Studenckie Koło Naukowe Linuksa i Wolnego Oprogramowania

Projekt: **Przetwarzanie zdjęć pozyskanych z kamery termowizyjnych**

Autorzy: inż. Tomasz Skowron, inż. Wojciech Król, inż. Dominik Nowocień

Konsultanci: mgr inż. Dariusz Marek, mgr inż. Jakub Szyguła

Spis treści:

1. Wstęp teoretyczny

Początki uczenia maszynowego sięgają lat sześćdziesiątych dwudziestego wieku, kiedy na uniwersytecie w Stanford pierwszy raz wykorzystano odkrycia dokonane przez komputer, w prasie naukowej. Przez kolejne dekady komputery wykorzystujące uczenie maszynowe były stosowane w wielu systemach, do rozwiązywania rozmaitych problemów. Jednak dopiero w końcówce lat dziewięćdziesiątych postępy na polu uczenia maszynowego nabrały tempa, głównie za sprawą popularyzacji internetu oraz, co za tym idzie, przeglądarek internetowych. Na początku dwudziestego pierwszego wieku komputery, w których zastosowano algorytmy uczenia maszynowego, zaczęły wygrywać z mistrzami szachowymi oraz zwyciężać w turniejach. Ostatnia dekada to znaczny rozwój badań w dziedzinie uczenia maszynowego, głównie za sprawą coraz to większej mocy obliczeniowej oraz coraz to łatwiejszemu dostępowi do niej. Skrypty zaprezentowane w dalszej części pracy przetwarzają dziesiątki tysięcy zdjęć, w krótkim czasie, na komputerze domowym, co dekadę temu było by zdecydowanie utrudnione.

Obecnie największą popularnością cieszą się zastosowania sieci neuronowych przy problemach związanych z klasyfikacją obrazów. Przyczynia się do tego trudność w tworzeniu satysfakcjonujących algorytmów heurystycznych, namacalność wyników

pracy oraz zdecydowanie lepsze wyniki niż te osiągane tradycyjnymi metodami. Spośród licznych rozwiązań wykorzystujących sztuczną inteligencję przytoczyć można rozpoznawanie znaków w tekście, włączając w to pismo odręczne. Wartym uwagi jest również projekt, który dzięki sieciom neuronowym, połączonym z licznymi sensorami chemicznymi tworzy “sztuczny nos” imitujący zachowanie nosów ssaków. Na szczególną uwagę zasługują zastosowania sztucznej inteligencji w dziedzinie medycyny, co może w znaczącym stopniu pomóc w przyspieszeniu procesów diagnostycznych. Kolejne przykłady można by długo wymieniać, przechodząc przez obszar finansów, bezpieczeństwa, transportu i wiele innych [6].

2. Cel projektu

Celem projektu „Przetwarzanie zdjęć pozyskanych z kamery termowizyjnych” jest wykonanie wstępnego przetworzenia obrazów i przygotowanie ich do wykorzystania w procesie uczenia sieci neuronowej. Wstępna analiza, standaryzacja rozmiaru zdjęć, systematyzacja nazewnictwa oraz ujednolicenie formatów jest kluczowym zabiegiem powszechnie stosowanym w detekcji i klasyfikacji obiektów na zdjęciach [4]. O znaczeniu odpowiedniego przygotowania zbioru danych dla algorytmów klasyfikacji przed ich użyciem w przypadku ogólnym traktuje artykuł [3]. W przypadkach, w których przedmiotem rozważań są obrazy, przetwarzanie wstępne poza aspektami ujednolicania zbioru i przystosowania go pod strukturę sieci neuronowej, stosuje się również do augmentacji zbiorów, by przeciwdziałać zbytniemu dopasowaniu sieci do danych [7]. Czasami przygotowanie danych może również opierać się o wykonywanie zmian w zdjęciach, które powodują utratę zbędnych informacji, tak by polepszyć wyniki otrzymane w etapie klasyfikacji [2]. Stosuje się szeroko rozumiane algorytmy wykorzystujące maski, progowanie, detekcję krawędzi, grupowanie etc. z satysfakcjonującymi efektami. W tym projekcie skupiono się jednak na algorytmach przetwarzania zdjęć, które opierały się o sztuczną inteligencję.

