

**AGH**

**AKADEMIA GÓRNICZO-HUTNICZA  
IM. STANISŁAWA STASZICA  
W KRAKOWIE**

Wydział Fizyki i Informatyki Stosowanej

Inż. Bartłomiej Mucha  
Inż. Dorian Kossowski  
Inż. Piotr Kolecki  
Inż. Piotr Witkoś

Analiza i przetwarzanie obrazów

Projekt

# **Wykrywanie masek w obrazie wideo**

## *Spis Treści*

<b>Założenia i cel projektu</b>	<b>3</b>
<b>Cel</b>	<b>3</b>
<b>Założenia</b>	<b>3</b>
<b>Wstęp teoretyczny</b>	<b>4</b>
Sieć neuronowa jako czarna skrzynka	4
Sztuczne neurony	4
Nauka sieci	5
Przetrenowanie	6
<b>Opis użytych technologii</b>	<b>6</b>
<b>Opis implementacji</b>	<b>7</b>
Trening modelu wykrywającego maskę	7
Zdefiniowanie argumentów wejściowych programu	8
Wczytanie danych (zdjęć)	8
Stworzenie oraz konfiguracja sztucznej sieci neuronowej	10
Trening modelu sieci neuronowej	12
Testowanie sieci neuronowej	12
Zapis modelu do pliku	13
Implementacja modelu wykrywającego maskę w obrazie na żywo z kamarki	13
Zdefiniowanie argumentów wejściowych programu oraz ich interpretacja	14
Funkcja odpowiadająca za wykrywanie maski na twarzy w obrazie z kamarki internetowej	15
Główna pętla realizująca przebieg programu	17
Definicja interfejsu	19
<b>Instrukcja użytkownika</b>	<b>19</b>
Przygotowanie środowiska programistycznego	19
Generowanie modelu zdolnego do wykrywania maseczki na twarzy	21
Obsługa skryptu analizującego czy osoby mają maseczkę na obrazie na żywo z kamarki	23
<b>Podsumowanie</b>	<b>26</b>
<b>Źródła</b>	<b>27</b>

# 1. Założenia i cel projektu

## 1.1. Cel

Celem projektu jest realizacja systemu rozpoznawania czy na twarzy osoby uchwyconej na obrazie wideo znajduje się maska. Program ma w czasie rzeczywistym dokonywać rozpoznania na obrazie z kamery internetowej. Kod programu został napisany w języku *Python* z wykorzystaniem pakietów, których dokładny opis znajduje się w punkcie **3 Opis użytych technologii**.

## 1.2. Założenia

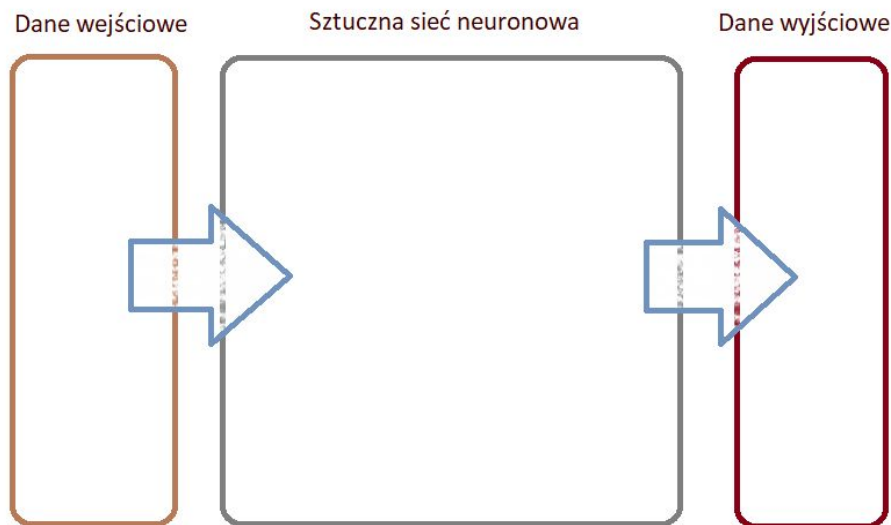
Działanie programu opiera się na dwóch instancjach sztucznej sieci neuronowej analizujących przechwycony obraz z kamery internetowej. Jedna z sieci będzie odpowiedzialna za rozpoznawanie i wyodrębnianie twarzy. Druga będzie miała na celu zweryfikowanie obecności maski na odnalezionej przez pierwszą sieć twarzy. Tematem projektu jest rozpoznawanie czy osoba na obrazie posiada maskę, a nie samo rozpoznawanie twarzy, dlatego jako instancja rozpoznająca twarze została zaadaptowana gotowa, już wytrenowana sieć neuronowa [https://github.com/thegopieffect/computer\\_vision](https://github.com/thegopieffect/computer_vision).

W projekcie zostanie zatem utworzona i wytrenowana sieć neuronowa odpowiedzialna za weryfikację obecności maski w ramach narzędzia pomocniczego. Główny program będzie realizował przechwytywanie obrazu z kamery i przekazywał do sieci neuronowych. Na początku analizy obrazu pierwsza sieć neuronowa podejmie próbę wykrycia twarzy człowieka. W przypadku pozytywnego rozpoznania twarzy obraz zostanie przycięty do rozmiaru samej twarzy i przekazany do drugiej sieci w celu wykrycia maski.

## 2. Wstęp teoretyczny

### 2.1. Sieć neuronowa jako czarna skrzynka

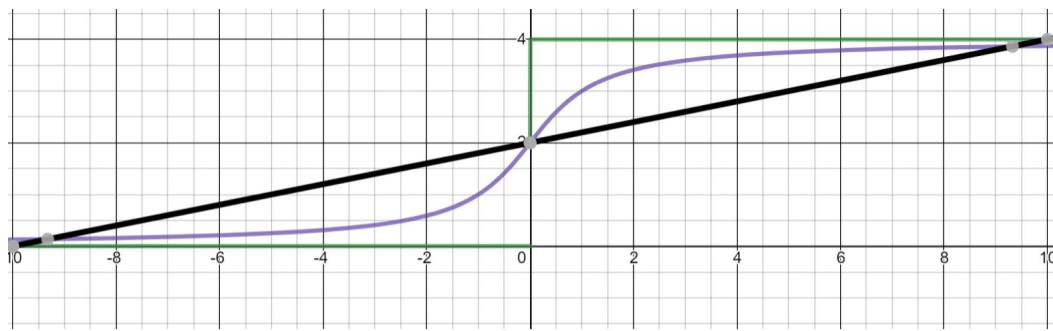
Z perspektywy programisty sieć neuronowa to czarna skrzynka. W związku z tym znajomość znaczenia parametrów konfiguracyjnych i konstrukcji zestawów treningowych jest wystarczająca, żeby taką sieć stworzyć i następnie wykorzystać.



