Projekt Indywidualny/Zespołowy na kierunku Automatyka i Robotyka Stosowana

Politechnika Warszawska
Wydział Elektryczny

Rozpoznawanie twarzy z kamery

Jakub Belka
Wojciech Staszewski
Dominik Urbanowicz
Semestr VI studiów I stopnia, kierunek Automatyka i Robotyka Stosowana
Wersja 2, 27.06.2019

Spis treści

1. Wprowadzenie

a) założenia projektowe

Założeniem projektu jest napisanie programu w języku Python, wykorzystującego sztuczną sieć neuronową, który będzie rozpoznawać twarze na podstawie obrazu z kamery i odpowiednio dobranego zbioru danych (zdjęć). Swoje zastosowanie ma znaleźć w sklepach, gdzie kamera umieszczona jest nieruchomo przy suficie. W związku z tym twarze osób wchodzących do sklepu “widziane” są wyłącznie od góry. Dodatkowym utrudnieniem jest stałe oświetlenie oraz ograniczona ilość klatek, na których dana twarz się znajduje (klienci przeważnie szybko przechodzą obok kamery w związku z czym mamy do dyspozycji tylko kilkusekundowe nagrania).

*Program napisany jest w języku Python w środowisku PyCharm;
*Wykorzystane do tego zostały narzędzia takie jak: OpenCV, TensorFlow, Keras, Caffe;
*Do samego wykrycia twarzy skorzystaliśmy z Caffe DNN Face Detector;
*Do rozpoznawania twarzy użyliśmy sieci stworzonej w oddzielnym skrypcie.

b) podział obowiązków

Jakub: - Przygotowanie bazy danych (obróbka obrazu, data augmentation),
- Ekspozycja wyników predykcji (rysowanie, pisanie przy użyciu OpenCV),
- Planowanie architektury i implementacja własnej sieci konwolucyjnej.

Wojciech: - Przygotowanie środowiska i potrzebnych bibliotek,
- Implementacja modelu Caffe wyszukującego twarze,
- Planowanie, implementacja i uczenie własnej sieci konwolucyjnej.

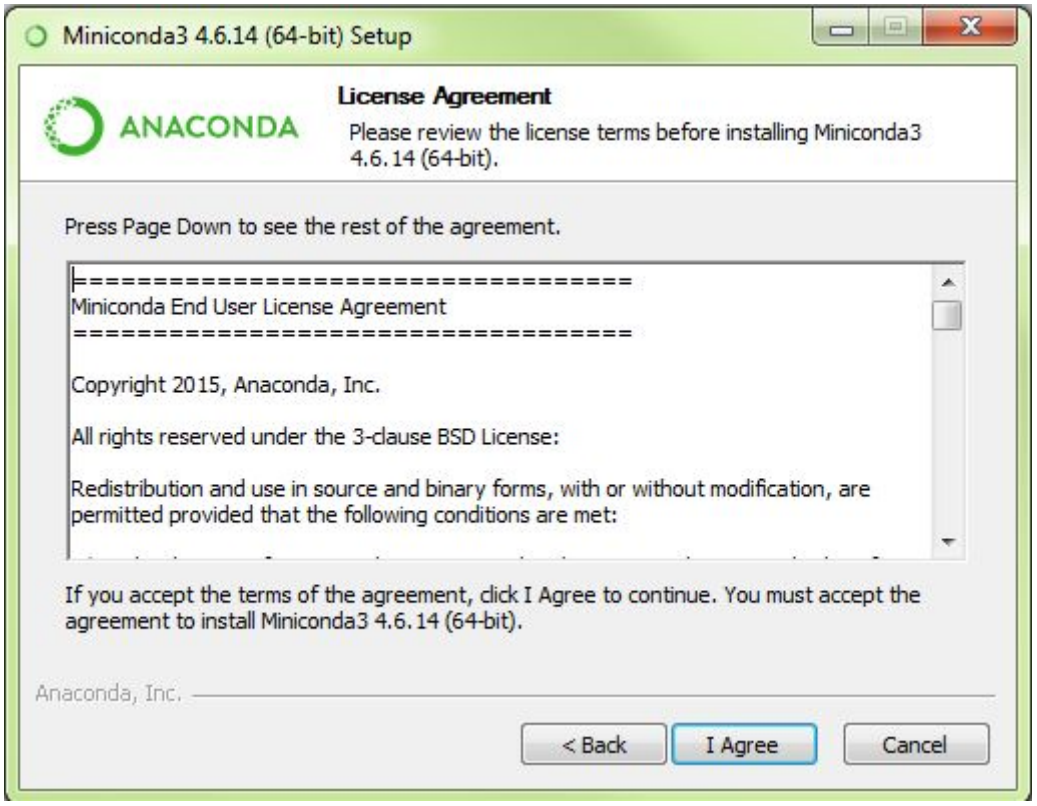
Dominik: - Planowanie i uczenie własnej sieci konwolucyjnej.

