

Nagłówek sprawozdania

Bartosz Psik 325 211 09.05.2024r 14.15-17.00

Analiza obrazu - detekcja twarzy

Inicjalizacja

import bibliotek, pobranie do lokalnego środowiska wykonawczego przykładowego obrazu testowego i modeli dla detektorów twarzy

```
import cv2
from google.colab.patches import cv2_imshow

import dlib

# biblioteka insightface wymaga zainstalowania
!pip install insightface
!pip install onnxruntime

import insightface

Collecting insightface
  Downloading insightface-0.7.3.tar.gz (439 kB)
  ━━━━━━━━━━━━━━━━ 439.5/439.5 kB 3.0 MB/s eta
0:00:00
    ents to build wheel ... etadata (pyproject.toml) ... ent already
    satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
    insightface) (1.25.2)
Collecting onnx (from insightface)
  Downloading onnx-1.16.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (15.9 MB)
  ━━━━━━━━━━━━━━━━ 15.9/15.9 MB 42.3 MB/s eta
0:00:00
    ent already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
    (from insightface) (4.66.4)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from insightface) (2.31.0)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from insightface) (3.7.1)
Requirement already satisfied: Pillow in
/usr/local/lib/python3.10/dist-packages (from insightface) (9.4.0)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from insightface) (1.11.4)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.10/dist-packages (from insightface) (1.2.2)
Requirement already satisfied: scikit-image in
```

```
/usr/local/lib/python3.10/dist-packages (from insightface) (0.19.3)
Requirement already satisfied: easydict in
/usr/local/lib/python3.10/dist-packages (from insightface) (1.13)
Requirement already satisfied: cython in
/usr/local/lib/python3.10/dist-packages (from insightface) (3.0.10)
Requirement already satisfied: albumentations in
/usr/local/lib/python3.10/dist-packages (from insightface) (1.3.1)
Requirement already satisfied: prettytable in
/usr/local/lib/python3.10/dist-packages (from insightface) (3.10.0)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from albumentations->insightface) (6.0.1)
Requirement already satisfied: qudida>=0.0.4 in
/usr/local/lib/python3.10/dist-packages (from albumentations->insightface) (0.0.4)
Requirement already satisfied: opencv-python-headless>=4.1.1 in
/usr/local/lib/python3.10/dist-packages (from albumentations->insightface) (4.9.0.80)
Requirement already satisfied: networkx>=2.2 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->insightface) (3.3)
Requirement already satisfied: imageio>=2.4.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->insightface) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->insightface) (2024.4.24)
Requirement already satisfied: PyWavelets>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->insightface) (1.6.0)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->insightface) (24.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->insightface) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->insightface) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->insightface) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->insightface) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->insightface) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib->insightface)
(2.8.2)
Requirement already satisfied: protobuf>=3.20.2 in
/usr/local/lib/python3.10/dist-packages (from onnx->insightface)
(3.20.3)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.10/dist-packages (from prettytable-
>insightface) (0.2.13)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->insightface)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->insightface)
(3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->insightface)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->insightface)
(2024.2.2)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn-
>insightface) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn-
>insightface) (3.5.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib->insightface) (1.16.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from qudida>=0.0.4-
>albumentations->insightface) (4.11.0)
Building wheels for collected packages: insightface
  Building wheel for insightface (pyproject.toml) ... e=insightface-
0.7.3-cp310-cp310-linux_x86_64.whl size=1054140
sha256=918f61e2b31f841a88f93a04d41916b532f4ea07210975c304439dc38ecaebe
9
  Stored in directory:
/root/.cache/pip/wheels/e3/d0/80/e3773fb8b6d1cca87ea1d33d9b1f20a223a64
93c896da249b5
Successfully built insightface
Installing collected packages: onnx, insightface
Successfully installed insightface-0.7.3 onnx-1.16.0
Collecting onnxruntime
  Downloading onnxruntime-1.17.3-cp310-cp310-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (6.8 MB)
6.8/6.8 MB 23.6 MB/s eta
0:00:00
onnxruntime)
```

```
  Downloading coloredlogs-15.0.1-py2.py3-none-any.whl (46 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.0/46.0 kB 6.6 MB/s eta
0:00:00
ent already satisfied: flatbuffers in /usr/local/lib/python3.10/dist-
packages (from onnxruntime) (24.3.25)
Requirement already satisfied: numpy>=1.21.6 in
/usr/local/lib/python3.10/dist-packages (from onnxruntime) (1.25.2)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from onnxruntime) (24.0)
Requirement already satisfied: protobuf in
/usr/local/lib/python3.10/dist-packages (from onnxruntime) (3.20.3)
Requirement already satisfied: sympy in
/usr/local/lib/python3.10/dist-packages (from onnxruntime) (1.12)
Collecting humanfriendly>=9.1 (from coloredlogs->onnxruntime)
  Downloading humanfriendly-10.0-py2.py3-none-any.whl (86 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━ 86.8/86.8 kB 13.2 MB/s eta
0:00:00
ent already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->onnxruntime) (1.3.0)
Installing collected packages: humanfriendly, coloredlogs, onnxruntime
Successfully installed coloredlogs-15.0.1 humanfriendly-10.0
onnxruntime-1.17.3

import time

# obraz testowy
# https://drive.google.com/file/d/1dKxVJ00n506VuiHDdvrS36touzAeCgbw/
view?usp=sharing
!gdown 1dKxVJ00n506VuiHDdvrS36touzAeCgbw

Downloading...
From: https://drive.google.com/uc?id=1dKxVJ00n506VuiHDdvrS36touzAeCgbw
To: /content/2_Demonstration_Demonstration_Or_Protest_2_58.jpg
  0% 0.00/173k [00:00<?, ?B/s] 100% 173k/173k [00:00<00:00, 80.7MB/s]

# model detektora Haar
# https://drive.google.com/file/d/1k2J4-4yIXrFFHZUhFv4pCRqcl-wB2Ve9/
view?usp=sharing
!gdown 1k2J4-4yIXrFFHZUhFv4pCRqcl-wB2Ve9
!unzip haarcascade_frontalface_default.zip

Downloading...
From: https://drive.google.com/uc?id=1k2J4-4yIXrFFHZUhFv4pCRqcl-wB2Ve9
To: /content/haarcascade_frontalface_default.zip
  0% 0.00/140k [00:00<?, ?B/s] 100% 140k/140k [00:00<00:00, 116MB/s]
Archive: haarcascade_frontalface_default.zip
  inflating: haarcascade_frontalface_default.xml

# model detektora MMOD (sieć neuronowa z biblioteki dlib)
# https://drive.google.com/file/d/1gwFX_zny4A6DVkQ2gV2FlKY39qh1iDYj/
```

```
view?usp=sharing
!gdown 1gwFX_zny4A6DVkQ2gV2FlKY39qh1iDYj

Downloading...
From: https://drive.google.com/uc?id=1gwFX_zny4A6DVkQ2gV2FlKY39qh1iDYj
To: /content/mmod_human_face_detector.dat
    0% 0.00/730k [00:00<?, ?B/s] 100% 730k/730k [00:00<00:00, 138MB/s]

# modele detektora twarzy (sieci neuronowa) z biblioteki insightface
!gdown lavyNxM0XoYh8YfxhhZoCvvI0lE7qT8Gy
!unzip insightface.zip

Downloading...
From: https://drive.google.com/uc?id=lavyNxM0XoYh8YfxhhZoCvvI0lE7qT8Gy
To: /content/insightface.zip
100% 23.4M/23.4M [00:00<00:00, 31.1MB/s]
Archive: insightface.zip
    creating: insightface/models/
    creating: insightface/models/buffalo_l/
    inflating: insightface/models/buffalo_l/det_10g.onnx
    creating: insightface/models/buffalo_m/
    inflating: insightface/models/buffalo_m/det_2.5g.onnx
    creating: insightface/models/buffalo_s/
    inflating: insightface/models/buffalo_s/det_500m.onnx
    creating: insightface/models/buffalo_sc/
    inflating: insightface/models/buffalo_sc/det_500m.onnx

# obraz łatwy
#https://drive.google.com/file/d/1sYPRVOL-c1H0IW0muCFq18htKldIqQjL/
view?usp=sharing
!gdown 1sYPRVOL-c1H0IW0muCFq18htKldIqQjL

Downloading...
From: https://drive.google.com/uc?id=1sYPRVOL-c1H0IW0muCFq18htKldIqQjL
To: /content/0_Parade_marchingband_1_95.jpg
    0% 0.00/142k [00:00<?, ?B/s] 100% 142k/142k [00:00<00:00, 81.5MB/s]

# obraz trudny
# https://drive.google.com/file/d/1anR16ksibBMqQmZ5NUPyw-nWONLYDidv/
view?usp=sharing
! gdown 1anR16ksibBMqQmZ5NUPyw-nWONLYDidv

Downloading...
From: https://drive.google.com/uc?id=1anR16ksibBMqQmZ5NUPyw-nWONLYDidv
To: /content/2_Demonstration_Demonstration_Or_Protest_2_29.jpg
    0% 0.00/116k [00:00<?, ?B/s] 100% 116k/116k [00:00<00:00, 88.6MB/s]

# obraz średni
# https://drive.google.com/file/d/11hv84wBa1B0yQuCbkTnGasj0GRkJXKpv/
view?usp=sharing
!gdown 11hv84wBa1B0yQuCbkTnGasj0GRkJXKpv
```

