeing-the-benefits, [29.11.2020].

[2] Przewodnik po strukturze ML.NET, https://docs.microsoft.com/pl-pl/dotnet/machine-learning/how-does-mldotnet-work, [30.11.2020].

[3] M. N. Gevorkyan, A. V. Demidova, T. S. Demidova, A. Sobolev, Review and comparative analysis of machine learning libraries for machine learning, Discrete And Continuous Models And Applied Computational Science 27 (2019) 305-315, http://dx.doi.org/10.22363/2658-4670-2019-27-4-305-315.

[4] A. Géron, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly, 2019.

[5] E. Brill, M. Banko, Scaling to very large corpora for natural language disambiguation, Proceedings of 39th Annual Meeting on Association for Computational Linguistics (2001) 26-33, https://doi.org/10.3115/1073012.1073017.

[6] V. Shankar, R. Roelofs, H. Mania, A. Fang, B. Recht, L. Schmidt, Evaluating Machine Accuracy on ImageNet, 37th International Conference on Machine Learning (2020) 8634-8644.

[7] E. Zuccarelli, Using machine learning to predict car accidents, https://towardsdatascience.com/using-machine-learning-to-predict-car-accidents-44664c79c942, [15.06.2021].

[8] M. Hartley, T. S. G. Olsson, dtoolAI: Reproducibility for Deep Learning, Patterns 1(5) (2020) 100099, https://doi.org/10.1016/j.patter.2020.100073.

[9] C. Deng, X. Ji, C. Rainey, J. Zhang, W. Lu, Integrating Machine Learning with Human Knowledge, iScience 23(11) (2020) 101656, https://doi.org/10.1016/j.isci.2020.101656.

[10] Optimize TensorFlow performance using the Profiler, https://www.tensorflow.org/guide/profiler, [15.06.2021].

[11] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. García, I. Heredia, P. Malík, L. Hluchý, Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey, Artificial Intelligence Review 52 (2019) 77-124, https://doi.org/10.1007/s10462-018-09679-z.

[12] F. Florencio, E. D. M. Ordonez, T. V. Silva, M. C. Júnior, Performance Analysis of Deep Learning Libraries: TensorFlow and PyTorch, Journal of Computer Science 15 (2019) 785-799, http://dx.doi.org/10.3844/jcssp.2019.785.799.

[13] Z. Ahmed, S. Amizadeh, M. Bilenko, R. Carr, W.-S. Chin, Y. Dekel, X. Dupre, V. Eksarevskiy, E. Erhardt, C. Eseanu, S. Filipi, T. Finley, A. Goswami, M. Hoover, S. Inglis, M. Interlandi, S. Katzenber, Machine Learning at Microsoft with ML.NET, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (2019) 2448-2458, https://doi.org/10.1145/3292500.3330667.

[14] T. Jin, G.-T. Bercea, T. D. Le, T. Chen, G. Su, H. Imai, Y. Negishi, A. Leu, K. O'Brien, K. Kawachiya, A. E. Eichenberger, Compiling ONNX Neural Network Models Using MLIR (2020), https://arxiv.org/abs/2008.08272.

[15] J. Redmon, YOLO: Real-Time Object Detection, https://pjreddie.com/darknet/yolov2, [10.06.2021].

[16] J. Redmon, A. Farhadi, YOLO9000:Better, Faster, Stronger (2016), https://arxiv.org/abs/1612.08242.

[17] W. Fang, L. Wang, P. Ren, Tinier-YOLO: A Real-Time Object Detection, IEEE Access 8 (2019) 1935-1944, https://doi.org/10.1109/ACCESS.2019.2961959.