2. Przygotowanie środowiska programistycznego

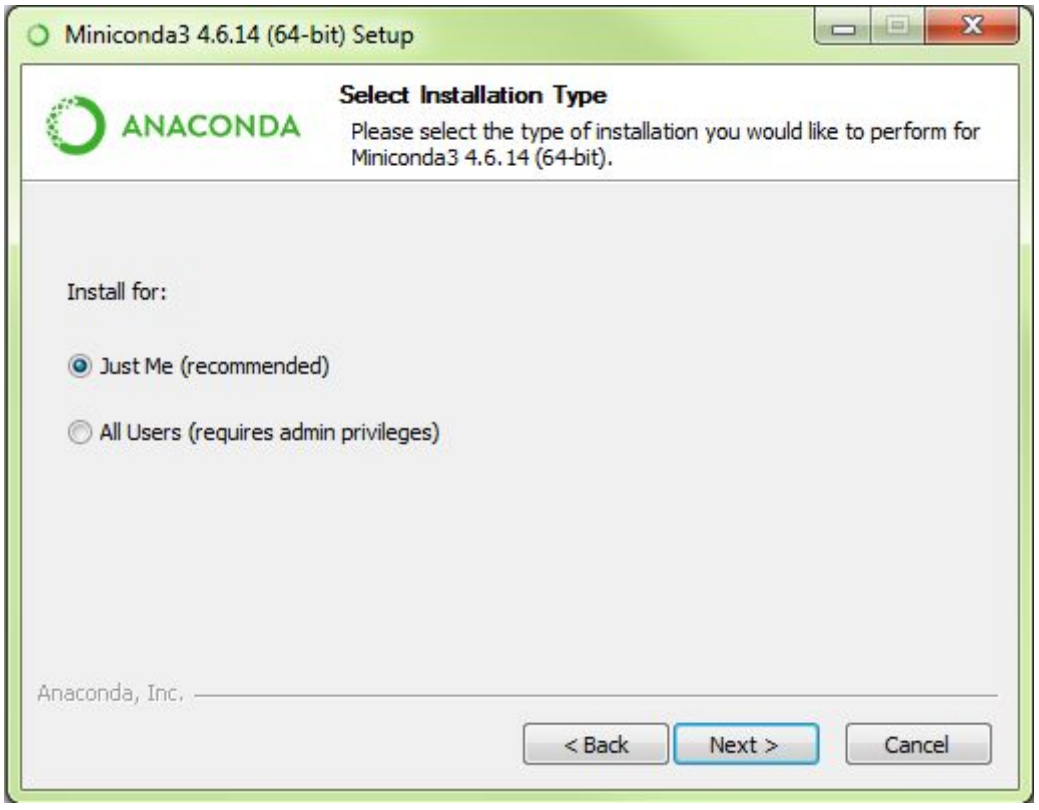
W internecie można znaleźć wiele instrukcji i poradników co jest potrzebne do przygotowania środowiska programistycznego, a także jak w pełni zainstalować wymagane biblioteki. Nie zawsze jednak są one w pełni zrozumiałe, bądź szczegółowo opisane. Tym samym poniżej zamieszczamy kompletny opis krok po kroku jak zainstalować wszelkie potrzebne komponenty i narzędzia. Poniższa instrukcja dotyczy systemu Windows w wersji siódmej i może się różnić w stosunku do innych wersji, bądź rodzajów systemów operacyjnych.

W pierwszej kolejności musimy pobrać z internetu i zainstalować oprogramowanie obsługujące język programowania Python np.: *PyCharm* w wersji *Community*. Podczas instalacji wybieramy wszystkie zalecane ustawienia. W kolejnym kroku instalujemy Python'a oraz biblioteki obsługujące zaawansowane funkcje związane z wykorzystaniem sztucznych sieci neuronowych. Takimi bibliotekami są: *tensorflow* (wsparcie dla wersji CPU i GPU) oraz *Keras* (systemy operacyjne, które są obsługiwane przez wspomniane biblioteki to: Windows, Mac oraz Linux). W obecnym momencie - czerwiec 2019 - najnowszą wersją jest Python 3.7.3. Do instalacji wykorzystamy oprogramowanie *Miniconda (*lżejsza wersja pełnego programu - *Anacondy*), które można pobrać z linku: *https://docs.conda.io/en/latest/miniconda.html.* Wybieramy interesującą nas wersję systemu, pobieramy i instalujemy. Poniżej przedstawiamy proces instalacji:

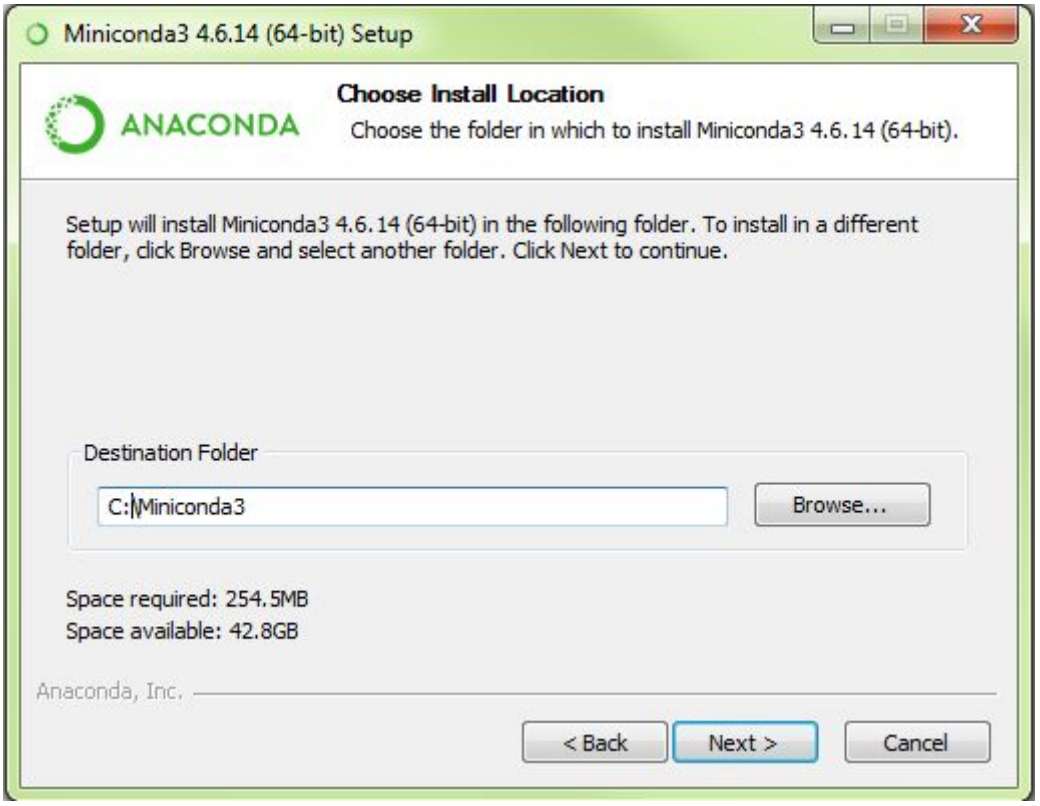
Po pojawieniu się pierwszego okienka wciskamy przycisk *Next* aby przejść dalej.