3. Przebieg projektu

Prace nad projektem rozpoczęto od analizy problemu polegającej na zapoznaniu się z danymi oraz przeprowadzeniu dyskusji na temat możliwych rozwiązań. Zdecydowano się skupić na wykorzystaniu algorytmów *RetinexNet* [1] oraz *Intel® Open Image Denoise* [5]. Oba wspomniane rozwiązania korzystają z sieci neuronowych. Poddanie obrazu działaniu sieci *RetinexNet* skutkuje jego rozjaśnieniem. Tymczasem Open Image Denoiser usuwa szumy ze zdjęcia. Podjęto

również analizę otrzymanego zbioru danych z kamery termowizyjnej, który jak się okazało nie wymagał większych nakładów pracy nad jego standaryzacją. Ponadto zbiór zawierał około 17.000 zdjęć i był dobrze dostosowany do sieci neuronowych, bowiem zapewniał zarówno zdjęcia z kamery termowizyjnej, jak i kamery RGB, a wymiary zdjęć były już ujednolicone. Jednakże koniecznym okazało się napisanie skryptu konwertującego istniejące zdjęcia na format 24 – bitowy, który nie został przewidziany przez twórców zbioru. Analogicznie wykonano również skrypt odwracający konwersję na powrót do ośmiu bitów. Na potrzeby uczenia modeli sieci neuronowych pojawiła się również potrzeba konwersji metadanych zapewnionych w zbiorze do formatu obsługiwanego przez algorytmy detekcji i klasyfikacji obiektów. W celu zrealizowania założeń postanowiono wykorzystać środowisko *Python* oraz dodatkową bibliotekę *opencv* [8].

Pierwszą metodą zastosowaną do zbioru zdjęć było ich przetworzenie algorytmami heurystycznymi. W tym celu napisano odpowiednie skrypty w języku *Python*. Podjęto próbę poddania zdjęć algorytmom progującym, ale otrzymane w wyniku tego zdjęcia bardzo często traciły znaczące informacje o obiektach, które sieć ma wykrywać. Niemniej, zdarzały się również obrazy, w których udawało się usunąć znaczne części tła i nieinteresujących nas szumów. Niestety na algorytmie tym nie można było polegać, a próby ulepszenia go poprzez dostosowywanie progu do średniej wartości piksela na zdjęciu nie dawały ani satysfakcjonujących rezultatów, ani nadziei na jakąkolwiek poprawę działania sieci neuronowych względem zbioru nieprzetworzonego.

Obraz oryginalny.	Po zastosowaniu progowania.
	
	

Następnie przystąpiono do testowania algorytmu RetinexNet. Niezgodności formatów wymusiły wykorzystanie skryptów konwertujących dane. Otrzymane na wyjściu z sieci *RetinexNet* zdjęcia zgodnie z oczekiwaniami były rozjaśnione, a niektóre aspekty zdjęć stawały się tym samym bardziej odróżniające od innych.

Obraz oryginalny.	Po zastosowaniu RetinexNet.

RetinexNet uruchomiono dla całego zbioru danych, a następnie poddano go działaniu przygotowanej sieci neuronowej opartej o algorytm *YOLO* i otrzymano następujące wyniki dokładności:

Porównanie *Mean Average Precision* dla różnych zbiorów.	
Zbiór nieprzetworzony	Zbiór przetworzony z użyciem *RetinexNet*
78.23%	77.09%

Porównanie dokładności wykrywania obiektów dla różnych zbiorów.		
Klasa	Zbiór nieprzetworzony	Zbiór przetworzony z użyciem *RetinexNet*
Osoba	83.38%	82.13%
Rower	62.64%	60.89%

Samochód	88.68%	88.26%

Otrzymane wyniki z wykorzystaniem *RetinexNet* nie były lepsze od zbioru nieprzetworzonego. Podejrzewać można, że rozjaśnianie czarno-białych zdjęć tą metodą może powodować utratę części informacji ułatwiających sieci wykrywanie obiektów. Niemniej różnice w dokładności okazały się nieduże.