Downloading...

From: <https://drive.google.com/uc?id=1hv84wBa1B0yQucbkTnGasj0GRkJXKpv>
To: /content/7_Cheering_Cheering_7_16.jpg
0% 0.00/154k [00:00<?, ?B/s] 100% 154k/154k [00:00<00:00, 103MB/s]

Obraz testowy

```
img = cv2.imread("2_Demonstration_Demonstration_Or_Protest_2_58.jpg")
# przygotowanie obrazów: monochromatycznego i RGB (cv2.imread() zwraca
# obraz w formacie BGR - inna kolejność składowych)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

cv2_imshow(img)
```



Kaskada Haara

Opracowano na podstawie: <https://www.pyimagesearch.com/2021/04/05/opencv-face-detection-with-haar-cascades/>

```
# utworzenie i inicjalizacja detektora
haar_detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
```

```

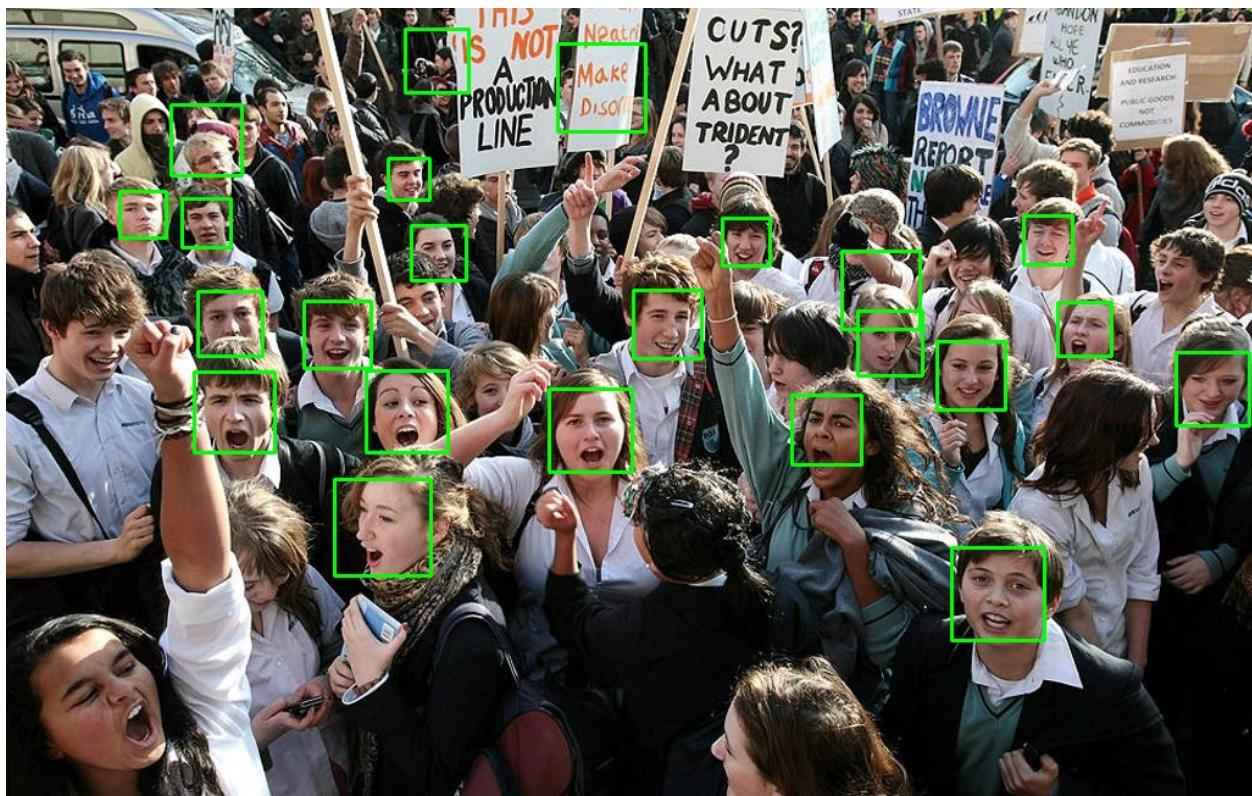
# wywołanie detektora dla określonego obrazu (img_gray)
# wynikiem jest lista prostokątów w formacie [x, y, width, height]
haar_results = haar_detector.detectMultiScale(img_gray,
scaleFactor=1.05,
minNeighbors=5, minSize=(30,
30),

flags=cv2.CASCADE_SCALE_IMAGE)
print(len(haar_results))

# narysowanie wyników na kopii obrazu i wyświetlenie
img_haar = img.copy()
for (x, y, w, h) in haar_results:
    cv2.rectangle(img_haar, (x, y), (x+w, y+h), (0, 255, 0), 2)
cv2_imshow(img_haar)

```

23



Histogram zorientowanych gradientów (HOG) z maszyną wektorów nośnych (SVM)