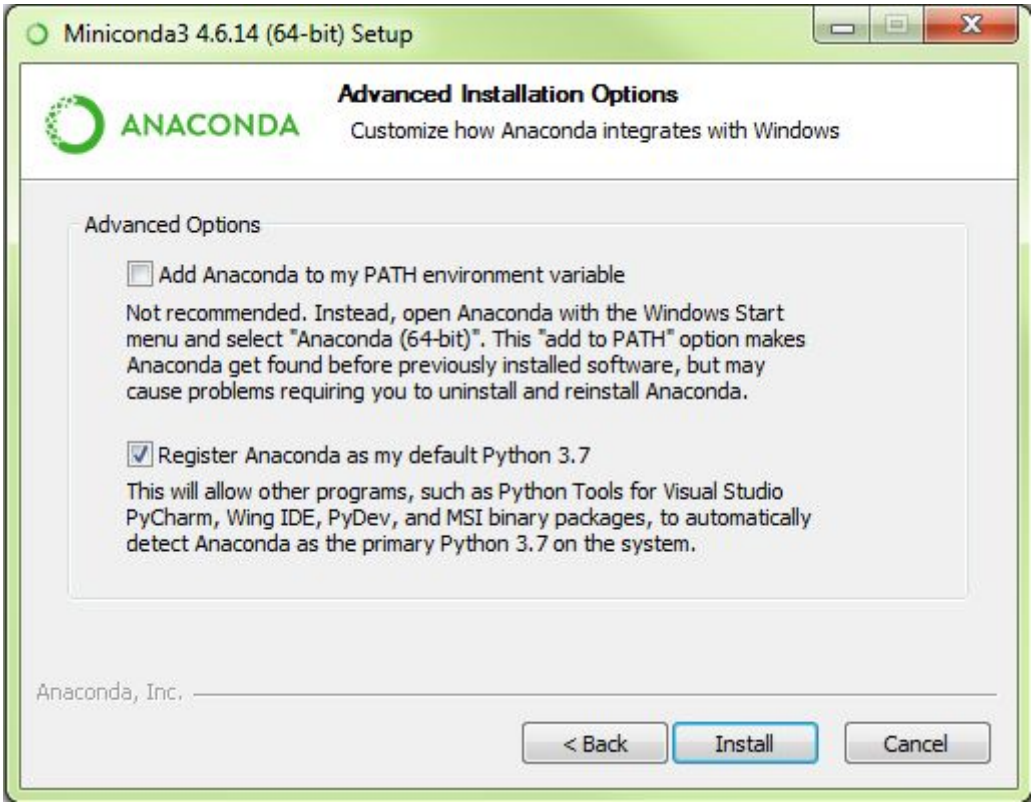
Jeżeli akceptujemy licencję wciskamy *I Agree* i przechodzimy do kolejnego okna.



Wybieramy instalację dla jednego użytkownika i klikamy *Next.* (W przypadku jeśli mamy stworzonego więcej niż jednego użytkownika na komputerze, możemy wybrać opcję *All Users (requires admin privileges)*, jednak to spowoduje, że program zainstaluje się w folderze root wybranego dysku)

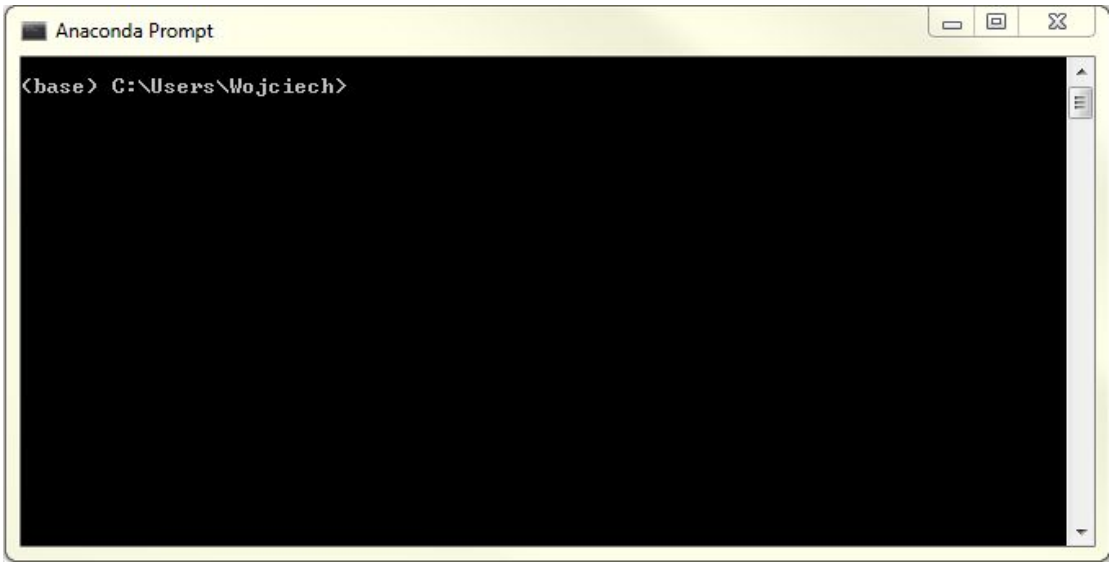


Wybieramy ścieżkę docelową, gdzie chcemy aby program się zainstalował. W naszym przypadku jest to: “C:\Users*nazwa_użytkownika*\Miniconda3”. Klikamy *Next* i przechodzimy dalej.