Analogicznie napisano odpowiedni skrypt dla algorytmu *Open Image Denoise*. Z wykorzystaniem konwerterów udało się otrzymać zbiór zdjęć o zredukowanym szumie, zgodnie z założeniami. Działanie *Open Image Denoise* na zdjęciach z kamery termowizyjnej nie było w żadnym stopniu spektakularne, a efektów działania trzeba się było doszukiwać. Niemniej stwierdzono, w szczególności na obszarach „tła” wyrównanie wartości pikseli.

Obraz oryginalny.	Po zastosowaniu Open Image Denoiser.
	
	

Na tak przygotowanym zbiorze podjęto próbę nauczenia sieci neuronowej. Wyniki okazały się gorsze niż przewidywano.

Porównanie *Mean Average Precision* dla różnych zbiorów.	
Zbiór nieprzetworzony	Zbiór przetworzony z użyciem *Intel Open Image Denoiser*
78.23%	79.49%

Porównanie dokładności wykrywania obiektów dla różnych zbiorów.		
Klasa	Zbiór nieprzetworzony	Zbiór przetworzony z użyciem *Intel Open Image Denoiser*
Osoba	82.38%	82.46%
Rower	62.64%	66.94%
Samochód	88.68%	89.08%

Zakładano, że usunięcie zupełnie zbędnych różnic w wartościach pikseli spowoduje poprawę jakości detekcji. Jakość wzrosła nieznacznie względem zbioru nieprzetworzonego, podobnie zresztą jak w przypadku *RetinexNet*.

Zbiór danych został poddany augmentacji. Wykonano duplikaty zdjęć, z czego jedna kopia została symetrycznie odbita w osi Y. Przygotowany został odpowiedni skrypt, który realizował to zadanie. Metadane dla sieci neuronowej przechowywane są w osobnych plikach *.txt* i również wymagały dostosowania do nowych zdjęć. Aby to zrealizować napisano odpowiedni skrypt.

4. Specyfikacja skryptów

Podczas pracy nad projektem napisano kilka skryptów pozwalających na realizację założeń projektowych. Skrypty zostały napisane w języku python [9].

Brak zgodności warstw wejściowych niektórych sieci neuronowych spowodowała, że koniecznym okazało się napisanie skryptu konwertującego zdjęcia z 8 na 24 bitowe. Należy zauważyć, że dzięki skryptowości języka python, do pewnych sieci skrypt ten mógł być wykorzystywany również do zmiany nazwy plików. Tak też został wykorzystany w wypadku sieci RetinexNet, która na wyjściu zwraca obrazy ze

zmienioną nazwą. Skrypt wykorzystuje znajdujący się w katalogu roboczym folder input do pobierania zdjęć, a następnie umieszcza je w katalogu output.

```
import os;
from PIL import Image;
import numpy as np;
files =os.listdir(os.getcwd()+"\\input");
files;
for file in files:
    im = Image.open(os.getcwd()+"\\input\\"+file);
    array = np.asarray(im);
    copy = [];
    for i in range(0,len(array)):
        copy.append([]);
        for j in range(0,len(array[i])):
            val = array[i][j];
            copy[i].append([]);
            copy[i][j].append(val);
            copy[i][j].append(val);
            copy[i][j].append(val);

    img = Image.fromarray(np.asarray(copy));
    img.save(os.getcwd()+"\\output\\"+file);
```

Skrypt *conver8to24bit.py*

Analogicznie do konwersji z 8 na 24 bity, napisano konwerter skierowany w stronę przeciwną. Skrypt ten powstał celem uniknięcia rozbieżności formatów wejściowych zdjęć podczas uczenia sieci neuronowej. Skrypt wykorzystuje znajdujący się w katalogu roboczym folder input do pobierania zdjęć, a następnie umieszcza je w katalogu output.

```
import os;
from PIL import Image;
import numpy as np;
files =os.listdir(os.getcwd()+"\\input");
files;
for file in files:
    im = Image.open(os.getcwd()+"\\input\\"+file);
    array = np.asarray(im);
    copy = [];
    for i in range(0,len(array)):
        copy.append([]);
        for j in range(0,len(array[i])):
            val = array[i][j][0];
            copy[i].append(val);
```

```
    img = Image.fromarray(np.asarray(copy));
    img.save(os.getcwd()+"\\output\\"+file);
```

