JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 12 (2019) 199-205
Wysłane: 2019-06-04
Przyjęte: 2019-06-11

Metody rozpoznawania gatunków grzybów na podstawie zdjęcia

Kamil Chodoła*, Grzegorz Czyż, Maria Skublewska-Paszkowska
Politechnika Lubelska,Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Celem artykułu jest porównanie dwóch metod rozpoznawania gatunków grzybów. W artykule zostały opisane dwie metody oparte na jednych z najpopularniejszych rozwiązań w dziedzinie image recognition, czyli Tensorflow oraz OpenCV. Do przeprowadzenia badań stworzono aplikację mobilną, w której obie metody zostały zaimplementowane oraz przetestowane. Dodatkowo aplikację wyposażono w mechanizmy ułatwiające zbieranie danych o aplikacji oraz algorytmach. Rezultaty badań wykazały, iż metoda oparta o Tensorflow o 9% skuteczniej rozpoznaje gatunki grzybów.

Słowa kluczowe: OpenCV;Tensorflow; image recognition

*Autor do korespondencji.

Adres e-mail: kamil.chodola@gmail.com

Methods for recognizing mushroom species on the basis of the photo

Kamil Chodoła*, Grzegorz Czyż, Maria Skublewska-Paszkowska
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B,20-618 Lublin, Poland

Abstract. The aim of the article is to compare two methods for identifying mushroom species. In article, two methods based on one of the most popular solutions in the field of image recognition, Tenosorflow and OpenCV, have been described. A research application was created to carry out the research, in which both algorithms were implemented and tested. In addition, the application was equipped with mechanisms facilitating the collection of application data and algorithms. The results of the research have show that the method based on Tensorflow by 9% more effectively recognizes mushroom species.

Keywords: OpenCV;Tensorflow; image recognition

*Corresponding author.

E-mail address: kamil.chodola@gmail.com

1. Wstęp

Szacuje się, iż obecnie w Polsce istnieje około 5 tysięcy gatunków grzybów[6]. Wśród nich istnieje wiele gatunków, które w mniejszym bądź większym stopniu zagrażają zdrowiu lub życiu człowieka po zjedzeniu lub nawet kontakcie z nimi. Co gorsza wiele popularnych, jadalnych grzybów jest podobnych do grzybów, które nie są zdatne do spożycia dla człowieka. Sprawia to, iż nawet osoby z dużym doświadczeniem w grzybiarstwie popełniają katastrofalne w skutkach błędy. Według statystyk, z roku na rok spada liczba zatruć spowodowanych spożyciem trujących grzybów, jednak ciągle poszukiwane są rozwiązania, które mogą spowodować zmniejszenie występowania tych przypadków do minimum.

W niniejszymartykule uwaga skupiać będzie się na metodach opartych o biblioteki OpenCV oraz Tensorflow. Do celów badawczych została utworzona aplikacja mobilna, w której zostały zaimplementowane wyżej wymienione metody. Po przeprowadzeniu badań została przeanalizowana poprawność zwracanych wyników, czas w jakim wyniki zostały zwrócone oraz wpływ danej metody na zasoby urządzenia. Na podstawie wyników tej analizy przedstawione zostały wnioski, dzięki którym wskazana została lepsza metoda oraz argumenty przemawiające za tym wyborem.

2. Metody rozpoznawania obiektów

Wykonane badania dotyczą zastosowania dwóch szkieletów aplikacji (ang. frameworków) w celu rozpoznawania obiektów na fotografiach. Są to: Tensorflow oraz OpenCV. W celu ich zaimplementowania w aplikacji mobilnej użyto dwóch nakładek (ang. wrapperów): OpenCVSharp oraz TensorflowSharp.

2.1. Tensorflow

Tensorflow jest jedną z najpopularniejszych bibliotek rozwiązujących problemy związane z uczeniem maszynowym (ang. machine learning) oraz głębokim uczeniem (ang. deep learning)[7]. Została ona zaprojektowana oraz wydana przez Google 9 listopada 2015 rokujako open-source. Tensorflow składa się z trzech modeli [4]:

- data model,
- execution model,
- programming model.

Data model składa się z tensorów, czyli podstawowych jednostek danych tworzonych, manipulowanych i zapisywanych w programie Tensorflow. *Execution model* polega na odpaleniu węzłów wykresu obliczeniowego w sekwencji zależności. Wykonanie rozpoczyna się od uruchomienia węzłów, które są bezpośrednio połączone z wejściami i zależą tylko od obecnych wejść.*Programming model* obejmuje wykresy przepływu danych lub wykresy obliczeniowe.

Jedną z największych zalet tego frameworku jest TensorBoard [3]. To narzędzie służy do wizualizacji danych. Dzięki niemu można w czytelny i prosty sposób prezentować m. in. dane wejściowe oraz wyjściowe, grafy obliczeniowe,

a nawet postęp treningu. Wizualizacja ułatwia wykrywanie błędów w strukturze sieci neuronowych oraz jej optymalizację.

Drugą bardzo użyteczną zaletą jest możliwość w łatwy sposób wykorzystywania tego samego modelu w wielu projektach bez względu na język programowania. W przypadku innych frameworków chcąc załadować parametry modelu wymuszane jest podanie całego kodu. Wiąże się to często z koniecznością przepisania kodu i ponowną kompilacją. Tensorflow natomiast potrzebuje jedynie pliku z wagami oraz nazwami warstw używanych do inferencji (najważniejsza jest warstwa wejściowa, ponieważ nie można bez niej wykonać grafu obliczeniowego).