W przedstawionym oknie należy zaznaczyć pole numer 2. Ustalamy w ten sposób powiązanie *Pythona* z “Anacondą”. Jest to ważne ze względu na dalsze zastosowanie Anacondy do instalacji bibliotek wykorzystywanych w *Pythonie.* Następnie klikamy Install i po zakończonej instalacji zamykamy okno dialogowe. Pole numer 1 możemy pominąć.

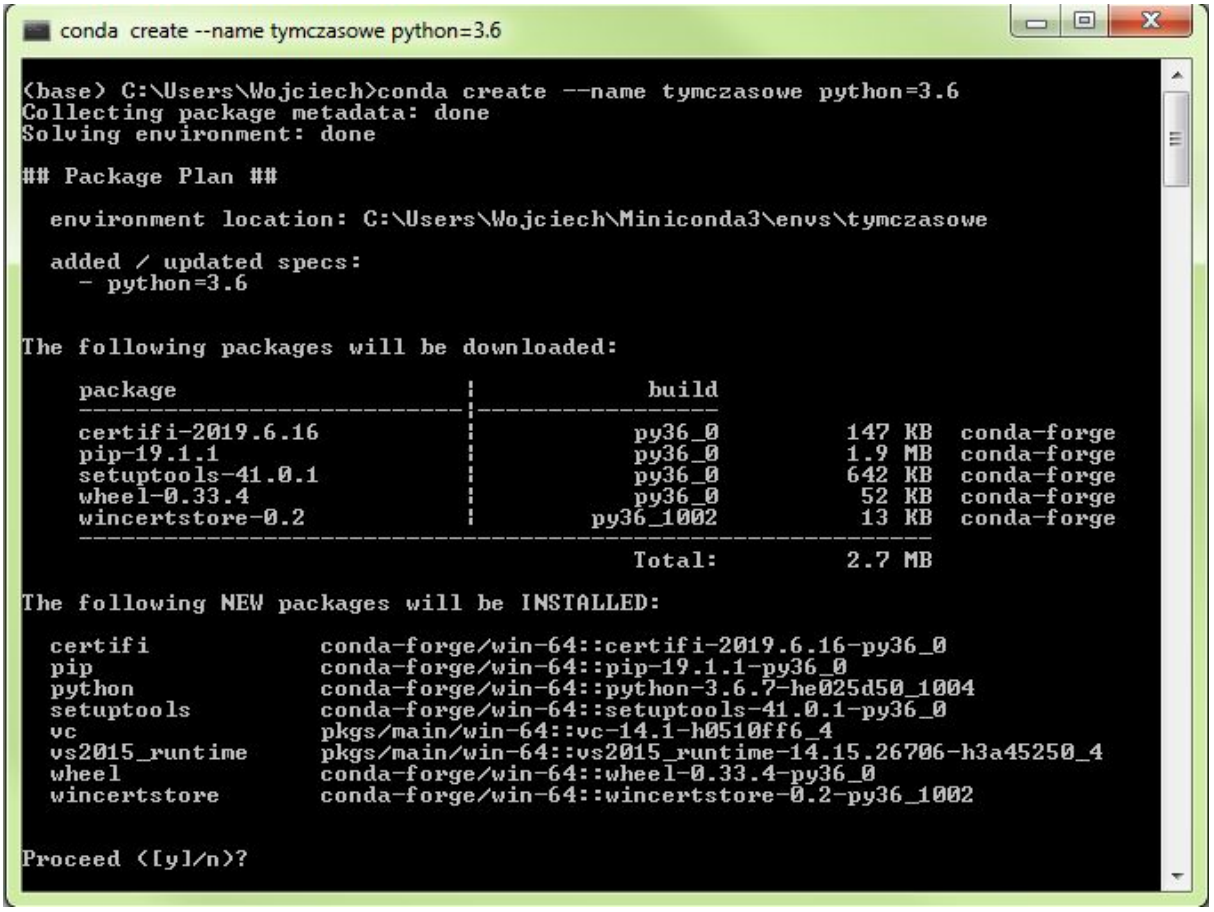
W tym momencie mamy już oprogramowanie do pisania naszego kodu: *PyCharm*, a także program do instalacji bibliotek. W kolejnym kroku uruchamiamy program *Anaconda Prompt* i przechodzimy do tworzenia środowiska dla naszych bibliotek.