Opracowano na podstawie: <https://www.pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/>

```

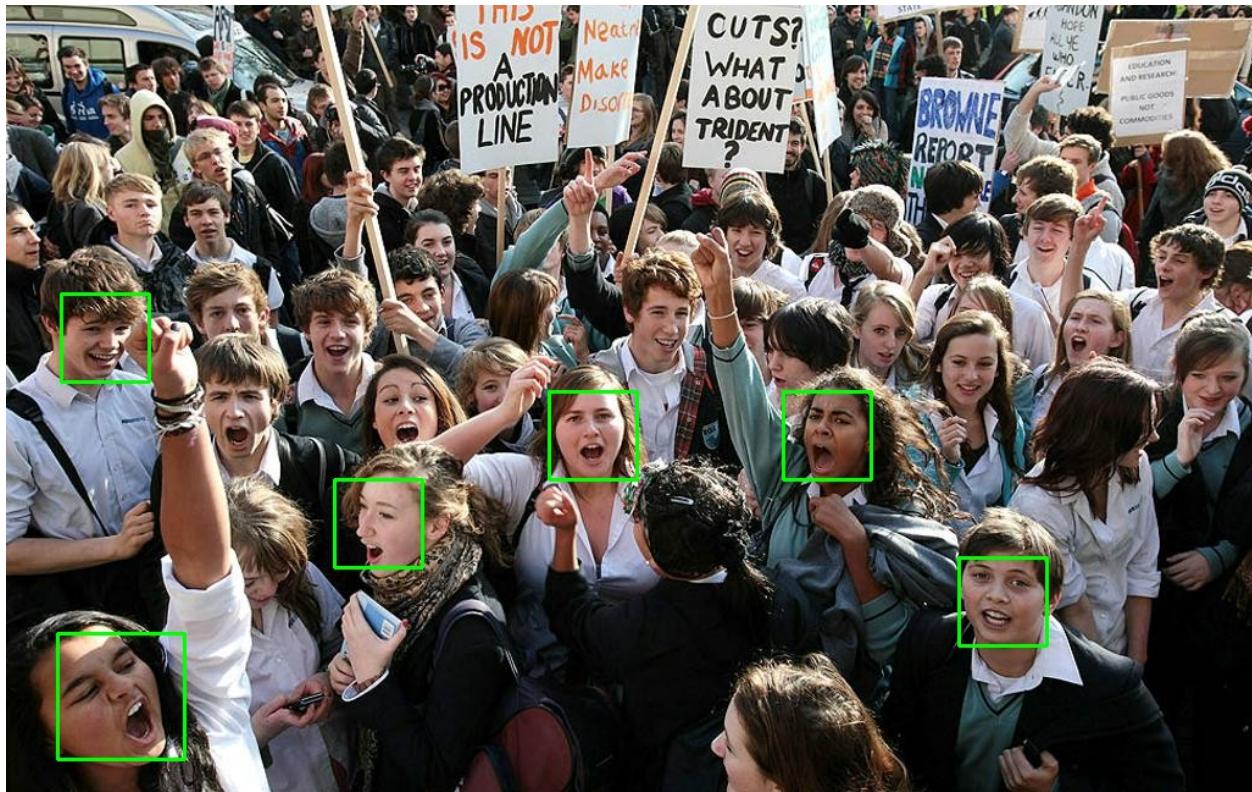
# utworzenie i inicjalizacja detektora
hog_svm_detector = dlib.get_frontal_face_detector()

# wywołanie detektora dla określonego obrazu (img_rgb)
# wynikiem jest lista obiektów rectangle, zawierających współrzędne
# lewego górnego i prawego dolnego narożnika prostokąta
for i in range(3):
    hog_svm_results = hog_svm_detector(img_rgb, i)
    print(len(hog_svm_results))

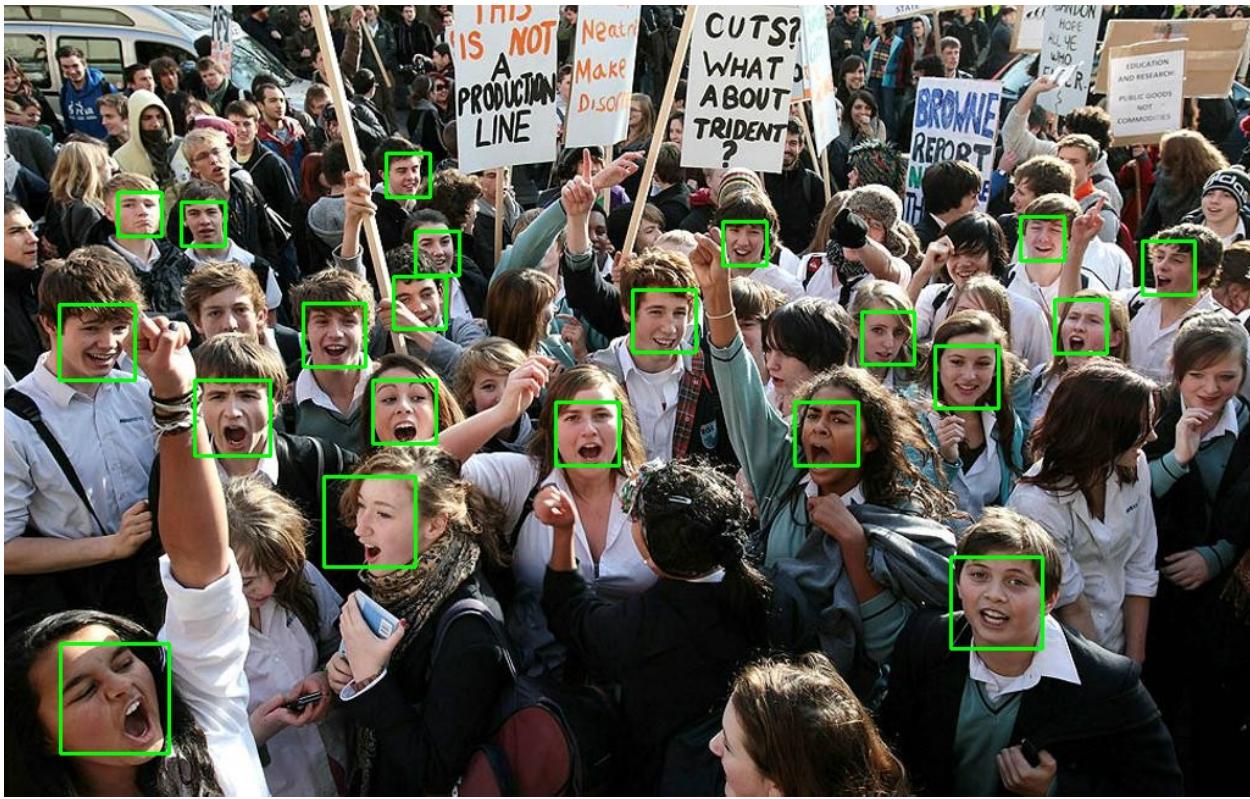
    # narysowanie wyników na kopii obrazu i wyświetlenie
    img_hog_svm = img.copy()
    for rect in hog_svm_results:
        cv2.rectangle(img_hog_svm, (rect.left(), rect.top()),
(rect.right(), rect.bottom()), (0, 255, 0), 2)
    cv2_imshow(img_hog_svm)

```

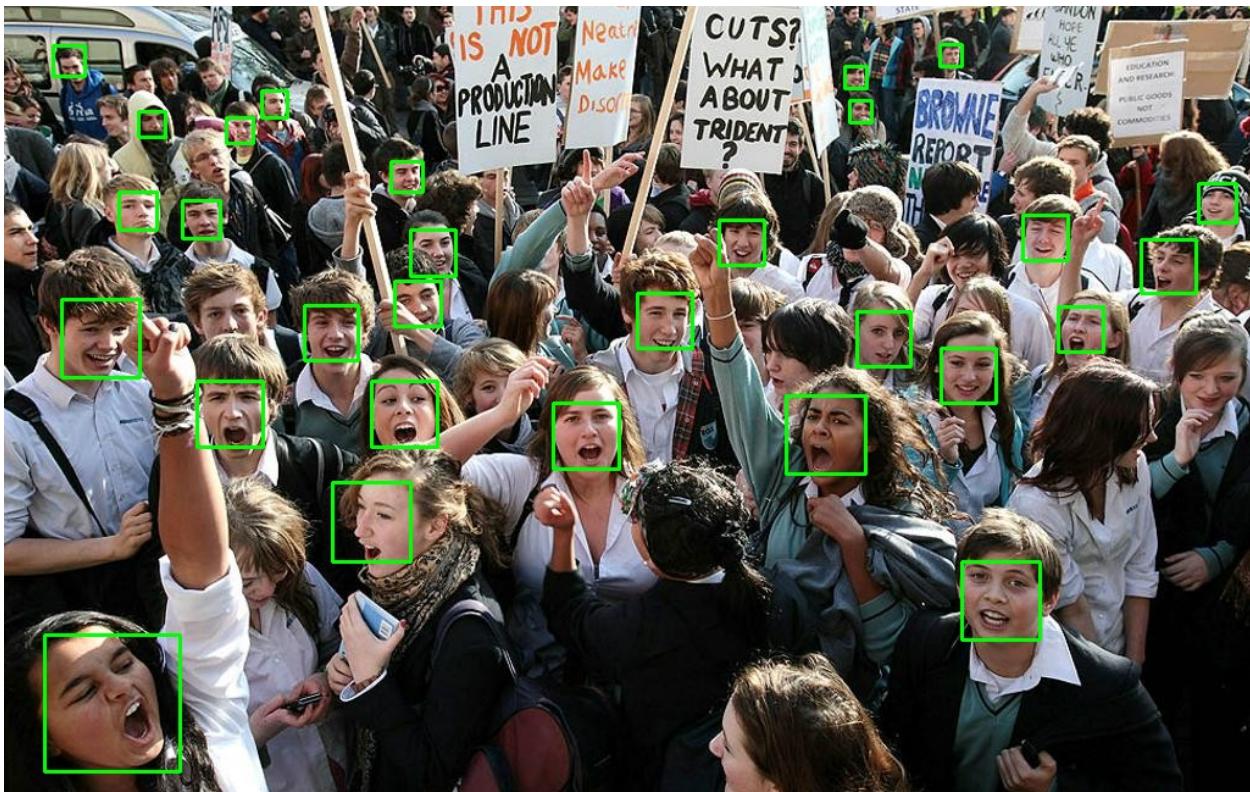
6



21



29



Porównanie wartości drugiego parametru detektora

```
values = [0, 1, 2]
results = []
for val in values:
    res = hog_svm_detector(img_rgb, val)
    results.append(len(res))

result_dict = dict(zip(values, results))
for key, value in result_dict.items():
    print(f"Value: {key}, Result: {value}")

Value: 0, Result: 6
Value: 1, Result: 21
Value: 2, Result: 29
```