Obraz 1 Czarnoskrzynkowy model sieci neuronowej.

### 2.2. Sztuczne neurony

Neuron jest pojedynczą komórką algorytmu wchodzącą w skład sztucznej sieci neuronowej. Jego działanie polega na utworzeniu i przekazaniu argumentu do wewnętrznej funkcji aktywacji. Argument to suma iloczynów wag i wartości poszczególnych danych wejściowych. Funkcja aktywacji musi być funkcją o ciągłej pochodnej i przebiegać tak by rozpoczynając się od swojej wartości minimalnej osiągała na koniec wartość maksymalną.



Obraz 2 Przykładowe funkcje aktywacji w neuronie.

Pojedyncza warstwa neuronów nie jest zbyt skuteczna. Dla zwiększania skuteczności tworzy się sieci wielowarstwowe, składające się z warstwy wejściowej, wyjściowej oraz ukrytej. Warstwa wejściowa składa się z tych neuronów, których wejście jest wystawione do wprowadzenia danych i tak samo warstwa wyjściowa wystawia wyjście znajdujących się w niej neuronów. Warstwa ukryta to ta skrywająca wszystkie pozostałe warstwy neuronów.

## 2.3. Nauka sieci

Sieć neuronowa, która rozpoznaje maski została wytrenowana metodą wstecznej propagacji. Polega ona na dostarczeniu zestawu treningowego składającego się z danych wejściowych i ustalonych danych weryfikujących. Najogólniej ujmując, trening polega na tym, że sieć próbuje tak przetworzyć dane wejściowe, żeby wynik operacji zgadzał się z danymi weryfikacyjnymi. Najpierw dane wejściowe są dzielone na porcję. Dana porcja zostaje przetworzona, w wyniku czego powstają dane wyjściowe. W wyniku porównania danych wyjściowych i weryfikacyjnych zostaje obliczony błąd. Błąd ten jest propagowany przez sieć od jej wyjścia do wejścia, w trakcie czego dochodzi do korekcji wag w poszczególnych neuronach. Proces jest powtarzany, aż błąd zostanie zminimalizowany do wartości akceptowalnie małej. Po osiągnięciu odpowiedniego błędu, proces jest powtarzany z kolejną porcją danych wejściowych. Okres przetwarzania w ten sposób całej puli danych wejściowych jest nazywany epoką. Na koniec każdej epoki obliczany jest błąd. Zatem proces nauczania sieci można zakończyć po wykonaniu określonej liczby epok lub do osiągnięcia ustalonego błędu.

## 2.4. Przetrenowanie

W kwestii doboru ilości epok lub błędu docelowego należy uważać by nie przetrenować sieci. Przetrenowanie objawia się niemalże perfekcyjnymi wynikami dla zestawu uczącego, przy jednoczesnych słabych wynikach dla nowych danych spoza tego zestawu. Innymi słowy przetrenowana sieć przestaje uogólniać otrzymane informacje, a zaczyna odbierać nieznaczne różnice jako znaczące przy jednoczesnym odbieraniu nie znaczących podobieństw jako znaczące. Zjawisko występuje w sytuacji, w której dobrany został zbyt mały błąd docelowy lub zbyt duża liczba epok względem ilości przykładów w zestawie treningowym.

## 3. Opis użytych technologii

Program został zrealizowany w środowisku języka *Python*. W skład najważniejszych użytych w projekcie pakietów języka *Python* wchodzi:

- 1) **Opencv-python (cv2)** - *API Python'a* do biblioteki *OpenCV*. Umożliwia ona szeroko rozumiane przetwarzanie grafiki, szybkie operacje na tablicach i wektorach oraz udostępnia funkcjonalność nauczania maszynowego. Posłuży do przetwarzania obrazu przechwyconego z kamery internetowej oraz korzystania ze sztucznej sieci neuronowej.
- 2) **Keras (framework tensorflow.keras)** - *API* umożliwiająca trening i korzystanie ze sztucznych sieci neuronowych. Ten framework został wykorzystany do utworzenia sieci neuronowej rozpoznającej maski na przechwyconym obrazie twarzy.
- 3) Zbiór narzędzi graficznych **imutils** - Jest to pakiet klas i metod opakowujących funkcjonalności biblioteki *opencv*. Udostępnia interfejs do przechwytywania obrazu z kamery, a także operacje na obrazach.
- 4) **Scikit-learn (sklearn)** - Pakiet ten posłużył do utworzenia zbioru treningowego i testowego dla sztucznej sieci neuronowej.

Jako sieć neuronową użyto **Mobile Net V2** z pakietu *Keras*- lekki i prosty w użyciu model sieci neuronowych przystosowany do pracy na urządzeniach o niskiej i średniej mocy obliczeniowej. Dobrze sprawdza się ona również do przeprowadzania detekcji w czasie rzeczywistym, co było kluczowe dla wykorzystania obrazu z kamery internetowej.

## 4. Opis implementacji

Wykrywanie masek na twarzach zostało wykonane w formie dwóch skryptów w technologii Python, które realizują poszczególne etapy implementacji.

### 1) Trening modelu wykrywającego maskę

Pierwszym etapem jest trening sieci neuronowej w celu wygenerowania modelu zdolnego do rozróżniania czy osoba w kamerze ma założoną maseczkę czy nie.

### 2) Implementacja modelu wykrywającego maskę w obrazie na żywo z kamery

Drugi etap to program uruchamiający kamerę, a następnie analizujący na podstawie dostarczonych modeli czy wykryte twarze na kamerze posiadają założoną maseczkę.

### 4.1. Trening modelu wykrywającego maskę

Trening modelu wykrywającego maskę można podzielić na poszczególne etapy:

- 1) Zdefiniowanie argumentów wejściowych programu
- 2) Wczytanie danych (zdjęć)
- 3) Stworzenie oraz konfiguracja sztucznej sieci neuronowej
- 4) Trening sieci na podstawie podanych danych
- 5) Testowanie sieci neuronowej
- 6) Zapis modelu do pliku

## Zdefiniowanie argumentów wejściowych programu

Zanim nastąpi krok dotyczący wczytywania danych, najpierw w programie zdefiniowano przetwarzanie argumentów wejściowych. Fragment kodu realizujący tę funkcjonalność przedstawiono na poniższej grafice.