Kolejną zaletą jest wsparcie wielu języków programowania takich jak:Python,Java,JavaScript,C++,Swift.

Dostęp do wszystkich funkcjonalności posiada jedynie Python, ponieważ jest najczęściej używany. Na potrzeby korzystania ze szkieletu aplikacji w niewspieranych językach zostały utworzone wiązania, dzięki czemu może być on używany w takich językach jak Ruby, czy C#. Biblioteka posiada również wady. Największą z nich jest nieuporządkowany kod. Nazwy wbudowanych metod nie pozwalają jednoznacznie stwierdzić, której z nich należy użyć, a wiele z nich nazywa się bardzo podobnie. Dokumentacja nie zawsze zawiera wszystkie informacje o instrukcjach jakie należyzawrzeć w kodzie, aby pewne rzeczy działały prawidłowo. Zdarzają się przypadki braku kompatybilności wstecznej niektórych funkcjonalności w wydawanych nowych wersjach, które pojawiają się dość często, bo nawet co 1-2 miesiące.

2.2. OpenCV

OpenCV (Open Source ComputerVisionlibrary) jest biblioteką open-source przeznaczoną na wiele platform, która dostarcza elementy dla aplikacji dot. wizji komputerowej (ang. computervision) [2]. Zapewnia wysokopoziomowe interfejsy do przechwytywania, przetwarzania i prezentacji danych obrazu. Na przykład wyodrębnia szczegóły dotyczące sprzętu kamery i alokacji macierzy. OpenCV jest szeroko stosowana zarówno w środowisku akademickim, jak i w rozwiązaniach deweloperskich [1].

Biblioteka ta została stworzona w języku C oraz C++, przez co jest mocno zoptymalizowana oraz wykorzystuje w pełni możliwości procesorów wielordzeniowych, co dla aplikacji czasu rzeczywistego jest niezwykle istotne. Obecnie rozwijana jest przy użyciu wrapperów w różnych językach programowania wśród których są:C#,Python,Ruby,Java,Matlab.

W dzisiejszych czasach wizja komputerowa może docierać do konsumentów w wielu kontekstach poprzez kamery internetowe, telefony z aparatami fotograficznymi i czujniki gamingowe, takie jak Kinect. OpenCV może pomóc w poszukiwaniu rozwiązań dla tych wymagań w języku wysokiego poziomu i w znormalizowanym formacie danych, który jest interoperacyjny z różnymi bibliotekami naukowymi. Rozbudowany zestaw narzędzi wspierających wykrywanie krawędzi oraz segmentację obrazu w bardzo skuteczny sposób potrafi wyabstrahować ze zdjęcia pożądane obiekty.

2.3. TensorflowSharp

Jest to wrapper umożliwiający wykorzystanie Tensorflow (które domyślnie zostało stworzone w Pythonie) na platformie .NET [8]. W tym przypadku jednak wrapper ten korzysta z API utworzonego w języku C. Jego źródła są dostępne publicznie, zatem aby rozpocząć korzystanie z niego wystarczy te źródła skompilować. Istnieje również możliwość zainstalowania dodatku do projektu w Visual Studio poprzez manager pakietów Nuget.

2.4. OpenCVSharp

Wrapper, dzięki któremu programiści C# mają możliwość tworzenia aplikacji zarówno mobilnych, jak i webowych lub desktopowych z użyciem OpenCV [5]. Implementuje on natywne metody OpenCV w sposób umożliwiający ich wykorzystanie przy rozpoznawaniu obiektów. Wiele klas zaimplementowanych w tym frameworku implementuje interfejs IDisposable, przez co programista nie musi martwić się o zarządzanie niebezpiecznymi zasobami pamięci. Dodatkowo OpenCVSharp nie wymusza programowania obiektowego. Wszystkie funkcje natywne można wywołać jedną prostą komendą. Do większości z nich należy przekazać obiekt typu Mat lub IplImage, które są reprezentacją obrazów przekonwertowanych w odpowiedni sposób. Problematyczne zatem jest zwrócenie z takiego bytu obrazu, który można wykorzystać do wyświetlenia rezultatu. OpenCVSharp jednak udostępnia metody BitmapFactory, dzięki którym istnieje możliwość przekonwertowania wielu obiektów na prosta bitmapę.

3. Cel i plan badań

Celem badań jest przeprowadzenie porównania metod rozpoznawania gatunków grzybów na podstawie zdjęcia. Należy potwierdzić lub odrzucić poniższą tezę:

Metoda oparta o bibliotekęTensorflow jest skuteczniejsza w rozpoznawaniu gatunków grzybów od metody opartej o OpenCV.

Do celów badawczych została utworzona aplikacja mobilna, w której zaimplementowano metody oparte o frameworkiTensorflow oraz OpenCV. Klasyfikatory/sieci szkolone były na komputerze stacjonarnym o następujących parametrach:

- procesor: AMD FX-6300 3,5 GHz,
- RAM: 8GB DDR3 2133MHz,
- karta graficzna: AMD Radeon R7 200 Series 1GB DDR5,
- dysk: SSD 60GB.