Najlepsze wyniki uzyskaliśmy dla wartości 2

Splotowa sieć neuronowa (CNN) z biblioteki dlib

Opracowano na podstawie: <https://www.pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/>

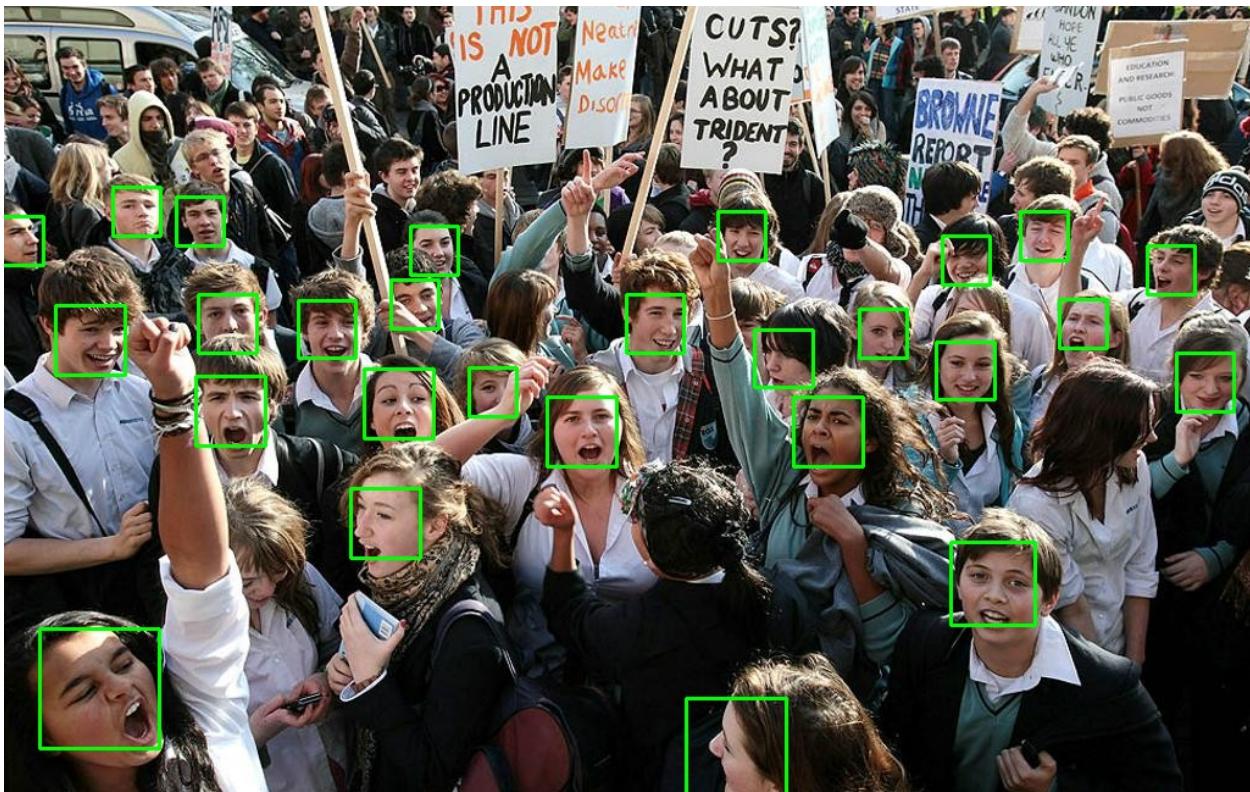
```
# utworzenie i inicjalizacja detektora
cnn1_detector =
dlib.cnn_face_detection_model_v1('mmod_human_face_detector.dat')

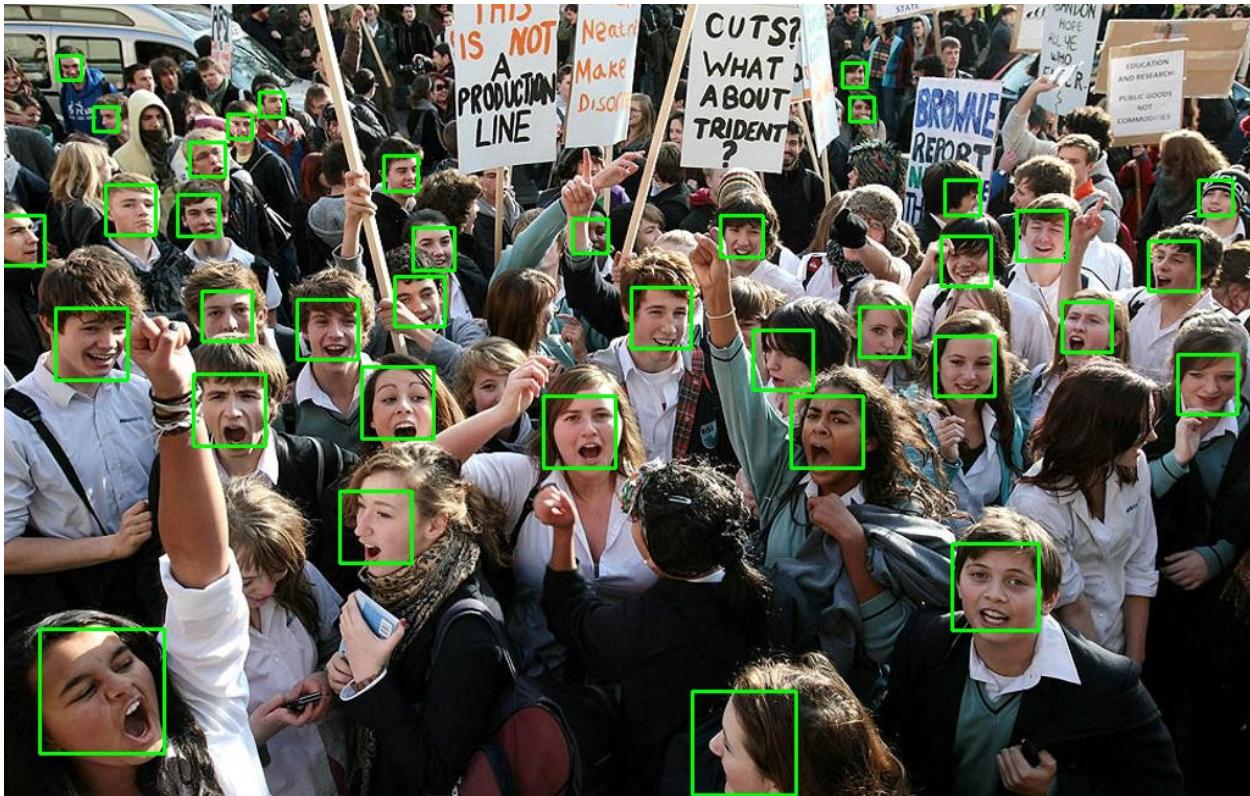
# wywołanie detektora dla określonego obrazu (img_rgb)
# wynikiem jest lista obiektów mmod_rectangle, zawierających m.in.
# pole rect ze współrzędnymi lewego górnego i prawego dolnego narożnika
# prostokąta
for i in range(3):
    cnn1_results = cnn1_detector(img_rgb, i)
    print(len(cnn1_results))

    # narysowanie wyników na kopii obrazu i wyświetlenie
    img_cnn1 = img.copy()
    for res in cnn1_results:
        rect = res.rect
        cv2.rectangle(img_cnn1, (rect.left(), rect.top()),
(rect.right(), rect.bottom()), (0, 255, 0), 2)
    cv2.imshow(img_cnn1)
```