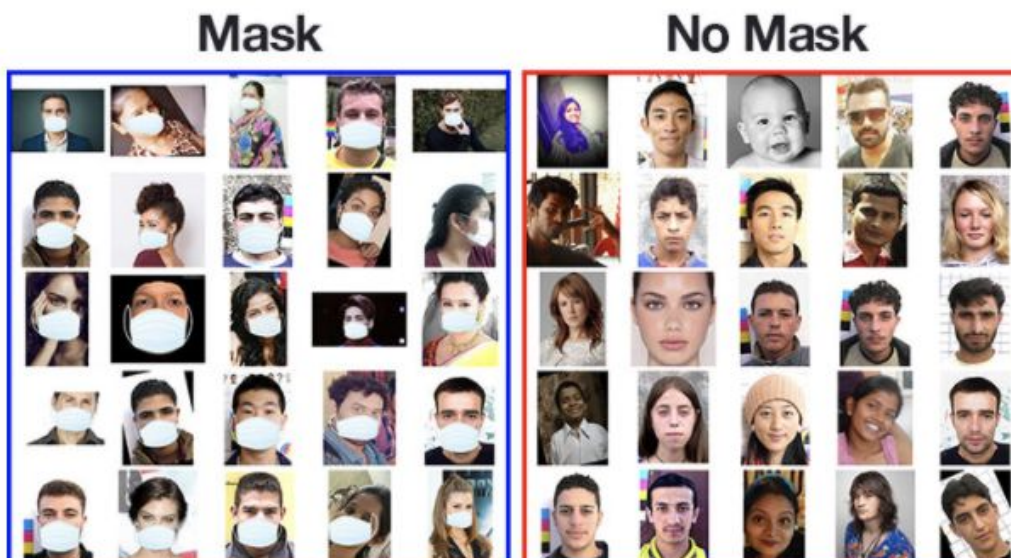
```
# prepare the parser and parse command line arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset",
                default="dataset",
                help="(direcotry path) to input dataset")
ap.add_argument("-m", "--model",
                default="mask_detector.model",
                help="(file path) to output mask detector model")
args = vars(ap.parse_args())
```

Obraz 3 Fragment kodu odpowiadający za przetwarzanie danych wejściowych.

## Wczytanie danych (zdjęć)

Pierwszym znaczącym etapem w działaniu skryptu jest przygotowanie danych do przetworzenia przez sieć neuronową.

Zestaw danych składa się z próbki 690 zdjęć postaci bez masek oraz analogicznej ilości postaci z nałożonymi maskami. Część z nich zostało przedstawione na poniższej grafice. Dane przygotowane do przetworzenia zostały pobrane z repozytorium Github użytkownika *prajnasb*. Link jest podany w rozdziale **7 Źródła**.



Obraz 4 Próbkę zdjęć postaci z maskami oraz bez masek.



Na poniższej grafice przedstawiono fragment kodu odpowiadający za wczytanie danych porównawczych oraz przygotowanie ich do skutecznej analizy przez sieć neuronową.

```
71 print("-I-: loading images...")
72 # list of images in dataset directory
73 imagePath = list(paths.list_images(args["dataset"]))
74 data = []
75 labels = []
76
77 # loop over the images in imagePath
78 for imagePath in imagePath:
79     # get the subdirectory name of image set (with_mask / without_mask)
80     # [-2] is 2nd element counting from end
81     label = imagePath.split(os.path.sep)[-2]
82
83     # load the input image
84     # (224x224) is widely used in deep learning for some reason
85     image = load_img(imagePath, target_size=(224, 224))
86     image = img_to_array(image)
87     # preprocess it using MobileNetV2 preprocessor
88     image = preprocess_input(image)
89
90     # update the data and labels lists
91     data.append(image)
92     labels.append(label)
93
94 # convert the data to NumPy arrays
95 data = np.array(data, dtype="float32")
96 labels = np.array(labels)
97
98 # perform one-hot encoding on the labels
99 # one-hot is (001000000) - only one 1
100 # this clarifies images as value and not some "label"
101 # very important for deep learning as now we can get a one value as output
102 lb = LabelBinarizer()
103 labels = lb.fit_transform(labels)
104 # set categories from labels (its one-hot encoded already)
105 labels = to_categorical(labels)
106
107 # split the data in 2 sets
108 # training - 75%
109 # testing - 25%
110 (trainX, testX, trainY, testY) = train_test_split(data, labels,
111     test_size=0.25, stratify=labels, random_state=42)
112
113 # data augmentation - modify images slightly for much larger data pool
114 # rotate / zoom / shear / scale / flip images to generate more data
115 aug = ImageDataGenerator(
116     rotation_range=20,
117     zoom_range=0.15,
118     width_shift_range=0.2,
119     height_shift_range=0.2,
120     shear_range=0.15,
121     horizontal_flip=True,
122     fill_mode="nearest")
```

Obraz 5 Fragment kodu odpowiadający za wczytanie i wstępne przygotowanie danych.

Zdjęcia w pierwszej kolejności są wczytane do tablicy *imagePaths*. W pętli for od linii 78 do 92 są rozróżniane na te z maską oraz bez niej, a następnie oznaczane według rozróżnienia. Zdjęcia są również skalowane oraz wstępnie przetwarzane przez funkcję *preprocess\_input(image)* dla sieci neuronowej *MobileNetV2*, która jest dostarczona przez moduł Tensorflow. Tablica data przechowuje zdjęcia odpowiednio przetworzone, a tablica label ich oznaczenia według kategorii. Kategorie są odpowiednio przygotowane w liniach od 102 do 105.

W kolejnym kroku dane są dzielone na kolejne 2 kategorie. Pierwsza część dotycząca 75% przypadków to dane przygotowane do trenowania modelu, a pozostałe 25% posłużą do testu wytrenowanej sieci. Dane przygotowane dla sieci neuronowej są umieszczane w tablicach *trainX*, *trainY* do treningu, a *testX* i *testY* do testów za pośrednictwem *train\_test\_split* dostarczonej przez moduł Sklearn.

W ostatnim etapie generowany jest obiekt *ImageDataGenerator* dostarczony przez moduł Tensorflow, który przeprowadza augmentację danych. Przetwarza obrazy poprzez rotacje, przybliżenie, obroty itp. Co pozwala na dużo szybsze potem przetwarzanie tych danych przez sieć neuronową.

## Stworzenie oraz konfiguracja sztucznej sieci neuronowej

Kolejny ważny krok to stworzenie sztucznej sieci neuronowej oraz jej konfiguracja. Odpowiednie parametry mają wpływ na prawidłowe uczenie się jak również za szybkość tego procesu. Odpowiednie dobranie parametrów, oraz poprawna konfiguracja sieci jest kluczowa, ponieważ to od nich zależy poprawne zidentyfikowanie poszukiwanych obiektów.