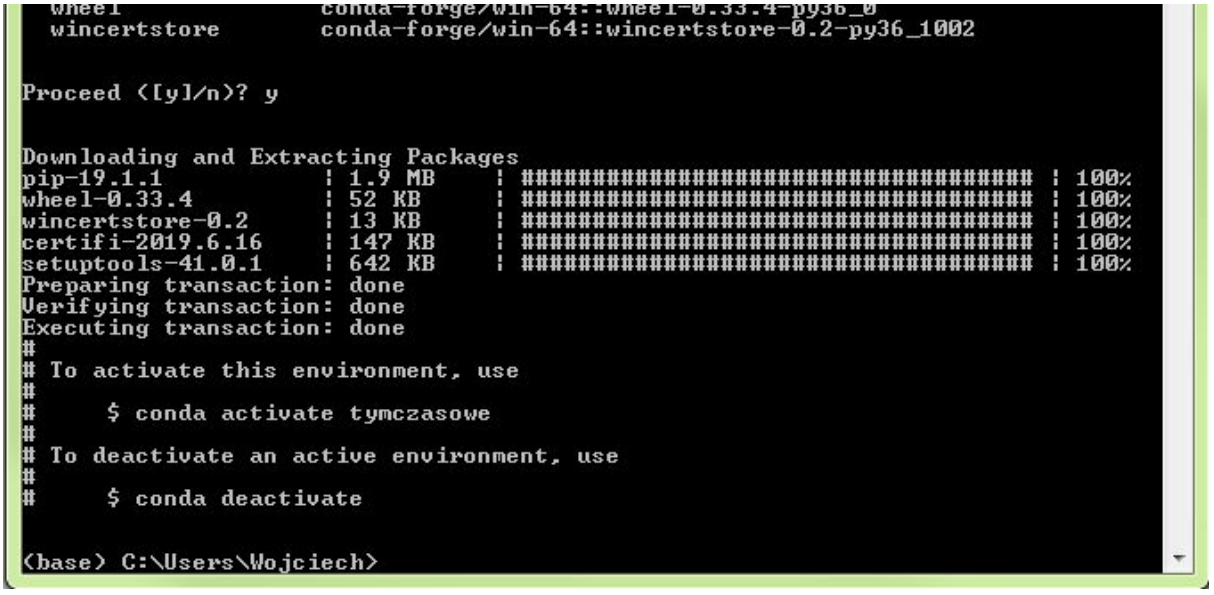
Po otwarciu programu widzimy okno poleceń jak na obrazku. Wpisujemy pierwsze polecenie, które wygląda następująco:

```
conda create --name NAZWA_FOLDERU python=3.6.
```

W ten sposób utworzyliśmy folder, w którym będą instalowane nasze biblioteki. W moim przypadku jest to folder: *C:\Users\Wojciech\Miniconda3\envs\tymczasowe (NAZWA_FOLDERU = tymczasowe).* (TensorFlow obecnie nie współpracuje z Pythonem w wersji 3.7).

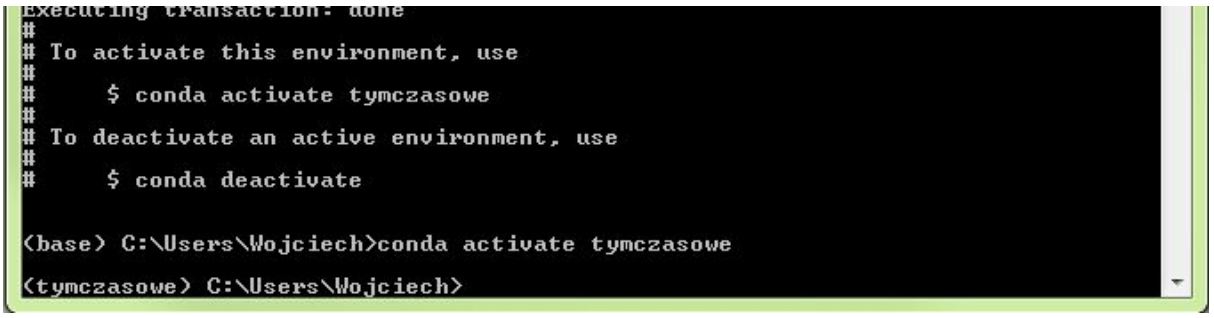


Po przesłaniu polecenia klawiszem *Enter* ukazują nam się elementy, które zostaną zainstalowane. Jeżeli akceptujemy to wpisujemy “y”i zatwierdzamy *Enter*’em.



Po automatycznym pobraniu kilku “paczek” następnie przechodzimy do utworzonego katalogu za pomocą polecenia:

```
conda activate NAZWA_FOLDERU
```



Następnie tym samym sposobem instalujemy potrzebne biblioteki:

```
pip install --upgrade tensorflow==1.13.1
pip install --upgrade keras==2.2.4
pip install --upgrade matplotlib
pip install --upgrade numpy
pip install --upgrade opencv-python
```

Z każdym wpisaniem komendy i akcpetacją przyciskiem *Enter*, zaczynają pobierać się i instalować kolejne dane. Po zainstalowaniu wszystkich powyższych możemy zamknąć okno poleceń programu *Anaconda Prompt.* Doinstalowanie innych bibliotek jest możliwe w późniejszym czasie tym samym sposobem lub w programie PyCharm. Nasze środowisko jest gotowe do uruchomienia.

3. Narzędzia potrzebne do realizacji projektu

Do realizacji projektu korzystaliśmy z następujących narzędzi:

a) PyCharm - zintegrowane środowisko programistyczne dla języka programowania Python firmy JetBrains. Pozwala na edycję i analizę kodu źródłowego, graficzny debugger, uruchamianie testów, integrację z systemem kontroli wersji. Wszystkie linie kody zostały utworzone w tym właśnie środowisku.

b) Tensorflow - Otwartoźródłowa biblioteka stworzona przez Google. Wykorzystywana jest w uczeniu maszynowym i głębokich sieciach neuronowych. Określana jest jako "silnik" uczenia maszynowego. Tensorflow udostępniony jest na zasadzie Open Source.

c) Keras - to również ogólnodostępna biblioteka wykorzystywana do uczenia maszynowego. Współpracuje z tensorflow'em, znacznie upraszcza pracę z sieciami neuronowymi.

d) OpenCV - zaawansowana ogólnodostępna biblioteka do obróbki obrazu. Pozwala w szybki sposób przekonwertować obraz np. do obrazu szarościowego, czy też wykonać na obrazie szereg przekształceń potrzebnych w technice data augmentation. Dzięki OpenCV mogliśmy również wyświetlić wyniki działania naszego programu na obrazie z kamery co wpływa na komfort użytkownika.