Skrypt *convert24to8bit.py*

Podczas pracy z Open Image Denoiser koniecznym okazało się napisanie skryptu systemowego w języku *batch*. Pozwolił on na wykonanie odszumiania na zbiorze zdjęć w danym folderze. W skrypcie należy dostosować ścieżkę do lokalizacji danych.

```
SET FILE_EXTENSION=jpeg
SET PATH_TO_DENOISER=E:\ML_preprocessed_photos\Denoiser_v1.3\Denoiser_v1.3
SET PATH_TO_NEW_FOLDER=..\denoised_val
if not exist "PATH_TO_NEW_FOLDER" mkdir %PATH_TO_NEW_FOLDER%

for /r %%v in (*.%FILE_EXTENSION%) do %PATH_TO_DENOISER%\Denoiser.exe -i
"%%~nv.%FILE_EXTENSION%" -o
"%PATH_TO_NEW_FOLDER%\%%~nv.%FILE_EXTENSION%"

cmd /k
```

Skrypt *denoise.bat*

Pracując z algorytmami progowania wykonano skrypt *threshold.py* w języku python. Uruchomienie go pozwala na poddanie zdjęć wejściowych z folderu *input* działaniu algorytmu progowania i zapisanie wyniku w folderze *output*. Algorytm ten bierze pod uwagę średnią wartość piksela, by lepiej dostosować wartość progu do obcięcia. Niestety wyniki działania tego algorytmu nie są satysfakcjonujące.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2

# Display one image
def display_one(a, title1 = "Original"):
    plt.imshow(a), plt.title(title1)
    plt.xticks([]), plt.yticks([])
    plt.show()
# Display two images
def display(a, b, title1 = "Original", title2 = "Edited"):
    plt.subplot(121), plt.imshow(a), plt.title(title1)
    plt.xticks([]), plt.yticks([])
    plt.subplot(122), plt.imshow(b), plt.title(title2)
    plt.xticks([]), plt.yticks([])
    plt.show()

def calculateMean(src):
    sum = 0;
```

```
    for item in src:
        sum+=item;
    return sum/len(src);

files = os.listdir(os.getcwd()+"\\input\\");
print(files);

imgs = [];
for i in range(0,len(files)):
    file = files[i];
    print(file);
    if file.split(".")[-1]=="jpg" or file.split(".")[-1]=="jpeg":
        #display_one(a=file,title1="Original")
        img = [cv2.imread(os.getcwd()+"\\input\\"+file,cv2.IMREAD_UNCHANGED)]
        #display_one(img);
        print(os.getcwd()+"\\input\\"+file);
        height = 640
        width = 512
        dim = (height,width)
        res_img = []
        for j in range(len(img)):
            res = cv2.resize(img[j], dim, interpolation=cv2.INTER_LINEAR)
            res_img.append(res)
        imgs.append([]);
        imgs[i].append(res_img[0])


#for i in range(0,len(imgs)):
    print("i"+str(i));
    image = imgs[i][0];
    blurred = cv2.medianBlur(image,3)
    histr = cv2.calcHist([image],[0],None,[256],[0,256]);
    meanPx = calculateMean(image.ravel());
    ret, thresh = cv2.threshold(blurred, 127-(abs(meanPx-127)), 127,
cv2.THRESH_TOZERO)
    ret, thresh = cv2.threshold(thresh, 255-(abs(meanPx-127)), 255,
cv2.THRESH_TOZERO_INV)
    #display(image, thresh, 'Original', 'Segmented');
    cv2.imwrite(os.getcwd()+"\\output\\"+files[i], thresh)
```

Skrypt *threshold.py*

Do augmentacji zbiorów napisano dwa skrypty w języku *Python*. Jeden odpowiedzialny za powiększanie zbioru danych i drugi zajmujący się generowaniem odpowiednich metadanych dla nowych zdjęć. Augmentacja zdjęć odbywa się przez wczytywanie kolejnych zdjęć z folderu *input* i wstawianie nowych do folderu *output* z dodaną na końcu literą ‘a’ w nazwie pliku. Analogicznie metadane umieszczone w folderze *input_meta* są analizowane, a powstałe pliki wynikowe zapisywane są w folderze *output_meta.*