Kod realizujący ten fragment został przedstawiony na poniższej grafice.

```

134 # using MobileNetV2
135 # using FC (Fully Connected) layers
136 # https://developer.ridgerun.com/wiki/index.php?title=Keras_with_MobilenetV2_for_De
137 # https://keras.io/api/applications/mobilenet/
138
139 # include_top
140 # whether to include the fully-connected layer at the top of the network.
141 # we don't want that because we create our model layers later
142
143 # input_tensor
144 # we use 224px x 224px images in RGB so we input shape (224,224,3)
145 baseModel = MobileNetV2(include_top=False, weights="imagenet",
146 | | | | | input_tensor=Input(shape=(224, 224, 3)))
147
148 # construct the head of the model that will be placed on top of the
149 # the base model
150 headModel = baseModel.output
151
152 # change this parameter to determine how big details are taken into account
153 # affects accuracy (to some extent)
154 # greater value - slightly harder to detect mask (less reliable)
155 # lower value - mask will be detected on small (distant) faces even without mask
156 # (7,7) is max
157 # for example (1,1) will be almost 0-1 detections (~100% mask or ~100% no mask)
158 # and even tiniest element covering mouth/nose will trigger mask to be detected
159 poolSizeTuple = (5,5)
160 # convolutional and pooling layer
161 headModel = AveragePooling2D(pool_size=poolSizeTuple)(headModel)
162 headModel = Flatten(name="flatten")(headModel)
163
164 # relu / softmax
165 # https://keras.io/api/layers/activations/
166
167 # change this parameter to determine how many details are taken into account
168 # affects accuracy (to some extent)
169 # greater value - slightly harder to detect mask (less reliable)
170 ReluLayerSize = 128
171 headModel = Dense(ReluLayerSize, activation="relu")(headModel)
172
173 # we have 2 classes - with_mask / without_mask so we have output space of size 2
174 headModel = Dense(2, activation="softmax")(headModel)
175
176 # place the head FC model on top of the base model (this will become
177 # the actual model we will train)
178 model = Model(inputs=baseModel.input, outputs=headModel)
179
180 # loop over all layers in the base model and freeze them so they will
181 # *not* be updated during the first training process
182 for layer in baseModel.layers:
183     layer.trainable = False
184
185 print("-I-: compiling model...")
186 # optimize model
187 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
188 # compile model
189 model.compile(loss="binary_crossentropy", optimizer=opt,
190 | | | metrics=["accuracy"])

```

Obraz 6 Kod odpowiadający za stworzenie oraz konfigurację sztucznej sieci neuronowej.

Moduł *Tensorflow* dostarcza implementację sieci neuronowej opartej o architekturę *MobileNetV2* co widać w linii 88. Sieć została przystosowana do przetwarzania obrazów zgodnie ze wcześniejszą konfiguracją. W dalszych liniach zdefiniowany jest input i output do konstrukcji modelu w linii 178. Pętla `for` w linii 182 pozwala na znacznie szybsze przeprowadzenie nauczania. Optymalizator wygenerowany dzięki konstruktorowi *Adam()* również pomaga później w szybszym przetwarzaniu. Na koniec model został skompilowany z udziałem wcześniej wygenerowanego optymalizatora.

## Trening modelu sieci neuronowej

Kolejny fragment kodu przedstawiony poniżej odpowiada za trening wcześniej przygotowanego modelu.

```
202 # train network using images pool and with augmented versions
203 print("-I-: training head...")
204 H = model.fit(
205     aug.flow(trainX, trainY, batch_size=BatchSize),
206     steps_per_epoch=len(trainX) // BatchSize,
207     validation_data=(testX, testY),
208     validation_steps=len(testX) // BatchSize,
209     epochs=EPOCHS)
```

Obraz 7 Kod realizujący trening sieci neuronowej

Jako argument do funkcji realizującej trening został podany wcześniej wygenerowany obiekt przeprowadzający augmentację danych, zestaw danych treningowych oraz liczba epok.

## Testowanie sieci neuronowej

Testowanie sieci neuronowej zostało zrealizowane poprzez fragment kodu przedstawiony na poniższej grafice.

```
221 # check model on test set
222 print("-I-: evaluating network...")
223 predIdxs = model.predict(testX, batch_size=BatchSize)
224
225 # get label of detection with larger prediction
226 predIdxs = np.argmax(predIdxs, axis=1)
227
228 # show classification report
229 print(classification_report(testY.argmax(axis=1), predIdxs,
230     target_names=lb.classes_))
```

Obraz 8 Kod realizujący testowanie sieci neuronowej

Funkcje testujące wykorzystują dane wcześniej przygotowane do testowania poprzednim etapem. Funkcja *predict()* generuje przewidywane dane wyjściowe na

podstawie danych wejściowych. Następnie *argmax()* pozwala wybrać bardziej prawdopodobny wynik. W ostatnim kroku zostaje wygenerowany raport z testu.

## Zapis modelu do pliku

Ostatnim etapem jest zapisanie wygenerowanego, wytrenowanego i przetestowanego modelu do pliku. Fragment kodu realizującego to zadanie został przedstawiony na grafice poniżej.

```
232 # save model
233 print("-I-: saving mask detector model...")
234 model.save(args["model"], save_format="h5")
```

Obraz 9 Kod realizujący zapis modelu do plik.

Model zostaje zapisany do pliku o nazwie podanej w argumencie wejściowym skryptu.

## 4.2. Implementacja modelu wykrywającego maskę w obrazie na żywo z kamierki

Implementację modelu wykrywającego maskę w obrazie na żywo z kamierki można podzielić na poszczególne człony:

- 1) Zdefiniowanie argumentów wejściowych programu oraz ich interpretacja
- 2) Definicja interfejsu
- 3) Funkcja odpowiadająca za wykrywanie maski na twarzy z kamierki
- 4) Główna pętla realizująca przebieg programu



## Zdefiniowanie argumentów wejściowych programu oraz ich interpretacja

Definicja argumentów wejściowych programu oraz ich interpretacja zostały przedstawione na poniższej grafice.

```
94 # construct the argument parser and parse the arguments
95 ap = argparse.ArgumentParser()
96 ap.add_argument("-f", "--face",
97     default="face_detector",
98     help="path to face detector model directory")
99 ap.add_argument("-m", "--model",
100     default="mask_detector.model",
101     help="path to trained face mask detector model")
102 ap.add_argument("-c", "--confidence", type=float,
103     default=0.5,
104     help="minimum probability to filter weak detections")
105 args = vars(ap.parse_args())
106
107 # load face model from disk
108 print("-I-: loading face detector model")
109 prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
110 weightsPath = os.path.sep.join([args["face"],
111     "res10_300x300_ssd_iter_140000.caffemodel"])
112 faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
113
114 # load mask model from disk
115 print("-I-: loading mask detector model")
116 maskNet = load_model(args["model"])
117
118 # initialize the video stream and allow the camera sensor to warm up
119 print("-I-: loading video stream")
120 vs = VideoStream(src=0).start()
121 time.sleep(2.0)
122
123 # setup variables
124 filterBW = False
125 filterSharpen = False
126 filterDenoise = False
127 drawContours = True
```

Obraz 10 Kod realizujący definicję argumentów wejściowych oraz ich interpretację.