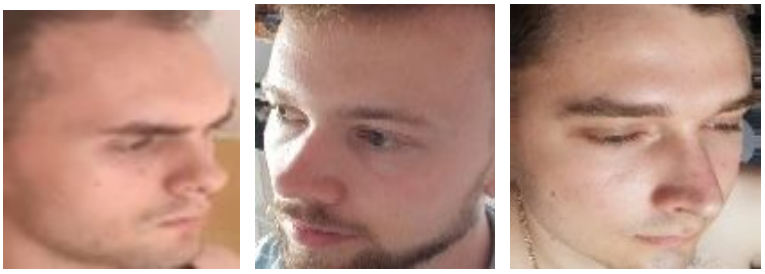
e) Caffe DNN Face Detector - Wytrenowany już model do wyszukiwania twarzy na zdjęciach od Caffe. Model ten został stworzony z bardzo wielu danych uczących co pozwala na bardzo dokładne zlokalizowanie twarzy na zdjęciu. W naszym projekcie chcieliśmy skupić się na samym rozpoznawaniu twarzy w związku z tym do wyszukiwania ich skorzystaliśmy z gotowego modelu.

4. Tworzenie bazy danych

Program rozpoznaje twarze które znajdują się w bazie danych. W związku z założeniem, że program ma działać w sklepie, ilość zdjęć przypadająca na jedną osobę została ograniczona do 50. W dodatku wszystkie zdjęcia robione są z góry, w jednakowych warunkach oświetleniowych.

Zdjęcia poddane są detekcji twarzy przez gotowy Caffe DNN Face Detector, a następnie wycięte zostają same twarze i przesłane do folderu TRAIN. W ten sposób uczymy sieć rozpoznawać wyłącznie same twarze, a nie otoczenie w którym zdjęcie do bazy danych zostało wykonane. W bazie danych do zarejestrowane są 3 osoby.

Przykładowe zdjęcia z bazy danych:



Program do wycinania twarzy i zapisywania ich w oddzielnym folderze:

```
net = cv2.dnn.readNetFromCaffe("deploy.prototxt.txt", "res10_300x300_ssd_ite

for filename in os.listdir('/Users/kubab/PycharmProjects/ProjektInd/it/'):
    filename2=('/Users/kubab/PycharmProjects/ProjektInd/it/' + filename)
    img = cv2.imread(filename2)
    image = cv2.resize(img, (300, 300))
    (h, w) = image.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
                                 (300, 300), (104.0, 177.0, 123.0))


    net.setInput(blob)
    detections = net.forward()


    for i in range(0, detections.shape[2]):

        confidence = detections[0, 0, i, 2]

        if confidence > 0.5:

            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            y = startY - 10 if startY - 10 > 10 else startY + 10
            cropped = image[startY - w:endY, startX - h:endX]
```

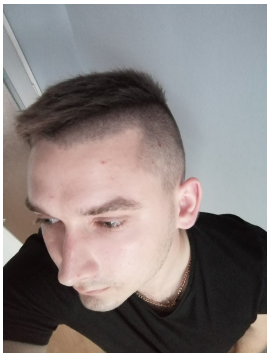
50 podobnych do siebie zdjęć na jedną osobę to dość mała ilość by wystarczająco nauczyć sieć neuronową rozpoznawania osób. W związku z tym skorzystaliśmy z techniki **data augmentation.** Polega ona na odpowiedniej obróbce każdego ze zdjęć na różne sposoby i dodawanie każdej kolejnej obróbki do bazy danych. W ten sposób z jednego zdjęcia można wyciągnąć dużo więcej informacji stosując różne transformacje.

Wykorzystaliśmy do tego 4 metody przekształcenia obrazu:

- Zmiana jasności - Ma to rozwiązać problemu stałego oświetlenia w sklepie. Każde zdjęcie jest kilka razy rozjaśniane i przyciemniane, a parametry określające sposób naświetlania za każdym razem dobierane są w sposób losowy;
- Rotacja - Rotacja w lewo i w prawo. Kąt obrotu również dobierany jest w sposób losowy, maksymalnie 75 stopni;
- Przybliżenie/rozciąganie - Zniekształcenia geometryczne obrazu;
- Nakładanie szumu.

Wszystkie operacje wykonane zostały dzięki bibliotece **imgaug.**

Poniżej przedstawione jest oryginalne zdjęcie, oraz przykładowe jego odpowiedniki po przekształceniach.



↓

Po wykorzystaniu wyżej wymienionych operacji z dostępnych 50 zdjęć otrzymaliśmy ich **1800**, co oznacza, że z jednego zdjęcia z folderu treningowego uzyskujemy 36 jego odpowiedników. Największą ich część stanowi zmiana jasności, ponieważ we wcześniejszych próbach wytrenowany model był bardzo wrażliwy na światło, tzn. działał poprawnie i rozpoznawał osoby wyłącznie w określonych warunkach oświetleniowych.
Wszystkie zdjęcia zostały przeniesione do folderu **images** podzielonego na foldery **train** oraz **validation**. Folder train zawierał po 1800 zdjęć dla każdej klasy. W folderze **validation** umieszczone zostało po 100 zdjęć dla każdej klasy. Zdjęcia te robione były z różnych kątów oraz frontalnie (nie tylko od góry jak w przypadku train).

5. CNN w rozpoznawaniu obrazów

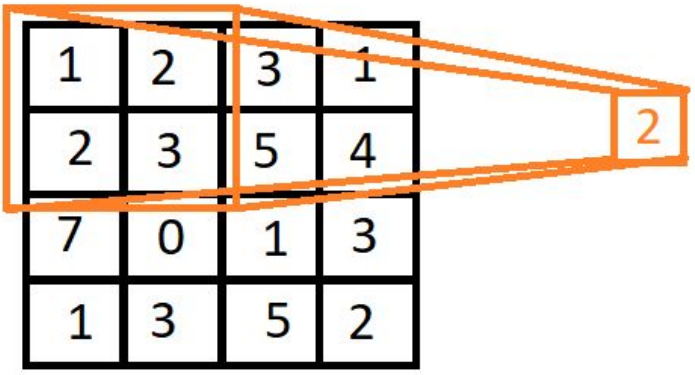
Do rozpoznawania obrazów używa się CNN - konwolucyjnych sieci neuronowych(Convolutional Neural Network). Obrazy interpretowane są tu jako mapa pikseli, a cechy wyszukiwane są wśród danych o bliskim sąsiedztwie. Cały proces rozpoznawania twarzy można przedstawić za pomocą trzech kroków:

5.1) Operacja konwolucji - wyznaczanie cech obrazu

Operacja konwolucji w analizie obrazu interpretowana jest jako filtrowanie.
W celu wyodrębnienia cech dokonuje się operacji konwolucji (inaczej splotu) polegającej na przesuwaniu różnych rodzajów filtrów po całym obrazie. Filtry te to małe jednostki (“okienka” np. 5x5 pikseli). W ten sposób dla każdego fragmentu obrazu oblicza się splot pomiędzy nim a filtrem. Zależnie od tego z ilu warstw zbudowana jest sieć neuronowa, przez tyle różnych filtrów zostanie przepuszczony obraz. Każdy filtr pozwala na wygenerowanie pojedynczego rodzaju cech. Im głębsza staje się nasza sieć tym bardziej abstrakcyjne cechy potrafi ona wyciągnąć z obrazu.

5.2) Operacja poolingu - redukcja informacji

Następnie przeprowadzana jest warstwa poolingu (redukcji). Pozwala ona zredukować ilość informacji znajdującej się na obrazie. Po wykonaniu już operacji konwolucji na obrazie i wyciągnięciu odpowiednich cech, obraz zostaje zubożony i zmniejszony, po czym znów można poddać go filtracji.



5.3) Spłaszczanie i uczenie

Wszystkie mapy cech, które powstały po wcześniejszych operacjach zostają spłaszczone do jednego jednowymiarowego wektora z wartościami. Dopiero ten wektor zostaje wysłany do sieci neuronowej i poddany procesowi uczenia. Sieć neuronowa na podstawie tego wektora uczy się i wyszukuje zależności między wektorem, który dostała na wejście a klasą z której pochodzi dany obraz.

6. Zastosowanie sieci neuronowych w praktyce.

a) wyszukiwanie twarzy

Do wyszukania twarzy na obrazie użyliśmy gotowego już modelu sieci neuronowej od Caffe.

Wczytanie modelu:

```
#Wczytanie modelu Caffe
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe("deploy.prototxt.txt", "res10_300x300_ssd_iter_140000.caffemodel")
```

Spakowanie klatki z kamery do ,,bloba" i wysłanie jej do sieci, gdzie dochodzi do detekcji twarzy:

```
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
                             (300, 300), (104.0, 177.0, 123.0))

# wysłanie bloba do sieci

net.setInput(blob)
detections = net.forward()

for i in range(0, detections.shape[2]):
    # confidence
    confidence = detections[0, 0, i, 2]

    if confidence < 0.5:
        continue
```

Jak widać na powyższej grafice dalsza część programu (rozpoznanie twarzy) odbywa się w pętli **for** po wykrytych twarzach co pozwala na poprawne działanie programu nawet gdy na obrazie z kamery znajdzie się więcej niż jedna osoba.

Model ten pozwala z dużym prawdopodobieństwem określić gdzie i czy wgl. znajduje się twarz na zdjęciu. W naszym programie określiliśmy próg prawdopodobieństwa, którego przekroczenie oznacza, że znaleziony obiekt traktujemy jako twarz. Wynosi on 50% co powoduje, że twarze rozpoznawane są zarówno z małej jak i dużej odległości, nawet do kilku metrów od kamery. Wykryta twarz zostaje obrysowana czerwoną ramką, wycięta i przekazana do dalszej części programu, tzn. do naszej własnej sieci neuronowej.

Obrys wyszukanej twarzy:

```
y = startY - 10 if startY - 10 > 10 else startY + 10
cv2.rectangle(frame, (startX, startY), (endX, endY),
              (0, 0, 255), 2)
cv2.putText(frame, text, (startX, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 0, 255), 2)
```

Zapis twarzy w formacie jpg:

```
cropped = frame[startY - h:endY, startX - w:endX]
cropped_photo = cv2.imwrite(os.path.join(path_output, 'elo.jpg'), cropped)
gray_image = cv2.cvtColor(cropped, cv2.COLOR_BGR2GRAY)  # COLOR_BGR2GRAY(capture)
```