```
import os;
from PIL import Image;
import PIL;
import numpy as np;
```

```
files =os.listdir(os.getcwd()+"\\input");
files;
for file in files:
    print("Processing file: "+file);
    im = Image.open(os.getcwd()+"\\input\\"+file);
    im2 = im.transpose(PIL.Image.FLIP_LEFT_RIGHT);
    name = file.split('.')[0];
    name2 = name+"a";
    im2.save(os.getcwd()+"\\output\\"+name2+'.'+file.split('.')[1]);
    im.save(os.getcwd()+"\\output\\"+file);
```

Skrypt *doAugmentation.py*

```
import os;
import numpy as np;
files =os.listdir(os.getcwd()+"\\input_meta");
files;
for file in files:
        print("Opening: "+os.getcwd()+"\\input_meta\\"+file);
        fp = open(os.getcwd()+"\\input_meta\\"+file);
        line = fp.readline();
        outLines = [];
        newLines = [];
        while line:
                items = line.split(' ');
                left = float(items[1]);
                left = 1-left;
                items[1] = str(left);
                newLine = items[0]+" "+items[1]+" "+items[2]+" "+items[3]+" "+items[4];
                newLines.append(newLine);
                outLines.append(line);
                line = fp.readline();
        fp.close();
        print("Opening: "+os.getcwd()+"\\output_meta\\"+file);
        fp = open(os.getcwd()+"\\output_meta\\"+file,"w");
        for line in outLines:
                fp.write(line);
        fp.close();
        newFileName = file.split('.')[0]+'a.'+file.split('.')[1];
        print("Opening: "+os.getcwd()+"\\output_meta\\"+newFileName);
        fp = open(os.getcwd()+"\\output_meta\\"+newFileName,"w");
        for line in newLines:
                fp.write(line+'\n');
        fp.close();
```

Skrypt *doAugmentation_meta.py*

5. Podsumowanie

Projekt „Przetwarzanie zdjęć pozyskanych z kamery termowizyjnych" dostarczył wielu interesujących spostrzeżeń. Okazało się, że niektóre sieci neuronowe mogą wspierać proces wstępnego przetwarzania zdjęć w procesie trenowania modeli opartych o sztuczną inteligencję. Niemniej testowane podczas projektu rozwiązania nie przyniosły efektów na tyle satysfakcjonujących, by rekompensowały czas poświęcony na ich przygotowanie. Z drugiej strony zauważono przykłady, w których efekt takiego podejścia może okazać się przeciwny do oczekiwanego. Zważając na doświadczenia zdobyte podczas realizacji projektu stwierdzono, że zastosowanie języka *Python* do pisania prostych skryptów do przetwarzania obrazu jest bardzo łatwe i szybkie.

Bibliografia

[1]	Chen Wei. RetinexNet https://github.com/weichen582/RetinexNet
[2]	Hussain Dar, Nasir. (2020). Image segmentation Techniques and its application.
[3]	Hongjun Lu Sam Yuan Sung Ying Lu. On Preprocessing Data for Effective Classification (1996)
[4]	Islam Hasabo. Image Classification using Machine Learning and Deep Learning https://medium.com/swlh/image-classification-using-machine-learning-and-deep-learning-2b18bfe4693f [data dostępu: 19.12.2020]
[5]	Intel® Open Image Denoise https://www.openimagedenoise.org/documentation.html [data dostępu: 19.12.2020]
[6]	Kadurumba, Chukwuma & Nwaiwu, Uchechukwu & Nwasuka, Nnamdi. (2020). Neural Network Applications.

[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, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens,Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker,Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
[8]	OpenCV documentation. https://docs.opencv.org/master/ [data dostępu: 19.12.2020]
[9]	Van Rossum, G., & Drake Jr, F. L. (1995). Python reference manual. Centrum voor Wiskunde en Informatica Amsterdam.