Program przyjmuje 3 argumenty wejściowe:

- Ścieżkę do modelu zdolnego do wykrywania twarzy
- Ścieżkę do modelu zdolnego do wykrywania maseczki na twarzy
- Minimalną wartość prawdopodobieństwa wymaganą do wskazania, że jest maseczka

W kolejnym kroku wczytywane są oba modele. Następnie uruchamiana jest kamera. Na koniec inicjalizacja domyślnych ustawień interfejsu, który zostanie opisany w kolejnym akapicie.

Funkcja odpowiadająca za wykrywanie maski na twarzy w obrazie z kamery internetowej

Kod realizujący implementację funkcji odpowiadającej za wykrywanie maski na twarzy w obrazie z kamery został przedstawiony na poniższej grafice.

```
27 def detectFaceAndMask(frame, faceNet, maskNet):
28     # get frame dimensions
29     (h, w) = frame.shape[:2]
30     # create blob from image with mean subtraction (104.0, 177.0, 123.0)
31     blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
32     (104.0, 177.0, 123.0))
33
34     # detect faces in blob
35     faceNet.setInput(blob)
36     detections = faceNet.forward()
37
38     # prepare data variables
39     faces = []
40     locs = []
41     preds = []
42
43     # loop over the detections
44     for i in range(0, detections.shape[2]):
45         # extract the confidence (i.e., probability) associated with
46         # the detection
47         confidence = detections[0, 0, i, 2]
48
49         # filter out weak detections by ensuring the confidence is
50         # greater than the minimum confidence
51         if confidence > args["confidence"]:
52             # compute the (x, y)-coordinates of the bounding box for the object
53             box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
54             (startX, startY, endX, endY) = box.astype("int")
55
56             # ensure the bounding boxes fall within the dimensions of the frame
57             (startX, startY) = (max(0, startX), max(0, startY))
58             (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
59
60             # extract the face
61             face = frame[startY:endY, startX:endX]
62             # convert from BGR to RGB
63             face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
64             # resize to 224x224 as this is how mask model is prepared
65             face = cv2.resize(face, (224, 224))
66             # add to array
67             face = img_to_array(face)
68             # preprocess
69             face = preprocess_input(face)
70             face = np.expand_dims(face, axis=0)
71
72             # save face and bounding boxes to variables
73             faces.append(face)
74             locs.append((startX, startY, endX, endY))
75
76             # only check for mask if at least one face was detected
77             if len(faces) > 0:
78                 pred = maskNet.predict(face)
79                 preds.append((pred[0][0], pred[0][1]))
80
81     # return a 2-tuple of the face locations and detection probabilities
82     return (locs, preds)
```

Obraz 12 Kod funkcji odpowiadającej za wykrywanie maski na twarzy w obrazie z kamery.

Funkcja ***detectFaceAndMask*** przyjmuje 3 argumenty:

- frame - pojedyncza klatka obrazu z kamery
- faceNet - model sieci zdolny do wykrywania twarzy
- maskNet - model sieci zdolny do wykrywania maseczki na twarzy

Funkcja ***blobFromImage*** przygotowuje klatkę obrazu z kamerki do przetworzenia przez model wykrywania twarzy w linii 35. Wykryte twarze są zapisywane do zmiennej `detections`.

W pętli od linii 44 twarze są analizowane pod kątem obecności maseczki. Jeśli pewność wykrytej twarzy przekracza wartość podaną w argumencie wejściowym to jest ona następnie przygotowywana pod kątem przetworzenia przez model sieci wykrywający maskę w linii 78. Funkcja zwraca 2 listy w postaci krotki:

- locs - lokalizacje znalezionych twarzy
- preds - wartości prawdopodobieństw wykrycia maski oraz jej braku



## Główna pętla realizująca przebieg programu

Kod przedstawiający przebieg pętli został przedstawiony na dwóch poniższych grafikach.

```
139 # loop over the frames from the video stream
140 while True:
141     # grab the frame from the threaded video stream and resize it
142     # to have a maximum width of 400 pixels
143     frame = vs.read()
144     frame = imutils.resize(frame, width=400)
145
146     # apply filters
147     if filterBW:
148         frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
149         frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2BGR)
150
151     if filterDenoise:
152         frame = cv2.fastNlMeansDenoisingColored(frame, None, 5, 5, 7, 15)
153
154     if filterSharpen:
155         frameBlurred = cv2.GaussianBlur(frame, (0, 0), 9);
156         frame = cv2.addWeighted(frame, 1.5, frameBlurred, -0.5, 0);
157
158     # detect faces and check for mask
159     (locs, preds) = detectFaceAndMask(frame, faceNet, maskNet)
160
161     # loop over the detections
162     for (box, pred) in zip(locs, preds):
163         # unpack the bounding box and predictions
164         (startX, startY, endX, endY) = box
165         (mask, withoutMask) = pred
166
167         # determine the class label and color we'll use to draw
168         # the bounding box and text
169         label = "Mask" if mask > withoutMask else "No Mask"
170         color = (0, mask*255, withoutMask*255) if label == "Mask" else (0, mask*255, withoutMask*255)
171
172         # include the probability in the label
173         label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
174
175         line_length = 3
176         line_width = 1
177
178         # draw contours
179         if drawContours:
180             edged = cv2.Canny(frame[startY:endY, startX:endX], 64, 192)
181             for y in range(startY, endY):
182                 for x in range(startX, endX):
183                     if edged[y - startY, x - startX]:
184                         #frame[y, x, 0] = frame[y, x, 0] - edged[y - startY, x - startX]
185                         frame[y, x] = color
186         else:
187             line_length = 10
188             line_width = 2
189
190         # display the label and bounding box rectangle on the output
191         # frame
192         cv2.putText(frame, label, (startX, startY - 10),
193                     cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
194
```

Obraz 13 Kod przedstawiający fragment głównej pętli realizującej przebieg programu.