27





```

values = [0, 1, 2]
results = []
for val in values:
    res = cnn1_detector(img_rgb, val)
    results.append(len(res))

result_dict = dict(zip(values, results))
for key, value in result_dict.items():
    print(f"Value: {key}, Result: {value}")

Value: 0, Result: 4
Value: 1, Result: 27
Value: 2, Result: 37

```

Podobnie dla CNN najlepsze wartości uzyskaliśmy dla wartości równej 2

InsightFace - inna sieć neuronowa

Opracowano na podstawie: <https://github.com/deepinsight/insightface/tree/master/python-package>

```
# utworzenie i inicializacja detektora
model_name = 'buffalo_s' # model: small (_s), medium (_m) lub large
```

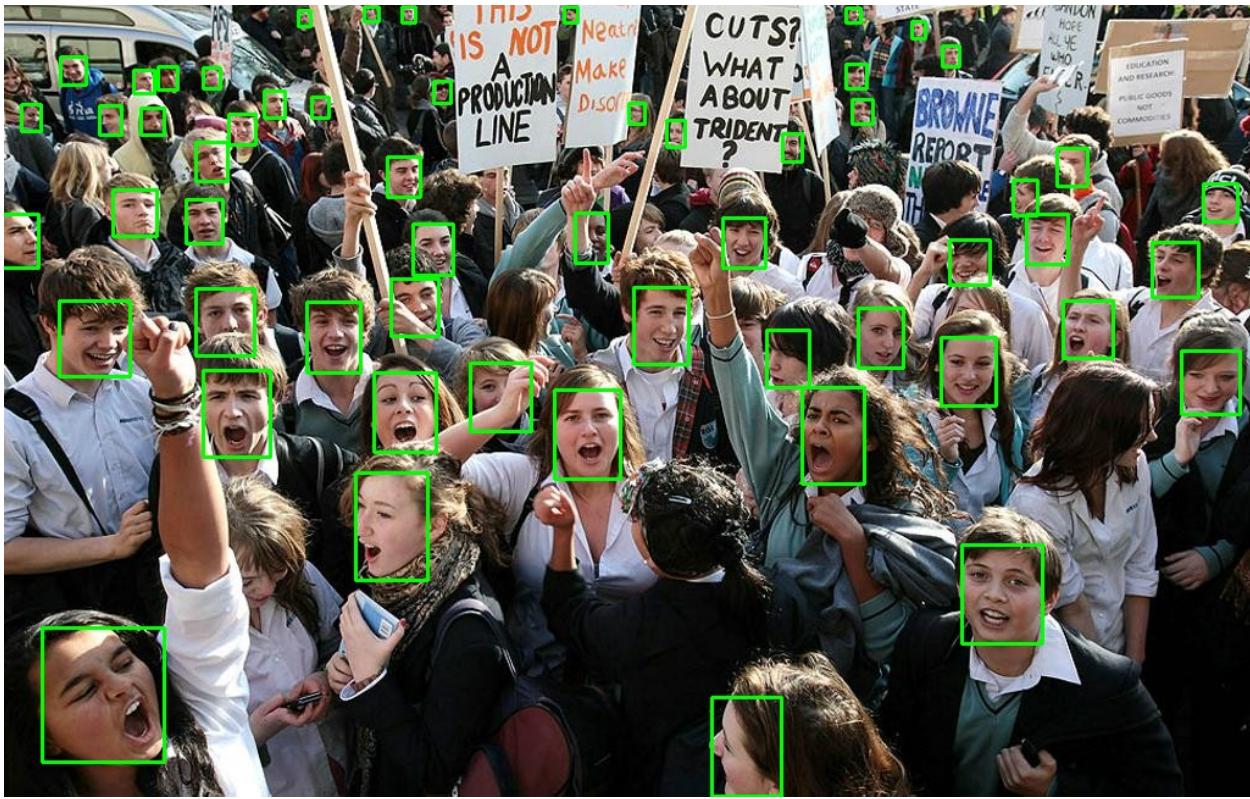
```
(_l)
insf_detector = insightface.app.FaceAnalysis(name=model_name,
root='insightface',

allowed_modules=['detection'], providers=['CPUExecutionProvider'])
insf_detector.prepare(ctx_id=0, det_size=(1024, 1024))

Applied providers: ['CPUExecutionProvider'], with options:
{'CPUExecutionProvider': {}}
find model: insightface/models/buffalo_s/det_500m.onnx detection [1,
3, '?', '?'] 127.5 128.0
set det-size: (1024, 1024)

# wywołanie detektora dla określonego obrazu (img_rgb)
# wynikiem jest lista obiektów, zawierających m.in. pole bbox ze
# współrzędnymi lewego górnego i prawego dolnego narożnika prostokąta
insf_results = insf_detector.get(img)
print(len(insf_results))

# narysowanie wyników na kopii obrazu i wyświetlenie
img_insf = img.copy()
for res in insf_results:
    # współrzędne prostokątów sa zapisane jako liczby rzeczywiste -
    # konwersja do liczb całkowitych
    rect = res.bbox.round().astype(int)
    cv2.rectangle(img_insf, (rect[0], rect[1]), (rect[2], rect[3]), (0,
255, 0), 2)
cv2_imshow(img_insf)
```



WNIOSKI DO OBRAZU TESTOWEGO

Założenia

- Jeżeli została wykryty bok twarzy, albo częściowo przysłonięta twarz, ale my jako ludzie patrząc na zdjęcie jesteśmy w stanie powiedzieć, że jest to twarz, to zaliczamy to jako dobrze wykrytą twarz przez detektor
- Jeżeli jest wykryta twarz i dość spory obszar zajmuje tło, ale nie ma tam więcej niż jednej twarzy, to też zaliczam jako dobrą detekcję
- Zaproponowana przeze mnie miara to $\frac{\text{liczba wykrytych kobiet}}{\text{liczby kobiet w zdjęciu}}$ i oznaczę ją jako LK
- Dla obrazu testowego liczba kobiet, którą wyliczyłem wynosi 20

Wyniki

Metoda	Ground Truths	Liczba wykrytych twarzy	FP	FN	LK	
Kaskada Haara	101	20	3	81	8 / 20	HOG + SVM
CNN	101	37	0	64	14 / 20	Insight Face
	101	56	0	45	19 / 20	

- Najlepszą metodą do wykrywania twarzy okazała się metoda InsightFace. Wykryła najwięcej twarzy
 - Tylko Kaskada Haara uznała obiekty niebędące twarzami, za twarze. Dała nam 3 false positives
 - HOG + SVM i CNN nie dawały false positives, ale też nie wykryły wszystkich twarzy
 - Różne detektory wykrywają na tych samych obrazach różne twarze (ale większość się pokrywa)
 - Dla mojej miary widać, że najlepiej kobiety wykrywa InsightFace i wykrył 19 na 20 kobiet, które wyliczyłem

FUNKCJE POMOCNICZE DO WYSWIETLANIA OBRAZÓW Z WYKRYTYMI TWARZAMI

```
def use_haar(image):
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    haar_results = haar_detector.detectMultiScale(image_gray,
scaleFactor=1.05,
minNeighbors=5, minSize=(30,
30),
flags=cv2.CASCADE_SCALE_IMAGE)

# narysowanie wyników na kopii obrazu i wyświetlenie
img_haar = image.copy()
for (x, y, w, h) in haar_results:
    cv2.rectangle(img_haar, (x, y), (x+w, y+h), (0, 255, 0), 2)
cv2_imshow(img_haar)

def use_hog(image, quality=2):
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    hog_svm_results = hog_svm_detector(image_rgb, quality)

# narysowanie wyników na kopii obrazu i wyświetlenie
img_hog_svm = image.copy()
for rect in hog_svm_results:
    cv2.rectangle(img_hog_svm, (rect.left(), rect.top()),
(rect.right(), rect.bottom()), (0, 255, 0), 2)
cv2_imshow(img_hog_svm)

def use_cnn(image, quality=2):
```

```

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
cnn1_results = cnn1_detector(image_rgb, quality)

# narysowanie wyników na kopii obrazu i wyświetlenie
img_cnn1 = image.copy()
for res in cnn1_results:
    rect = res.rect
    cv2.rectangle(img_cnn1, (rect.left(), rect.top()), (rect.right(),
rect.bottom()), (0, 255, 0), 2)
cv2_imshow(img_cnn1)

def use_insight_face(image):
    insf_results = insf_detector.get(image)

# narysowanie wyników na kopii obrazu i wyświetlenie
img_insf = image.copy()
for res in insf_results:
    # współrzędne prostokątów sa zapisane jako liczby rzeczywiste -
konwersja do liczb całkowitych
    rect = res.bbox.round().astype(int)
    cv2.rectangle(img_insf, (rect[0], rect[1]), (rect[2], rect[3]),
(0, 255, 0), 2)
cv2_imshow(img_insf)