Zostały przygotowane zbiory danych dla następujących gatunków grzybów: Borowik szlachetny, Czubajka kania, Pieczarka, Muchomor czerwony, Pieprznik jadalny. Przy wyborze gatunków grzybów czynnikiem wiodącym była różnorodność budowy, rozmiarów oraz ich kolorystyki. Dla każdego z nich zostały pobrane zdjęcia w takiej liczbie, aby sumarycznie na nich oznaczyć 1000 obiektów wzorcowych.

OpenCV do rozpoznawania obiektów ze zdjęć wykorzystuje wcześniej wygenerowane pliki wzorcowe, w których zawarte są szczegółowe informację na temat danego obiektu. Aby przystąpić do procesu trenowania sieci neuronowej w przypadku Tensorflow, bądź klasyfikatora

w OpenCV, oprócz uprzednio przygotowanego zbioru zdjęć grzybów, nazywanego dalej zbiorem pozytywnym, należy przygotować również odpowiedni zbiór zdjęć negatywnych. Zdjęcia negatywne to takie na których obiekt nie występuje, ale zawierają tło na którym może on występować. Kolejnym krokiem jest utworzenie pliku .vec, który będzie zawierał obrobione zdjęcia pozytywne. Aby go stworzyć należy wykorzystać metodę opencv_createsamples przekazując następujące argumenty:

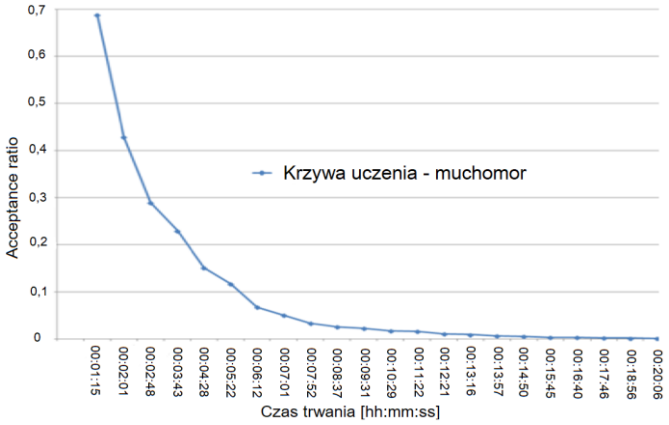
- info - ścieżka do pliku, w którym umieszczone są lokalizację zdjęć pozytywnych. Dodatkowo do każdego zdjęcia należy dodać informację o liczbie wystąpień obiektu na zdjęciu, lokalizację obiektu oraz jego rozmiar,
- num - liczba zdjęć,
- vec - ścieżka do zapisu pliku wynikowego,
- w - szerokość zdjęć pozytywnych,
- h - wysokość zdjęć pozytywnych,
- bg - ścieżka do pliku ze spisem zdjęć negatywnych.

Posiadając plik z przykładami zdjęć pozytywnych można rozpocząć pracę nad tworzeniem nowego wzorca na potrzeby identyfikacji. Należy skorzystać wówczas z metody opencv_traincascade do której należy przekazać odpowiednie argumenty:

- data - ścieżka do zapisu wyniku,
- vec - uprzednio wygenerowany plik zdjęć pozytywnych,
- bg - ścieżka do pliku ze spisem zdjęć negatywnych,
- numPos - liczba zdjęć pozytywnych,
- numNeg - liczba zdjęć negatywnych,
- numStages - liczba etapów szkolenia,
- w - szerokość zdjęć pozytywnych,
- h - wysokość zdjęć pozytywnych,
- featureType - typ cech na podstawie których tworzony będzie wzorzec,
- minHitRate - minimalne trafienie klasyfikatora podczas weryfikacji efektu szkolenia.

Najistotniejszym parametrem jest featureType, czyli rodzaj cech na podstawie których rozpoznawany jest obiekt. W tym przypadku został zastosowany typ LBP (ang. LocalBinaryPatterns)[2]. Polega on na konwersji obrazu na skalę szarości. Dla każdego piksela w obrazie w skali szarości wybierane jest sąsiedztwo o rozmiarze r, otaczające piksel środkowy. Wartość LBP jest następnie obliczana dla tego środkowego piksela i zapisywana w wyjściowej tablicy 2D z taką samą szerokością i wysokością jak obraz wejściowy. Proces progowania, gromadzenia łańcuchów binarnych i przechowywania wyjściowej wartości w tablicy jest następnie powtarzany dla każdego piksela. Wynikiem wykonania powyższych kroków jest plik wzorcowy, który jest wykorzystywany do rozpoznawania obiektów.

Szkolenie klasyfikatora opartego na OpenCV rozbijane jest na wiele etapów. Po każdym etapie w konsoli pojawia się informacja o aktualnej wartości parametru acceptanceRatio. Z przeprowadzonych testów wynika, iż najlepszym momentem na zakończenie szkolenia jest osiągnięcie wartości wcześniej wymienionego parametru w przedziale od 0,00003 do 0,00001. W celu wizualizacji wyników została utworzona krzywa uczenia dla muchomora widoczna na rysunku 1.



Rys. 1. Krzywa uczenia muchomora przy wykorzystaniu OpenCV

Jak widać w obu przypadkach wartość parametru acceptanceRatio dąży do zera. Wartość mieszcząca się w przedziale, o którym wspomniano wcześniej została osiągnięta w różnych przedziałach czasowych. Jest to spowodowane różną jakością przygotowanego zbioru fotografii dla obu gatunków.