W pętli w każdym obiegu odczytywane są poszczególne ramki z obrazu kamery. Następnie są one wstępnie przetwarzane przez filtry zdefiniowane w interfejsie użytkownika. Kolejnym krokiem jest wywołanie funkcji **detectFaceAndMask** opisanej w

rozdziale 4.10, która zwraca informacje o lokalizacji wykrytych twarzy, jak również o prawdopodobieństwie, że są zakryte масечką. W zależności od zwróconej wartości prawdopodobieństwa generowany jest komunikat na obrazie o posiadanej масечке lub jej braku wraz z informacją o pewności tej informacji. Od linii 179 zdefiniowane jest rysowanie konturów twarzy w zależności od tego czy użytkownik włączył tę opcję, a na koniec całość jest nakładana na obraz. Dalsza część przedstawiona jest na poniższym obrazie.

```
194
195     # corner lines on face bounding box
196     cv2.rectangle(frame, (startX, startY), (startX, startY + line_length), color, line_width) # top-left
197     cv2.rectangle(frame, (startX, startY), (startX + line_length, startY), color, line_width)
198
199     cv2.rectangle(frame, (startX, endY), (startX, endY - line_length), color, line_width) # top-right
200     cv2.rectangle(frame, (startX, endY), (startX + line_length, endY), color, line_width)
201
202     cv2.rectangle(frame, (endX, startY), (endX, startY + line_length), color, line_width) # bottom-left
203     cv2.rectangle(frame, (endX, startY), (endX - line_length, startY), color, line_width)
204
205     cv2.rectangle(frame, (endX, endY), (endX, endY - line_length), color, line_width) # bottom-right
206     cv2.rectangle(frame, (endX, endY), (endX - line_length, endY), color, line_width)
207
208     # show the output frame
209     cv2.imshow("Mask Detector", frame)
210     key = cv2.waitKey(1) & 0xFF
211
```

Obraz 14 Druga część kodu przedstawiającego fragment głównej pętli realizującej przebieg programu.

Powyższy fragment rysuje kontury obszaru, na którym wykryto twarz, a następnie wyświetlany jest zaktualizowany stan obrazu z kamery przez program. W ostatniej linii powyższego fragmentu odczytywana jest interakcja użytkownika z interfejsem.

## Definicja interfejsu

Kod z implementacją interfejsu użytkownika został przedstawiony na grafice poniżej.

```
226
227     # if the `q` key was pressed, break from the loop
228     if key == ord("q"):
229         break
230
231     # if the `b` key was pressed, enable BW filter
232     if key == ord("b"):
233         filterBW = not filterBW
234
235     # if the `s` key was pressed, enable Sharpen filter
236     if key == ord("s"):
237         filterSharpen = not filterSharpen
238
239     # if the `d` key was pressed, enable Denoising filter
240     if key == ord("d"):
241         filterDenoise = not filterDenoise
242
243     # if the `c` key was pressed, enable Denoising filter
244     if key == ord("c"):
245         drawContours = not drawContours
246
```

Obraz 15 Kod realizujący implementację interfejsu użytkownika.

Niniejszy fragment kodu znajduje się wewnątrz głównej pętli realizującej przebieg programu, która zostanie opisana w dalszej części tego rozdziału. Funkcjonalności interfejsu zostały opisane w rozdziale dotyczącym instrukcji użytkownika.

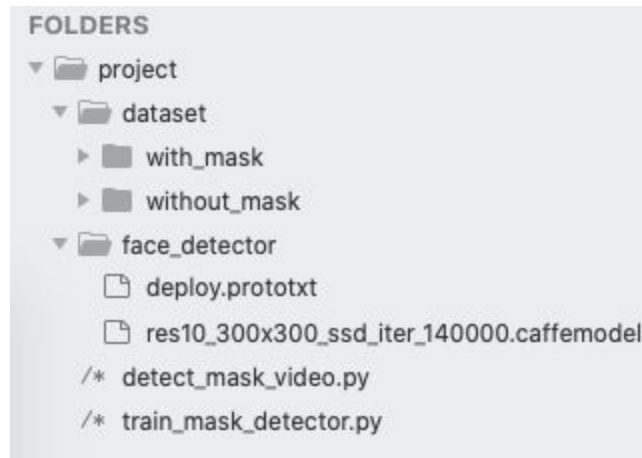
## 5. Instrukcja użytkownika

Instrukcja użytkownika została podzielona na trzy etapy:

- Przygotowanie środowiska programistycznego
- Generowanie modelu zdolnego do wykrywania maseczki na twarzy
- Obsługa skryptu analizującego czy osoby mają maseczkę na obrazie na żywo z kamery.

### 5.1. Przygotowanie środowiska programistycznego

Początkowa struktura plików wejściowych prezentuje się następująco:



Obraz 13 Struktura plików projektowych

W folderze projektowym znajdują się następujące pliki:

- **dataset** - folder przechowujący obrazy przygotowane do wytrenowania sieci:
  - **with\_mask** - folder przechowujący obrazy ludzi z maską
  - **without\_mask** - folder przechowujący obrazy ludzi bez maski
- **face\_detector** - folder przechowujący gotowe pliki modelu służącego do wykrywania twarzy
  - *deploy.prototxt* - plik konfiguracyjny do modelu wykrywania twarzy
  - *res10\_300x300\_ssd\_iter\_140000.caffemodel* - model wykrywający twarze
- **detect\_mask\_video.py** - skrypt python analizujący obraz z kamery na żywo
- **train\_mask\_detector.py** - skrypt python generujący model służący do wykrywania maseczki

Do uruchomienia skryptów wymagane jest zainstalowanie następujących modułów pythona:

- *Tensorflow (Tensorflow-GPU)*
- *Opencv-python*
- *Skylearn*
- *Imutils*
- *Pillow*

Wykorzystując menedżer pakietów pythona należy wykonać następujące komendy w terminalu:

- *pip install tensorflow*
  - alternatywnie: *pip install tensorflow-gpu*
- *pip install opencv-python*
- *pip install sklearn*
- *pip install imutils*
- *pip install pillow*

## 5.2. Generowanie modelu zdolnego do wykrywania maseczki na twarzy

Do wygenerowania modelu zgodnego wykryć czy dana osoba ma ubraną maseczkę czy jej nie ma, służy skrypt *train\_mask\_detector.py*.

Skrypt przyjmuje następujące argumenty:

### 1) Zdefiniowanie folderu z zestawem danych do przetworzenia

Flaga: -d

Argument: nazwa folderu przechowującego zestaw danych przygotowanych do przetworzenia

Domyślnie: *dataset*

### 2) Zdefiniowanie nazwy pliku modelu, który zostanie zwrócony przez program

Flaga: -m

Argument: *nazwa\_pliku.model*

Domyślnie: *mask\_detector.model*

Przykład zastosowania:

```
python train_mask_detector.py -d dataset -m mask_detector.model
```

Skrypt ma również zdefiniowane domyślne argumenty, a więc do uruchomienia treningu sieci wystarczy wykonać:

```
python train_mask_detector.py
```