```

OBRAZ ŁATWY

```

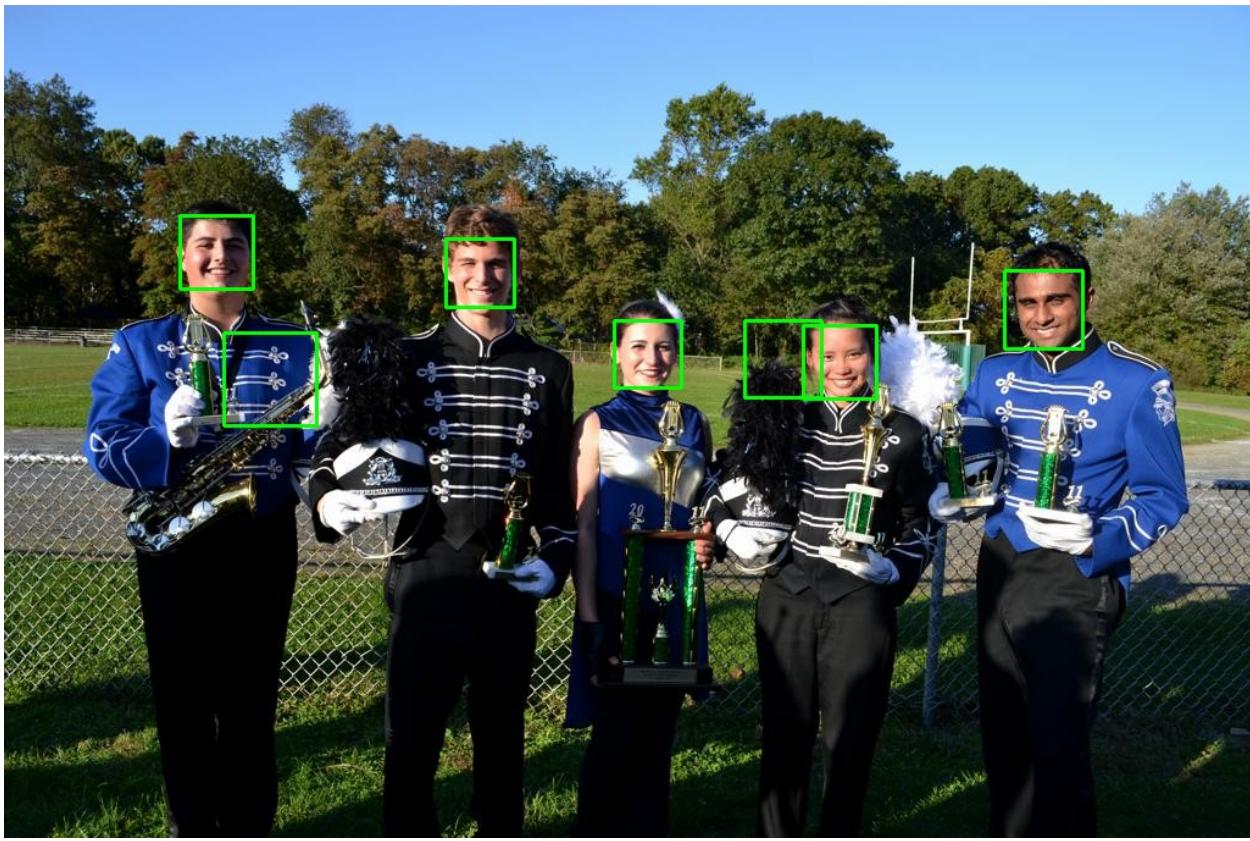
easy_img = cv2.imread("/content/0_Parade_marchingband_1_95.jpg")
cv2_imshow(easy_img)

```



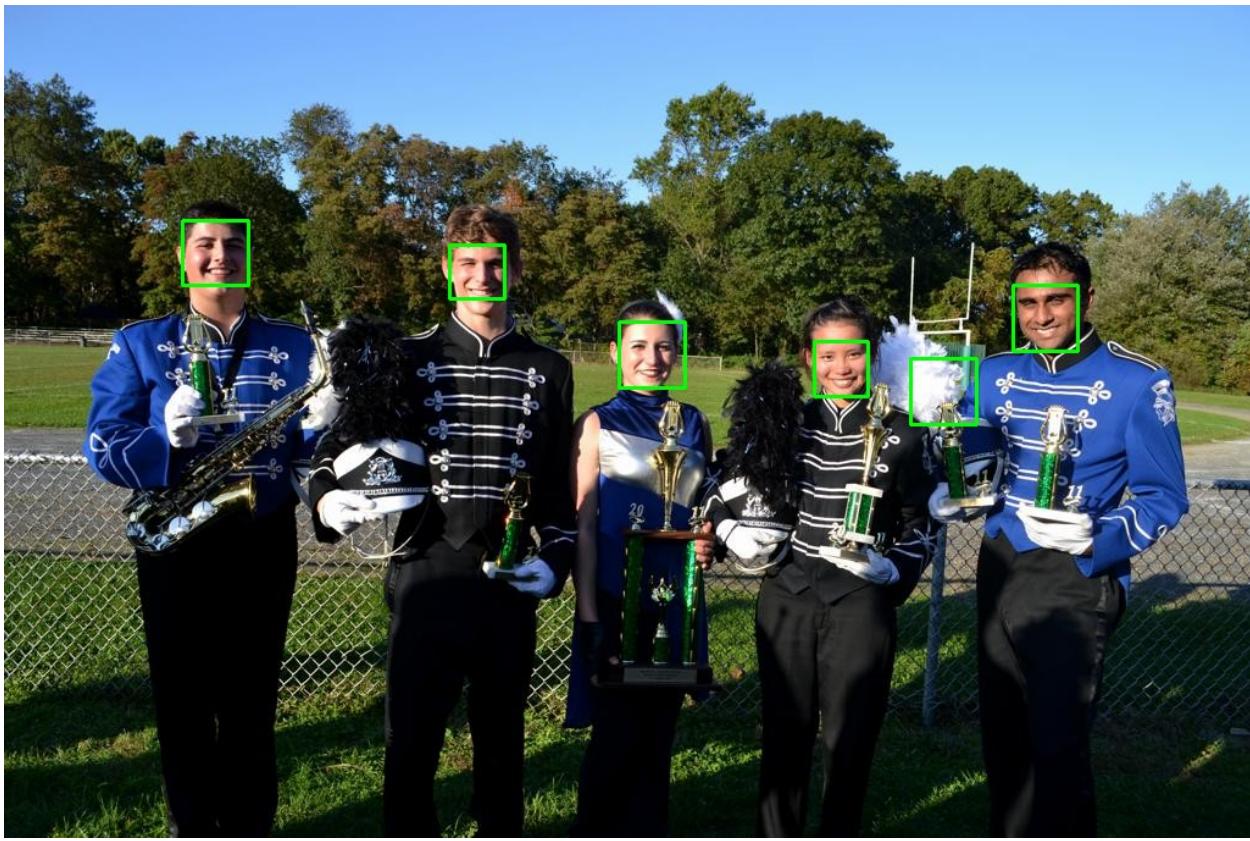
KASKADA HAARA

use_haar(easy_img)