Do identyfikacji stworzona została metoda w oparciu o wrapperOpenCVSharp [5], dzięki któremu istnieje możliwość tworzenia aplikacji oraz bibliotek w języku C# przy zachowaniu wszystkich funkcjonalności bibliotekiOpenCV. Dla każdego pliku wzorcowego wywoływana jest metoda FindMushroom, która pobrane zdjęcie przedstawia w skali szarości, następnie nakłada na zdjęcie maskę Otsu, która wyodrębnia ze zdjęcia najjaśniejsze fragmenty oraz krawędzie i tak obrobione zdjęcie przekazuje do metody DetectMultiScale (przykład1) przekazując następujące informacje:

- image - obiekt przechowujący zdjęcie,
- scaleFactor - współczynnik skalowania obrazu,
- minNeighbors - współczynnikokreślajacy minimalna liczbę nakładających się na siebie obszarów. Ustawiona np. wartość 4 określa, iż dla danego miejsca muszą zostać zwrócone minimalnie cztery nakładające się na siebie kwadraty, aby dany obszar został określony jako obiekt rozpoznany,
- flags - dodatkowy ustawienie wspierające określenie krawędzi na fotografii,
- minSize - minimalny rozmiar obiektu, który ma zostać zwracany.

Przykład 1. Metoda służąca do rozpoznawania grzybów na zdjęciu za pomocą OpenCVSharp

```
privatestaticvoidFindMushroom(File cascadeFile, Mat image,
List<Tuple<string, Rect, float>>mushrooms,
RectFmushroomRect, intoffsetY)
{
CascadeClassifiermushroom =
newCascadeClassifier(cascadeFile.AbsolutePath);
Mat ugray = new Mat();
Mat threshold = new Mat();
Cv2.CvtColor(image, ugray,
ColorConversionCodes.BGR2GRAY);
Cv2.Threshold(ugray, threshold, 0, 255,
ThresholdTypes.Otsu);

Rect[] mushroomsDetected =
mushroom.DetectMultiScale(threshold, 1.01, 4,
```

```
HaarDetectionType.FindBiggestObject,
newOpenCvSharp.Size(68, 80));

floatbestResult = 0;
string filename = cascadeFile.Name.Replace(".xml", "");
if (mushroomsDetected.Any())
mushrooms.Add(Tuple.Create(filename,
BestRect(mushroomsDetected, mushroomRect, out
bestResult, offsetY), bestResult));
}
```

W przeciwieństwie do metody opartej na OpenCV, Tensorflow do pełnego działania wymaga wygenerowania dwóch plików:

1) plik modelu (z rozszerzeniem .pb),
2) plik z nazwami obiektów (z rozszerzeniem .pbtxt).

Uprzednio przygotowany zbiór fotografii konkretnego gatunku grzyba należy odpowiednio spreparować w celu rozpoczęcia szkolenia modelu.

Pierwszym krokiem jest utworzenie pliku zawierającego wszystkie zdjęcia wzorcowe danego obiektu. Efekt ten można osiągnąć poprzez wykorzystanie metody tf.train.Example w Pythonie, która utworzy plik o rozszerzeniu .record. Po wykonaniu powyższego kroku dla wszystkich przygotowanych zbiorów fotografii należy również utworzyć w dokładnie ten sam sposób plik .record dla zbioru testowego tj. porcji fotografii wzorcowych, dzięki którym podczas szkolenia sieć neuronowa będzie mógł sprawdzać aktualną skuteczność.

Posiadając pliki z przykładami należy również utworzyć plik z nazwami oraz identyfikatorami obiektów (.pbtxt), które będą zawierały się w modelu. Zawarte w nim informacje to identyfikator obiektu oraz jego nazwa. Nazwa musi odpowiadać nazwie wykorzystanej w poprzednim kroku do oznakowania obiektów na zdjęciach. Plik ten będzie wykorzystany do trenowania sieci oraz oznaczania i opisywania znalezionych obiektów na fotografii.

Przed rozpoczęciem trenowania należy mieć przygotowany plik konfiguracyjny oraz plik “checkpoint”, czyli wstępnie wytrenowany model, który można doszkolić na swoje potrzeby. Tensorflow udostępnia przykładowe modele wraz z ich checkpointami oraz plikami konfiguracyjnymi, dzięki czemu rozpoczęcie tworzenia własnego modelu jest prostsze. Po pobraniu plików należy przystąpić do rekonfiguracji pliku .config. Należy w nim podać informacje między innymi o skalowaniu obrazu, lokalizacji pliku checkpoint, lokalizacji pliku z nazwami i identyfikatorami obiektów (.pbtxt) itp.