Oczekiwany wynik w terminalu został przedstawiony na poniższej grafice.

```
piotrek@Piotrs-MacBook-Pro ~/Desktop/project2 python3 train_mask_detector.py -d
dataset -m mask_detector.model
-I-: loading images...
WARNING:tensorflow: `input_shape` is undefined or non-square, or `rows` is not in [96,
128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
2020-06-15 13:05:46.581035: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your
CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
FMA
2020-06-15 13:05:46.601784: I tensorflow/compiler/xla/service/service.cc:168] XLA service
0x7fa3105e7940 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
2020-06-15 13:05:46.601807: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): Host, Default Version
-I-: compiling model...
-I-: training head...
Epoch 1/5
129/129 [=====] - 33s 258ms/step - loss: 0.1583 - accuracy:
0.9448 - val_loss: 0.0284 - val_accuracy: 0.9942
Epoch 2/5
129/129 [=====] - 33s 258ms/step - loss: 0.0345 - accuracy:
0.9922 - val_loss: 0.0160 - val_accuracy: 0.9971
Epoch 3/5
129/129 [=====] - 31s 244ms/step - loss: 0.0240 - accuracy:
0.9942 - val_loss: 0.0123 - val_accuracy: 0.9971
Epoch 4/5
129/129 [=====] - 33s 257ms/step - loss: 0.0212 - accuracy:
0.9932 - val_loss: 0.0123 - val_accuracy: 0.9913
Epoch 5/5
129/129 [=====] - 33s 257ms/step - loss: 0.0156 - accuracy:
0.9961 - val_loss: 0.0107 - val_accuracy: 0.9971
-I-: evaluating network...
              precision    recall  f1-score   support

   with_mask         1.00      0.99      1.00       172
 without_mask         0.99      1.00      1.00       172

   accuracy
 macro avg          1.00      1.00      1.00       344
weighted avg          1.00      1.00      1.00       344

-I-: saving mask detector model...
```

Obraz 16 Rezultat poprawnie wykonanego skryptu train\_mask\_detector.py

Rezultatem wykonania programu powinien być plik zawierający model zdolny do wykrywania maseczki na twarzy o nazwie *mask\_detector.model*



### 5.3. Obsługa skryptu analizującego czy osoby mają maseczkę na obrazie na żywo z kamery

Wcześniej utworzony plik *mask\_detector.model* jest konieczny do wykonania skryptu *detect\_mask\_video.py*, który uruchomi kamerę wbudowaną w urządzenie oraz na żywo będzie analizował czy postacie na obrazie mają założoną maseczkę czy nie.

Skrypt przyjmuje następujące argumenty:

- 1) **Zdefiniowanie folderu przechowującego model wykrywania twarzy**  
Flaga: -f  
Argument: nazwa folderu przechowującego model wykrywania twarzy  
Domyślnie: *face\_detector*
- 2) **Zdefiniowanie modelu do wykrywania maseczki na twarzy**  
Flaga: -m  
Argument: *nazwa\_pliku\_modelu*  
Domyślnie: *mask\_detector.model*
- 3) **Zdefiniowanie minimalnej wartości pewności, dla której program zakłada, że dana osoba ma założoną maseczkę**  
Flaga: -c  
Argument: wartość pewności za zakresu 0 - 1  
Domyślnie: 0.5

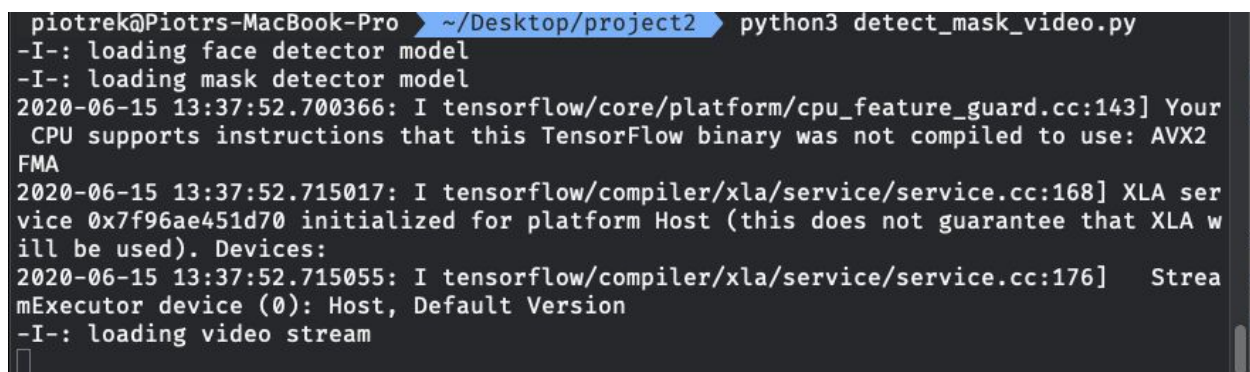
Przykład zastosowania:

```
python detect_mask_video.py -f face_detector -m mask_detector.model -c 0.5
```

Skrypt ma również zdefiniowane domyślne argumenty, a więc do uruchomienia treningu sieci wystarczy wykonać:

```
python detect_mask_video.py
```

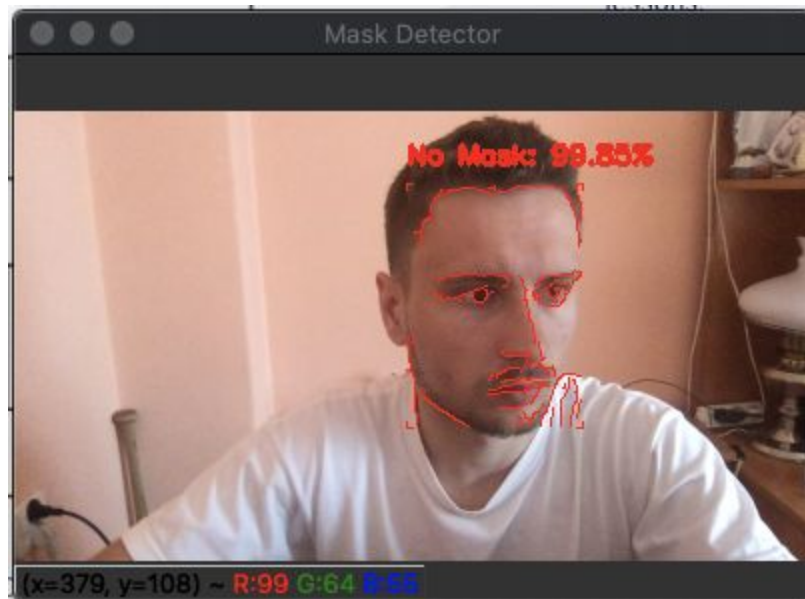
Oczekiwany wynik w terminalu został przedstawiony na poniższej grafice.



```
piotrek@Piotrs-MacBook-Pro ~/Desktop/project2$ python3 detect_mask_video.py
-I-: loading face detector model
-I-: loading mask detector model
2020-06-15 13:37:52.700366: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your
CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
FMA
2020-06-15 13:37:52.715017: I tensorflow/compiler/xla/service/service.cc:168] XLA ser
vice 0x7f96ae451d70 initialized for platform Host (this does not guarantee that XLA w
ill be used). Devices:
2020-06-15 13:37:52.715055: I tensorflow/compiler/xla/service/service.cc:176] Strea
mExecutor device (0): Host, Default Version
-I-: loading video stream
```

Obraz 17 Rezultat poprawnie wykonanego skryptu *detect\_mask\_video.py*

Oczekiwany wynik otwartego programu został przedstawiony na poniższej grafice.