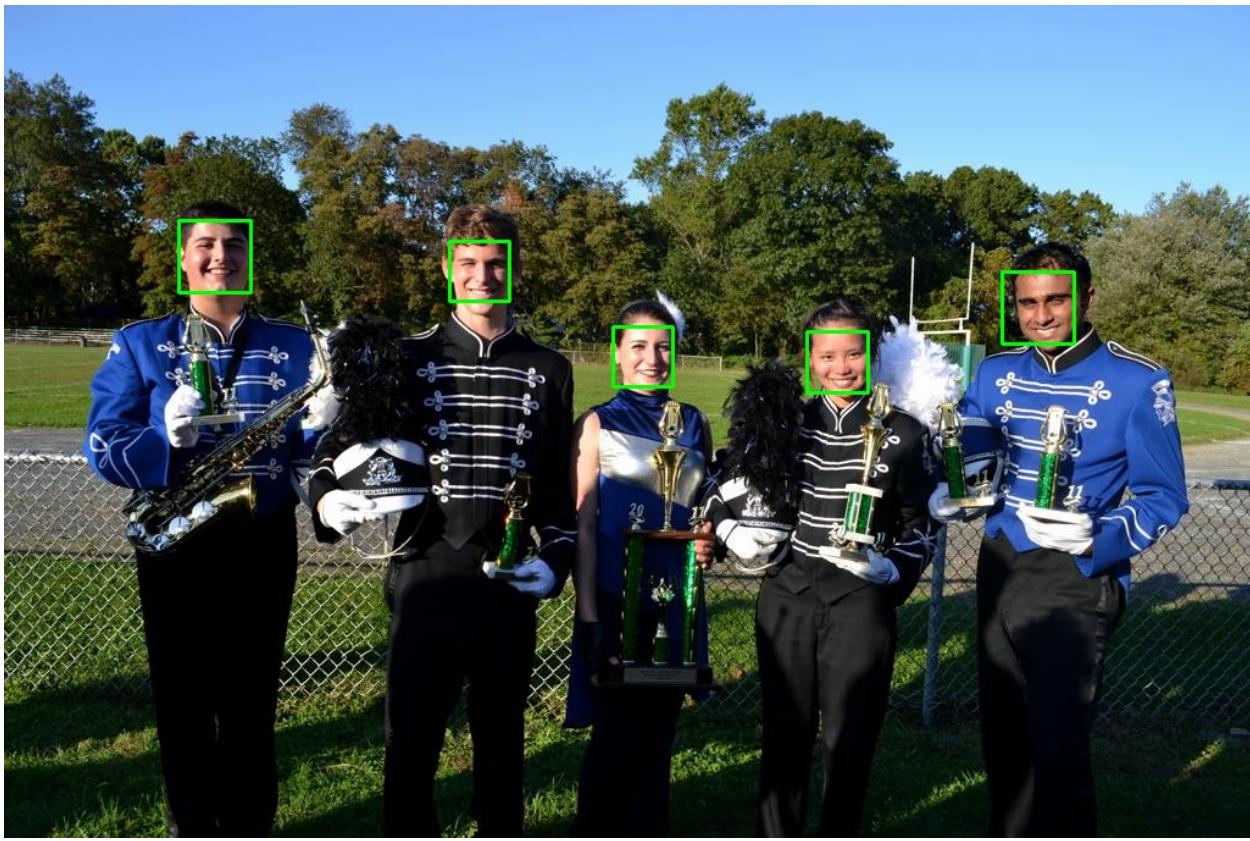
HOG

```
use_hog(easy_img)
```



CNN

```
use_cnn(easy_img)
```



Insight Face

```
use_insight_face(easy_img)
```



WNIOSKI

Metoda	Ground Truths	Liczba wykrytych twarzy	FP	FN	LK	-----
-----	-----	-----	-----	-----	Kaskada Haara	5 5 2 0 2/2
1 0 2/2	CNN	5 5 0 0 2/2	Insight Face	5 5 0 0 2/2	HOG + SVM	5 5

- Detektory wykorzystujące sieci neuronowe najlepiej rozpoznały twarze. Miały 100% skuteczność
- Zarówno Kaskada Haara jak i HOG + SVM wykryły też wszystkie twarze, ale dodatkowo dały nam w wynikach false positive -> uznali za twarze obiekty, co nie są twarzami
- Wszystkie detektory miał 100% skuteczność w wykrywaniu kobiet na zdjęciu. Może to wynikać z tego, że zdjęcie było bardzo proste i twarze dobrze widoczne

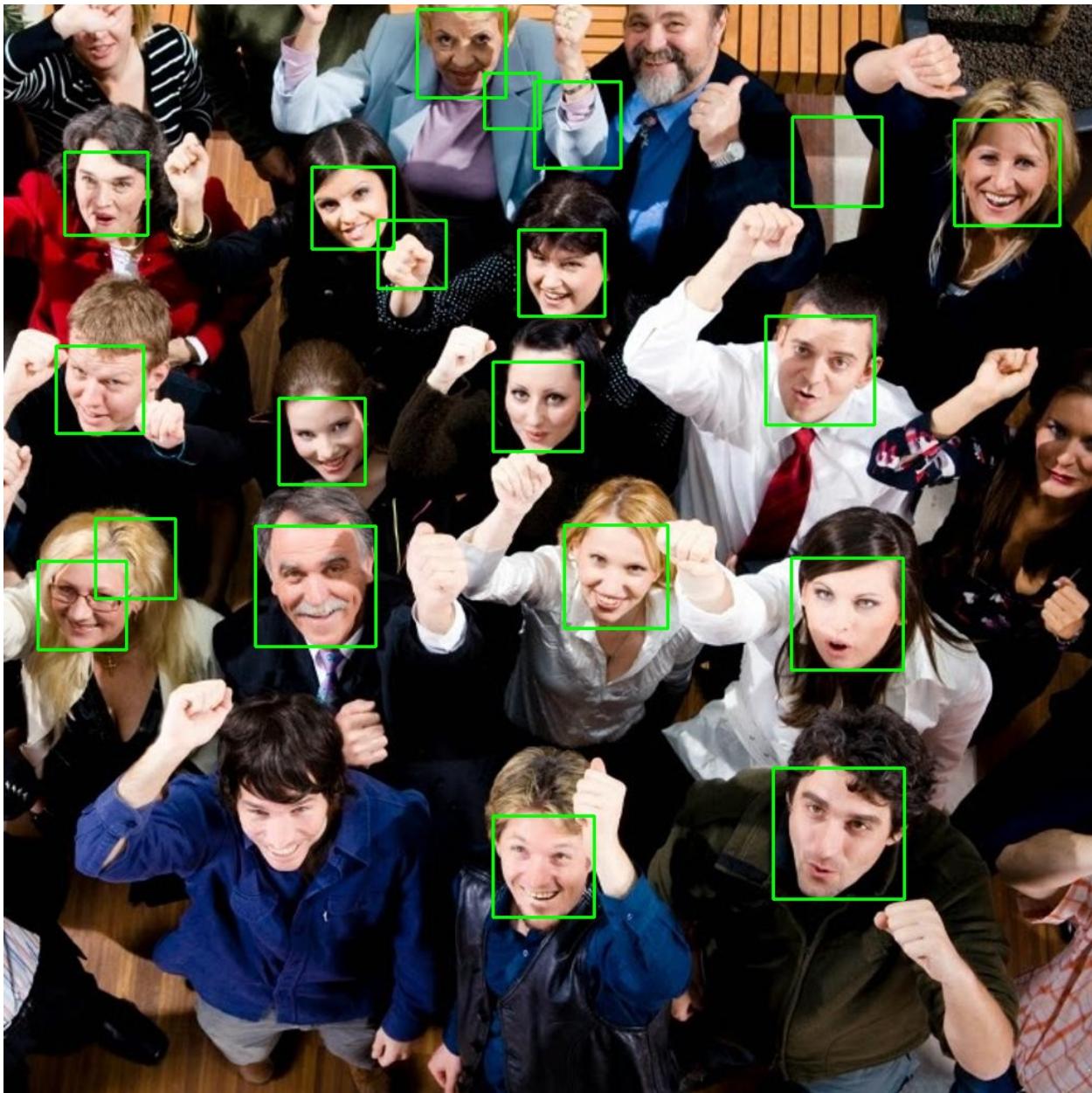
OBRAZ ŚREDNI

```
medium_img = cv2.imread("/content/7_Cheering_Cheering_7_16.jpg")
cv2.imshow(medium_img)
```