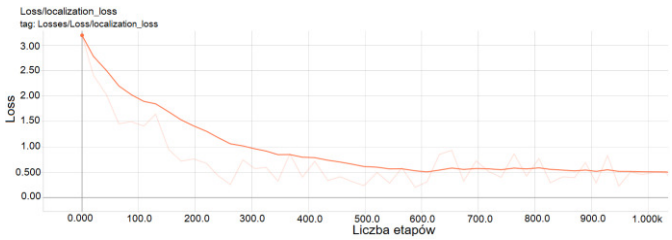
Po skompletowaniu wszystkich potrzebnych plików można rozpocząć trenowanie modelu. Tensorflow w swoich źródłach posiada plik train.py. Poniżej znajduje się przykładowe wywołanie tego pliku z poziomu linii poleceń:

train.py --logtostdderr --train_dir=path --pipeline_config_path=path

W przypadku wcześniej wspomnianej metody train.py użyta została jednokierunkowa sieć neuronowa. Jej główną cechą charakterystyczną są neurony ułożone w jednej warstwie, które są zasilane jedynie z węzłów wejściowych. Nie występują tutaj żadne sprzężenia zwrotne, tzn. każdy sygnał przekazany do sieci przechodzi przez każdy neuron tylko raz w całym swoim cyklu. Węzły te nie tworzą warstwy neuronowej, ponieważ nie zachodzą w nich żadne procesy obliczeniowe. Sieci jednokierunkowe można podzielić na: jednowarstwowe, dwuwarstwowe, wielowarstwowe. Wybór rodzaju sieci zależy od skomplikowania problemu, który ma zostać rozwiązany.

Sam proces trenowania należy kontynuować do momentu osiągnięcia parametru loss na poziomie od 0 do 1.Następnie po wytrenowaniu sieci należy skorzystać z pliku export_inference_graph.py udostępnionego przez Tensorflow. Korzysta on z utworzonego, dzięki poprzedniej metodzie, pliku checkpoint i generuje z niego model w rozszerzeniu .pb.

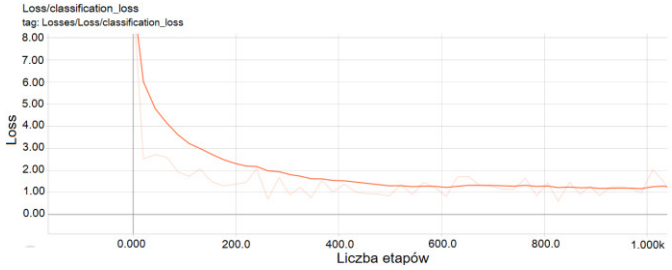
W przypadku sieci szkolnej w Tensorflow najważniejszym parametrem, który ma wpływ na jej jakość jest loss, dzięki któremu można stwierdzić jak dobrze wyszkolony jest model. Loss jest to wynik niezgodności pomiędzy przewidywaną wartością,a rzeczywistą etykietą [7]. Krzywe uczenia generowane są przy użyciu komendy tensorboard --logdirModelPath. Warto zwrócić uwagę na dwa wykresy, a mianowicie wykres przedstawiający niezgodność predykcji klasy obiektu oraz niezgodność lokalizacji obiektu.Jak można zauważyć na krzywej uczenia z rysunku 2 sieć oparta o Tensorflow najgorzej lokalizuje obiekt w początkowych etapach szkolenia. Następnie stopniowo spada, aż osiąga poziom około 0,5 loss, co jest bardzo dobrym wynikiem.



Rys. 2. Krzywa uczenia oparta o lokalizację obiektu

Świadczy to o tym, iż sieć lokalizuje obiekty bardzo trafnie. Wyniki te Tensorflow wnioskuje z porównania sieci w bieżącym stanie z przekazanym w pliku konfiguracyjnym zbiorem testowym w którym przekazane są informacje o lokalizacji obiektów znajdujących się na fotografiach.

Na rysunku 2 widać kolejne etapy szkolenia sieci i jej wyników dotyczących klasyfikowania obiektów. Są one nieco gorsze niż wyniki z krzywej uczenia z rysunku 3, co oznacza nieznacznie słabszą zdolność do rozpoznawania konkretnego gatunku.



Rys. 3. Krzywa uczenia oparta o klasyfikację obiektu

Wynik ten mógłby ulec poprawie w przypadku przekazania do sieci większej liczby zdjęć, jednak podczas jej

testowania uznano, iż jest ona satysfakcjonująca pod względem aktualnej skuteczności. Na rysunkach 2 oraz 3 widoczna jest druga linia, która również reprezentuje krzywą uczenia, jednak jest to wynik przed wygładzeniem wykresu. Opcja wygładzania jest wbudowana w TensorBoard.

Do identyfikacji stworzona została metoda w oparciu o wrapperTensorflowSharp [8] (przykład 2). Metoda ta jest dużo prostsza w implementacji, ponieważ składa się z trzech kluczowych fragmentów:

1) Utworzenie tensora na podstawie fotografii.
2) Utworzenie oraz uruchomienie runnera odpowiadającego za rozpoznanie grzybów na podstawie pliku modelu oraz pliku z opisami obiektów.
3) Skorzystanie z rezultatu w celu narysowania prostokątów na obrazie oznaczających rozpoznane obiekty.