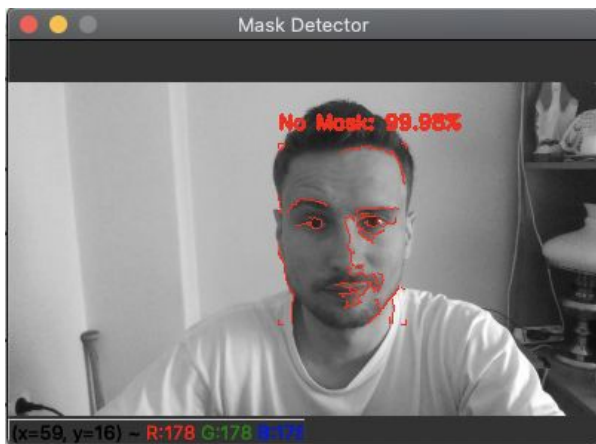
Obraz 18 Działający program *detect\_mask\_video.py*

Program udostępnia następujące opcje obsługi:

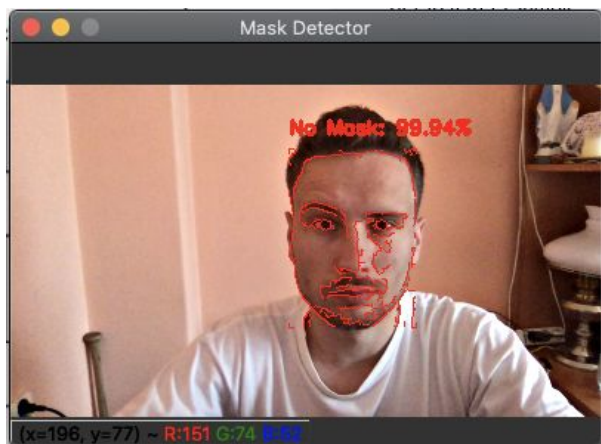
- 1) Funkcja: Wyłączenie programu  
Klawisz: q
- 2) Funkcja: Włączenie/Wyłączenie filtru czarno białego  
Klawisz: b
- 3) Funkcja: Włączenie/Wyłączenie filtru wyostrającego  
Klawisz: s
- 4) Funkcja: Włączenie/Wyłączenie filtru odszumiającego  
Klawisz: d
- 5) Funkcja: Włączenie/Wyłączenie rysowania konturów  
Klawisz: c

Rezultat działania poszczególnych opcji został przedstawiony na poniższych grafikach.

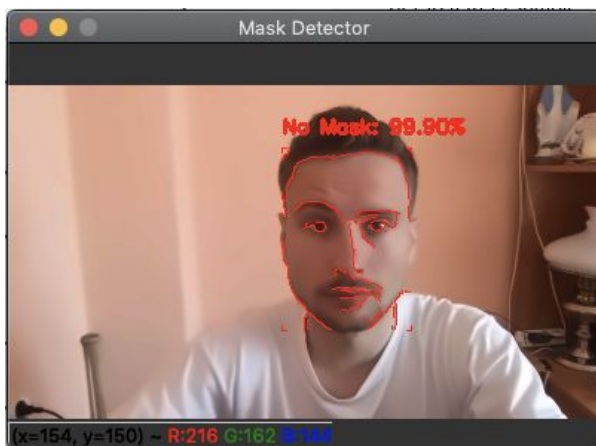




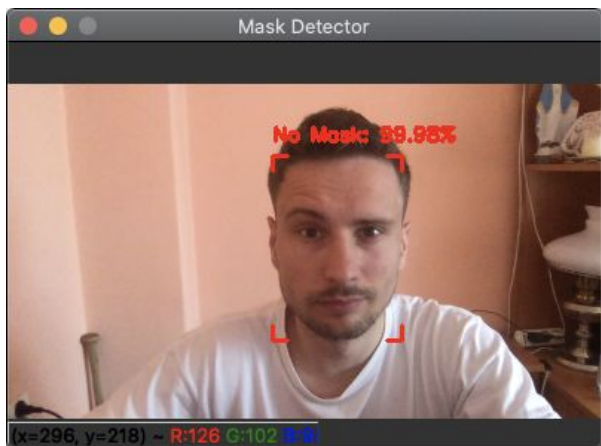
Obraz 19: Włączony filtr czarno biały



Obraz 20: Włączony filtr wyostrażający



Obraz 21: Włączony filtr odszumiający



Obraz 22: Wyłączone kontury twarzy



Obraz 23: Prawidłowe wykrycie maseczki

## 6. Podsumowanie

Wszystkie cele projektu zostały osiągnięte. System wykrywa twarze i skutecznie ocenia czy osoba na obrazie ma założoną maskę. W ramach optymalizacji przechwytywany obraz jest zmniejszany, a mimo to w przypadku filtra odszumiającego ilość klatek na sekundę na wideo drastycznie spada. Zaobserwowano jednak pozytywny wpływ zastosowania filtrów przed wykrywaniem masek w konkretnych przypadkach. Przykładowo zastosowanie filtra czarno-białego (*BW*) pozwala lepiej wykrywać maski z kolorowymi, niejednorodnymi kształtami. Natomiast filtr wyostrzający lepiej radzi sobie w sytuacjach gdy obraz jest niedoświetlony.

Zdolność do wykrywania twarzy jest bardzo wysoka. Przy jednej postaci mieszczącej się w obrazie wartości prawdopodobieństwa są zbliżone do 99%. Przy pochylaniu lub odchyleniu twarzy te wartości ulegają zmniejszeniu. Ma to szczególne znaczenie w sytuacji kiedy postać oddala się od kamery, gdyż od około 2 m oddalenia od kamery program zaczyna się mylić przy niekorzystnym ustawieniu twarzy. Problem ten można jednak zmniejszyć, dobierając odpowiednie parametry przy uczeniu sieci.

Przykład działania programu, oraz efekty testowania zostały przedstawione na nagraniach dostępnych pod linkiem [examples](#).

Skrypty stworzone na potrzeby realizacji projektu umieszczone zostały na dysku google i dostępne są pod linkiem [Kod źródłowy](#).

## 7. Źródła

- Dokumentacja *Keras* - <https://keras.io/>
- Dokumentacja *OpenCV-Python* - <https://github.com/skvark/opencv-python>
- Dokumentacja *imutils* - <https://github.com/jrosebr1/imutils>
- Dokumentacja *Scikit-Learn* <https://scikit-learn.org/stable/>
- Repozytorium zdjęć przeznaczonych do nauczania  
[https://github.com/prajnasb/observations/tree/master/experiements/data/with\\_mask](https://github.com/prajnasb/observations/tree/master/experiements/data/with_mask)