HAAR

```
use_haar(medium_img)
```



HOG + SVM

```
use_hog(medium_img)
```



CNN

```
use_cnn(medium_img)
```



InsightFace

```
use_insight_face(medium_img)
```



WNIOSKI

Metoda	Ground Truths	Liczba wykrytych twarzy	FP	FN	LK
Kaskada Haara	19	15	4	4	10 / 12
HOG + SVM	19	15	0	4	10 / 12
CNN	19	18	1	0	11 / 12
Insight Face	19	19	0	0	12 / 12

- Podobnie jak dla obrazka łatwego najlepszą skuteczność odniostły CNN jak i InsightFace
- HOG + SVM i CNN nie wykryły wszystkich twarzy w ilościach odpowiednio 4 i 1
- Tylko Kaskada Haara uznała obiekty niebędące twarzami za twarze
- Jest to zdjęcie w miarę proste i również wszystkie metody zadziałały dość skutecznie

OBRAZ TRUDNY

```
hard_img =  
cv2.imread( "/content/2_Demonstration_Demonstration_Or_Protest_2_29.jpg  
")  
  
cv2_imshow(hard_img)
```



HAAR

```
use_haar(hard_img)
```



- W tym przykładzie mamy jeden box, który wykrył dwie twarze na raz i w tym przypadku nie uznaje go jako dobre wykrycie pojedyńczej twarzy

HOG + SVM

```
use_hog(hard_img)
```



CNN

```
use_cnn(hard_img)
```



InsightFace

```
use_insight_face(hard_img)
```



WNIOSKI

Metoda	Ground Truths	Liczba wykrytych twarzy	FP	FN	LK	
Kaskada Haara	33	7	5	26	2/5	HOG + SVM 33
CNN	33	13	0	20	4/5	8 0 25 2/5 Insight Face 33 20 0 13 5/5

- Byłem w stanie stwierdzić, że tylko 5 twarzy widocznych na zdjęciu należy do kobiet. Zdjęcie jest trudne, twarze są małe i czasami ciężko określić płeć
- Zdjęcie jest dosyć trudne, mamy dużo małych twarzy oraz wiele jest przysłoniętych dlatego wszystkie detektory poradziły sobie na nim już gorzej
- Najlepszym znowu okazał się InsightFace, a najgorszym Kaskada Haara

WYNIKI Z WSZYSTKICH OBRAZÓW

Metoda	Ground Truths	Liczba wykrytych twarzy	FP	FN	LK	
Kaskada Haara	158	47	14	111	22/39	HOG + SVM 158 57 1 101 25/39
CNN	158	73	1	84	31/39	38/39 Insight Face 158 100 0 58

WNIOSKI

- Najlepszym detektorem twarzy okazuje się InsightFace. Wykrywa on zarówno najwięcej twarzy ze wszystkich, najlepiej rozpoznaje kobiety oraz nie wykrywa żadnych FP
- Najgorszym detektorem jest Kaskada Haara (dlatego, jest to metoda już przestarzała). Wykrywa najmniej twarzy ze zdjęć oraz najczęściej błędnie ocenia czy dany fragment zdjęcia to twarz. Dał nam najwięcej FP, bo aż 14
- Dla prostych zdjęć, na których wszystkie twarze są dobrze widoczne i nie ma ich dużo. Wszystkie detektory dają nam dobrą skuteczność i dobrze wykrywają twarze i robią mało błędów
- Wraz z wzrostem "trudności zdjęcia" Kaskada Haara jak i HOG + SVM tracą na jakości i lepiej w takowych sytuacjach zastosować metody bazujące na sieciach neuronowych

DETEKCJA CZASU

```
# inicjalizacja detektorów

# HARR gray
haar_detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# HOG rgb
hog_svm_detector = dlib.get_frontal_face_detector()

# CNN rgb
cnnl_detector =
dlib.cnn_face_detection_model_v1('mmod_human_face_detector.dat')

#InsightFace

model_name = 'buffalo_s' # model: small (_s), medium (_m) lub large (_l)
insf_detector = insightface.app.FaceAnalysis(name=model_name,
root='insightface',

allowed_modules=['detection'], providers=['CPUExecutionProvider'])
insf_detector.prepare(ctx_id=0, det_size=(1024, 1024))

# funkcje liczące wynik detektorów

def detect_haar(image_gray):
    haar_results = haar_detector.detectMultiScale(image_gray,
```

```

scaleFactor=1.05,
                           minNeighbors=5, minSize=(30,
30),

flags=cv2.CASCADE_SCALE_IMAGE)

def detect_hog(image_rgb):
    hog_svm_results = hog_svm_detector(image_rgb, 2)

def detect_cnn(image_rgb):
    cnn1_results = cnn1_detector(image_rgb, 2)

def detect_if(image):
    insf_results = insf_detector.get(image)

Applied providers: ['CPUExecutionProvider'], with options:
{'CPUExecutionProvider': {}}
find model: insightface/models/buffalo_s/det_500m.onnx detection [1,
3, '?', '?'] 127.5 128.0
set det-size: (1024, 1024)

images = [easy_img, medium_img, hard_img]
images_in_gray = [cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in
images]
images_in_rgb = [cv2.cvtColor(image, cv2.COLOR_BGR2RGB) for image in
images]

def calculate_times_dict(iterations):

    dict_with_times = {
        'Kaskada Haara': 0,
        'HOG': 0,
        'CNN' : 0,
        'IF' : 0
    }

    for i in range(0, iterations-1):
        # HARR
        st_haar = time.time()
        detect_haar(images_in_gray[i%3])
        end_harr = time.time()

        dict_with_times['Kaskada Haara'] += end_harr - st_haar

        # CNN
        st_cnn = time.time()
        detect_cnn(images_in_rgb[i%3])
        end_cnn = time.time()

        dict_with_times['CNN'] += end_cnn - st_cnn

```

```

# HOG
st_hog = time.time()
detect_hog(images_in_rgb[i%3])
end_hog = time.time()

dict_with_times['HOG'] += end_hog - st_hog

# IF
st_if = time.time()
detect_if(images[i%3])
end_if = time.time()

dict_with_times['IF'] += end_if - st_if

return dict_with_times

def calculate_avg_time(dict_with_times, iterations):
    for key in dict_with_times.keys():
        time = dict_with_times[key] / iterations
        print(f"Metoda: {key}, Czas: {time}")

iterations = 20

for i in range(3):
    print("Iteracja : ", i)
    dictionary_with_time = calculate_times_dict(iterations)
    calculate_avg_time(dictionary_with_time, iterations)

Iteracja : 0
Metoda: Kaskada Haara, Czas: 0.9549247860908509
Metoda: HOG, Czas: 1.9658023595809937
Metoda: CNN, Czas: 0.6250534176826477
Metoda: IF, Czas: 0.1017235279083252
Iteracja : 1
Metoda: Kaskada Haara, Czas: 0.9363463759422302
Metoda: HOG, Czas: 1.9648291349411011
Metoda: CNN, Czas: 0.6174036860466003
Metoda: IF, Czas: 0.10554277896881104
Iteracja : 2
Metoda: Kaskada Haara, Czas: 0.957996416091919
Metoda: HOG, Czas: 1.996031367778778
Metoda: CNN, Czas: 0.6176560401916504
Metoda: IF, Czas: 0.10285462141036987

```

Wnioski

- Najszybszą metodą okazała się metoda IF, która rezultat potrafi osiągnąć w 0.1 sekundy
- Najwolniejsza metoda to metoda korzystająca z HOG + SVM i jej wykonanie trwa w okolicach 2 sekund
- Zarówno CNN i Kaskada Haara wykonują się poniżej sekundy
- Wszystkie wyliczenia są raczej stabilne, wnioskuję to po podobieństwie w czasach dla 3 iteracji
- Wykorzystano GPU w środowisku Google Collab