Przykład 2. Metoda rozpoznająca gatunki grzybów przy użyciu TensorflowSharp

```
public static Bitmap Recognize(Activity activity, string
picturePath, RectFmushroomRect, out List<Tuple<string,
Rect, float>>mushrooms)
{
Activity = activity;

vargraph = newTFGraph();
varbytes =
System.IO.File.ReadAllBytes(_modelTuple.Item2.Path);
graph.Import(newTFBuffer(bytes));

varsession = newTFSession(graph);
var tensor =
ImageUtil.CreateTensorFromImageFile(picturePath,
TFDataType.UInt8);
varrunner = session.GetRunner();

runner
.AddInput(graph["image_tensor"][0], tensor)
.Fetch(
graph["detection_boxes"][0],
graph["detection_scores"][0],
graph["detection_classes"][0],
graph["num_detections"][0]);
varoutput = runner.Run();

varboxes = (float[,,])output[0].GetValue(jagged: false);
varscores = (float[,])output[1].GetValue(jagged: false);
varclasses = (float[,])output[2].GetValue(jagged: false);
varnum = (float[])output[3].GetValue(jagged: false);

return DrawBoxes(boxes, scores, classes, picturePath,
mushroomRect, out mushrooms,
MIN_SCORE_FOR_OBJECT_HIGHLIGHTING);
}
```

Jak można zauważyć metoda ta oprócz zwracania obszarów z potencjalnymi obiektami oraz ich nazwami zwraca również wynik. Wynik ten jest wyliczony bezpośrednio przez framework. W autorskiej aplikacji wynik ten jest dodatkowo sprawdzany poprzez porównanie obszarów wyznaczonych przez algorytm do obszaru narysowanego przez użytkownika podobnie jak to miało miejsce w przypadku metody opartej na OpenCV.

Po wyszkoleniu sieci/klasyfikatorów dla obu bibliotek przystąpiono do ich testowania oraz porównywania. W tym celu wykorzystano wcześniej napisaną aplikację oraz dwa telefony o różnych specyfikacjach.

Telefon 1:

- model: Huawei P20 Lite,
- RAM: 4GB,
- system: Android 8.0,
- procesor: HiSilicon Kirin 659,
- taktowanie procesora: 2360 MHz,
- aparat: 16 Mpx,
- rozdzielczość aparatu: 4608x3456 px.

Telefon 2:

- model: Xiaomi Pocophone F1,
- RAM: 6GB,
- system: Android 9.0,
- procesor: QualcommSnapdragon 845,
- taktowanie procesora: 2800 MHz,
- aparat: 12 Mpx,
- rozdzielczość aparatu: 4032x3024 px.

Badania zostały przeprowadzone przy użyciu aparatów telefonów w czterech różnych konfiguracjach. Niestety nie było możliwości skonfigurowania urządzeń do takich samych parametrów z powodu różnych wersji systemu. Zostały użyte zbliżone ustawienia, aby osiągnąć jak najbardziej porównywalne wyniki. W tym celu wybrano kadry 4:3 oraz podobne do siebie kadry 16:9 i 18:9. Dodatkowo w przypadku Huawei'a wybrane zostały rozdzielczości aparatu 5,8,10,16 MPx. W Xiaomi były do wyboru jedynie trzy opcję: słaba jakość, normalna jakość, bardzo dobra jakość. Do badania zostały wybrane dwa skrajne ustawienia oraz zmieniany był kadr. Same zdjęcia były wykonywane w takich samych warunkach otoczenia przy zachowaniu tego samego kąta ustawienia aparatu.

4. Wyniki badań

Metody zostały porównane, zwracając szczególna uwagę na następujące aspekty:skuteczność rozpoznawania,czas pracy,wpływ na zużycie zasobów.

4.1. Skuteczność rozpoznawania

Dla każdego gatunku grzyba został przygotowany zestaw 50 zdjęć zawierających jeden egzemplarz danego gatunku. Żadne z tych zdjęć nie znajdowało się wśród zdjęć pozytywnych wykorzystywanych przy szkoleniu sieci/klasyfikatorów. Są to zdjęcia w naturalnym środowisku, które zostały przed samym badaniem odpowiednio przygotowane. Poniżej przedstawiono wyniki pozytywnej weryfikacji grzyba, czyli gdy gatunek grzyba miał najlepszy wynik procentowy spośród wszystkich dostępnych.Z tabeli 1 można odczytać, iż nieznacznie lepsze wyniki osiągnęła metoda oparta na Tensorflow.

Tabela 1. Wyniki rozpoznawania gatunków grzybów przy użyciu poprawnych zdjęć

	OpenCV	**Tensorflow**
Muchomor czerwony	66%	78%
Prawdziwek	58%	74%
Kurka	48%	58%
Pieczarka	42%	44%
Kania	50%	58%

Dodatkowo w tabeli widać, iż tylko w trzech przypadkach osiągnięta została skuteczność poniżej 25 poprawnie

rozpoznanych gatunków, czyli poniżej 50%. Najlepszy wynik został osiągnięty w przypadku muchomora. Może to być wynikiem lepszego zbioru zdjęć oraz bardziej charakterystycznym wyglądem. Z drugiej strony najsłabiej wypadła pieczarka, która przez swój jednolity kolor oraz budowę zbliżoną do nie w pełni rozwiniętego muchomora lub prawdziwka często była z nimi mylona.

4.2. Czas pracy

Następnym czynnikiem, który został wzięty pod uwagę był czas jaki kluczowe metody służące do detekcji potrzebowały na obliczenie wyniku. Pomiar czasu odbywał się poprzez otoczenie kluczowej metody kodem mierzącym czas wykonania danego bloku operacji. Pomiary dla każdych ustawień urządzeń zostały wykonane dziesięciokrotnie, a następnie z otrzymanych wyników zostały wyciągnięte średnie czasy. W ten sposób osiągnięto wyniki z dokładnością do sześciu miejsc po przecinku, które następnie zostały zaokrąglone do czterech miejsc po przecinku. W tabelach 2 oraz 3 zostały przedstawione średnie czasy działania metody opartej na OpenCV w wybranych konfiguracjach aparatu.

Tabela 2. Czas działania metody opartej na OpenCV na telefonie marki Huawei

	3264 x 1616 18:9 (Aparat)	3264 x 2448 4:3 (Aparat)	4608 x 2272 18:9 (Aparat)	4608 x 3456 4:3 (Aparat)
Czas działania[s]	10,1720	9,9728	9,6862	10,2009

Tabela 3. Czas działania metody opartej na OpenCV na telefonie marki Xiaomi

	Słaba jakość 4:3 (Aparat)	Dobra jakość 4:3 (Aparat)	Słaba jakość 16:9 (Aparat)	Dobra jakość 16:9 (Aparat)
Czas działania[s]	4,4345	4,5751	4,5028	4,8938

W zestawieniu dużo słabiej wypadło urządzenie Huawei, najprawdopodobniej ze względu na słabsze parametry sprzętowe. Nie miało to jednak wpływu na wyniki zwracane przez metody, które zostaną omówione w kolejnej części badania. Dodatkowo widać zależność rosnącej jakości oraz rozmiaru zdjęcia do czasu działania metody. Wraz z coraz mocniejszymi ustawieniami aparatu wzrastał również czas analizy fotografii. Wyniki mogły być delikatnie zakłócone poprzez aplikacje systemowe działające w tle.

Następnie przy użyciu tych samych parametrów i przy użyciu tego samego zdjęcia wzorcowego zbadano metodę opartą na Tensorflow. Zmierzone czasy zostały przedstawione w tabelach 4 i 5.

Tabela 4. Czas działania metody opartej na Tensorflow na telefonie marki Huawei

	3264 x 1616 18:9 (Aparat)	3264 x 2448 4:3 (Aparat)	4608 x 2272 18:9 (Aparat)	4608 x 3456 4:3 (Aparat)
Czas działania[s]	11,3520	11,7354	11,6489	12,5733

Tabela 5. Czas działania metody opartej na Tensorflow na telefonie marki Xiaomi

	Słaba jakość 4:3 (Aparat)	Dobra jakość 4:3 (Aparat)	Słaba jakość 16:9 (Aparat)	Dobra jakość 16:9 (Aparat)
Czas działania[s]	4,9456	5,4237	5,3477	6,0123

Rezultaty potwierdzają wcześniej stwierdzony fakt korelacji pomiędzy wielkością zdjęcia, a czasem działania. Czasy są większe niż w przypadku rozwiązania opartego o OpenCV. Dodatkowym aspektem, który warto zaznaczyć jest spełnienie w przypadku Xiaomi początkowego założenia dotyczącego maksymalnego czasu działania metody, który wynosił 10 sekund. W przypadku Huaweia granica ta została delikatnie przekroczona, jednak uśredniając wszystkie wyniki średnia mieści się w normie. Ponadto Huawei dysponował lepszym aparatem tylnym niż Xiaomi co zwiększało szczegółowość zdjęcia oraz jego rozmiar co mogło wpłynąć na czas działania.

4.3. Wpływ na zużycie zasobów

Zużycie zasobów podczas tworzenia i testowania metod można podzielić na dwie kategorie:

1) Zużycie zasobów komputera podczas szkolenia algorytmu.
2) Zużycie zasobów telefonu podczas rozpoznawania obiektów na zdjęciach.

Podczas nauki sieci/klasyfikatorów opartych na OpenCV w zależności od momentu szkolenia w danym etapie następował wzrost obciążenia poszczególnych podzespołów. Podczas ładowania zdjęć pozytywnych i negatywnych elementami, które zostały najbardziej obciążone był procesor, dysk oraz nieznaczny wzrost pamięci RAM. Zużycie CPU wzrastało momentami do 100%, podobnie było w przypadku dysku, jednak wzrosty były na tyle krótkotrwałe, że nie wpływały w znaczącym stopniu na pracę komputera. Po załadowaniu zdjęć następowało właściwe szkolenie podczas którego wzrastało zużycie pamięci RAM i w tym przypadku trwało aż do zakończenia etapu. Maksymalne zużycie jest definiowane parametrycznie i jego domyślna wartość to 1GB. Metoda oparta na Tensorflow po uruchomieniu trenowania i wewnętrznej konfiguracji od razu przystępuje do szkolenia. Przy użyciu domyślnych ustawień framework ten pobiera tyle zasobów komputera ile tylko jest w stanie zaalokować. Efektem tego jest bardzo mocna praca komputera oraz duże trudności w innych działaniach.

Do celów pomiarowych została wykorzystana aplikacja “Monitor zasobów Mini”, która w czytelny sposób wyświetla na ekranie telefonu aktualnie dostępne zasoby sprzętowe. Pomiary zostały wykonane dziesięciokrotnie dla każdych ustawień urządzeń, z których została wyciągnięta średnia. Przy rozpoznawaniu obiektów za pomocą OpenCV odnotowano wzrosty zużycia zasobów na telefonach widoczne w tabelach 6 oraz 7.

Tabela 6. Wyniki zużycia zasobów na telefonie marki Huawei przy użyciu metody opartej na OpenCV

	3264 x 1616 18:9 (Aparat)	3264 x 2448 4:3 (Aparat)	4608 x 2272 18:9 (Aparat)	4608 x 3456 4:3 (Aparat)
RAM[MB]	1300	1250	1310	1290
CPU	30%	60%	64%	65%

Tabela 7. Wyniki zużycia zasobów na telefonie marki Xiaomi przy użyciu metody opartej na OpenCV

	Słaba jakość 4:3 (Aparat)	Dobra jakość 4:3 (Aparat)	Słaba jakość 16:9 (Aparat)	Dobra jakość 16:9 (Aparat)
RAM[MB]	1200	1240	1400	1370
CPU	70%	65%	73%	75%

Przy rozpoznawaniu obiektów za pomocą Tensorflow odnotowano wzrosty zużycia zasobów na telefonach widoczne w tabelach 8 oraz 9.

Tabela 8. Wyniki zużycia zasobów na telefonie marki Huawei przy użyciu metody opartej na Tensorflow

	3264 x 1616 18:9 (Aparat)	3264 x 2448 4:3 (Aparat)	4608 x 2272 18:9 (Aparat)	4608 x 3456 4:3 (Aparat)
RAM[MB]	1320	1330	1310	1330
CPU	55%	62%	64%	71%

Tabela 9. Wyniki zużycia zasobów na telefonie marki Xiaomi przy użyciu metody opartej na Tensorflow

	Słaba jakość 4:3 (Aparat)	Dobra jakość 4:3 (Aparat)	Słaba jakość 16:9 (Aparat)	Dobra jakość 16:9 (Aparat)
RAM	1600MB	1650MB	1710MB	1750MB
CPU	72%	73%	77%	80%

Z otrzymanych wyników zauważyć można, iż oprogramowanie zabiera większość dostępnych zasobów w celu analizy przekazanej fotografii. Wraz ze wzrostem jakości zdjęcia zauważalny jest wzrost wykorzystania zasobów sprzętowych. Różnica pomiędzy urządzeniami wynika głównie z różnicy ich specyfikacji. W przypadku Xiaomi istniała możliwość pobrania większej ilości pamięci RAM oraz większego wykorzystania procesora. Aplikacja w momencie rozpoznania była nieużywalna, jednak samo działanie urządzenia nie zostało zakłócone w znaczącym stopniu. Metoda stworzona w Tensorflow potrzebowała nieznacznie więcej zasobów.

5. Wnioski

Postawioną tezę:

Metoda oparta o bibliotekęTensorflow jest skuteczniejsza w rozpoznawaniu gatunków grzybów od metody opartej o OpenCV,

udało się udowodnić. W przypadku najważniejszego aspektu badawczego, czyli skuteczności rozpoznawania gatunków grzybów Tensorflow okazał się lepszy o 9%. Jednym z powodów tego stanu rzeczy może być fakt, iż Tensorflow korzysta ze zbioru testowego podczas trenowania sieci neuronowej, który zostaje ułożony przez programistę przed trenowaniem. Sprawia to, iż aktualny stan rozwoju sieci jest porównywany do dobrze dobranego zbioru wzorcowego. Dodatkowo Tensorflow jest zdecydowanie świeższym rozwiązaniem, ale jednocześnie wystarczająco dojrzałym, przez co posiada szeroką gamę funkcji, które usprawniają pracę oraz zapewniają lepsze rezultaty.

Warto jednak zwrócić uwagę nie tylko skuteczność metody, ale również na inne cechy, które zostały zbadane. Proces rozpoznawania gatunku grzyba trwał 1 do 2 sekund dłużej w przypadku Tensorflow. Czas trwania spowodował, że na urządzeniu o słabszej specyfikacji sprzętowej została przekroczona granica dziesięciu sekund wyznaczona w wymaganiach niefunkcjonalnych. Dodatkowym atutem przemawiającym za OpenCV jest mniejszy pobór zasobów urządzenia, na którym dokonuje się detekcji gatunku grzyba. Jest to bardzo ważny argument przemawiający na korzyść tego frameworka, ponieważ istnieje możliwość uruchomienia aplikacji z tą metodą na telefonach bądź komputerach o słabszej specyfikacji bez obawy o stabilność działania programu. Powodem tego stanu rzeczy może być fakt, iż pliki z których korzysta OpenCV podczas analizy obrazu są zdecydowanie mniejsze niż pliki modelu Tensorflow. Pliki te podczas detekcji są ładowane do pamięci, przez co jej zużycie znacząco wzrasta.

Literatura

[1] Baggio D. L.: Mastering OpenCV with Practical Computer Vision Projects, Packt Publishing Ltd, 2012.

[2] Bradski G., Kaehler A.: Learning OpenCV, O'Reilly 2008.

[3] He K., Zhang X., Ren S., Sun J.: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, s. 770-778.

[4] Krzyśko M., Wołyński W., Górecki T., Skorzybut M.: Systemy uczące się, Wydawnictwo Naukowo-Techniczne Sp. z o.o., 2008.

[5] OpenCVSharp, https://github.com/shimat/OpenCVsharp [07.05.2019].

[6] Staniszewski P.: Użytkowanie grzybów leśnych – praktyka i problemy badawcze, Stud. Mater. CEPL w Rogowie, 2014, s. 143 – 152.

[7] Tang J.: Intelligent Mobile Projects with Tensorflow, Packt Publishing Ltd. 2018.

[8] TensorflowSharp, https://github.com/migueldeicaza/TensorflowSharp [11.05.2019].