b) rozpoznawanie twarzy

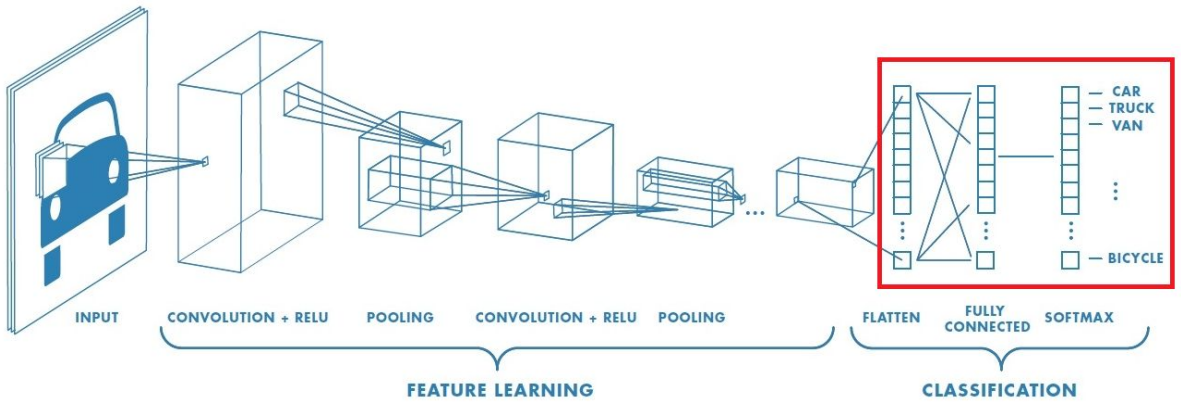
Do rozpoznawania twarzy zbudowaliśmy własną konwolucyjną sieć neuronową o architekturze jak poniżej:

```python
import numpy as np
#import seaborn as sns
from keras.preprocessing.image import load_img, img_to_array
import matplotlib.pyplot as plt
import os
import cv2

# size of the image: 48*48 pixels
pic_size = 100

# input path for the images
base_path = "C:/Users/Wojciech/Desktop/images4/"

plt.figure(0, figsize=(12, 20))
cpt = 0

for people in os.listdir(base_path + "train/"):
    for i in range(1, 6):
        cpt = cpt + 1
        plt.subplot(7, 5, cpt)
        img = load_img(base_path + "train/" + people + "/" +os.listdir(base_path + "train/" +
                                                                       people)[i], target_size=(pic_size, pic_size))
        plt.imshow(img, cmap="gray")

plt.tight_layout()
plt.show()
for expression in os.listdir(base_path + "train"):
    print(str(len(os.listdir(base_path + "train/" + expression))) + " " + expression + " images")

from keras.preprocessing.image import ImageDataGenerator

# number of images to feed into the NN for every batch
batch_size = 8

datagen_train = ImageDataGenerator()
datagen_validation = ImageDataGenerator()

train_generator = datagen_train.flow_from_directory(base_path + "train",
                                                    target_size=(pic_size, pic_size),
                                                    color_mode="grayscale",
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=True)

validation_generator = datagen_validation.flow_from_directory(base_path + "validation",
                                                              target_size=(pic_size, pic_size),
                                                              color_mode="grayscale",
                                                              batch_size=batch_size,
                                                              class_mode='categorical',
                                                              shuffle=False)




#Określanie modelu sieci
from keras.layers import Dense, Input, Dropout, GlobalAveragePooling2D, Flatten, Conv2D, \
                         BatchNormalization, Activation, MaxPooling2D
from keras.models import Model, Sequential
from keras.optimizers import Adam


# number of possible label values
nb_classes = 3



# Initialising the CNN
model = Sequential()

# 1 - Convolution
model.add(Conv2D(64, (3, 3), padding='same', input_shape=(100, 100, 1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 2nd Convolution layer
model.add(Conv2D(128, (5, 5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 3rd Convolution layer
model.add(Conv2D(128, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

```

```
# 4rd Convolution layer
model.add(Conv2D(256, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 5th Convolution layer
model.add(Conv2D(256, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# 6th Convolution layer
model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 7th Convolution layer
model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 8th Convolution layer
model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))


# Flattening
model.add(Flatten())

# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(nb_classes, activation='softmax'))

opt = Adam(lr=0.0001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])



#Trenowanie modelu
# number of epochs to train the NN
epochs = 20

from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]

history = model.fit_generator(generator=train_generator,
                              steps_per_epoch=train_generator.n//train_generator.batch_size,
                              epochs=epochs,
                              validation_data = validation_generator,
                              validation_steps = validation_generator.n//validation_generator.batch_size,
                              callbacks=callbacks_list
                              )


model.load_weights('model_weights.h5')

model.save('model_20x350photos.model')
```

Sieć ta na wejście dostaje zdjęcie wcześniej wyszukanej i wyciętej z obrazu twarzy. Analizuje je i stara się znaleźć podobieństwa do tego czego nauczyła się w fazie uczenia.
Nasza sieć składa się z ośmiu warstw konwolucyjnych. Pomiędzy nimi znajdują się 3 warstwy pooling. Następnie dochodzi do spłaszczenia. Ostatnie 3 warstwy mają za zadanie określenie do której z klas należy dany obiekt na podstawie wcześniej wydobytych cech. Warstwa wyjściowa zawiera 3 neurony - bo mamy trzy dostępne klasy. Funkcja aktywacji softmax w ostatniej warstwie sprawia, że na wyjściu otrzymujemy wektor składający się z trzech cyfr, określających podobieństwo do każdej z 3 klas. Program wybiera najwyższe prawdopodobieństwo i wyświetla na obrazie nazwę klasy - w tym przypadku imię rozpoznanej osoby. Jeśli żadne z prawdopodobieństwo nie przekroczy bariery 80% na ekranie wyświetli się napis **UNKNOWN.**
Fragment kodu odpowiedzialny za predykcje oraz jej prezentację:

```
    prediction = model.predict(prepare(cropped_photo))

    pewnosc = np.amax(prediction)
    pewnosc2 = pewnosc*100
    znak = '%'

    if pewnosc < 0.8:
        text = 'Unknown'
        text2 ='_'
    elif pewnosc > 0.8:
        text = CATEGORIES[np.argmax(prediction)] + round(pewnosc2, 2).__str__()+znak


    print(prediction)
    y = startY - 10 if startY - 10 > 10 else startY + 10
    cv2.rectangle(frame, (startX, startY), (endX, endY),
                  (0, 0, 255), 2)
    cv2.putText(frame, text, (startX, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 0, 255), 2)
```



Struktura naszej sieci przedstawia się w uproszczeniu podobnie jak na grafice powyżej. Warstwy konwolucyjne, RELU oraz pooling służą do wydobycia złożonych wzorców i cech ze zdjęć na wejściu. Do ostatecznej klasyfikacji obiektu stosuje się tzw. warstwę gęstą (ang. dense layer), która ostatecznie identyfikuje do jakiej klasy należy obiekt wejściowy. Dokonuje tego na zasadzie prawdopodobieństwa czy obiekt ze zdjęcia pasuje do określonego rodzaju (klasy).

Wyniki uczenia sieci obserwowaliśmy na wykresach tworzonych w następujący sposób:

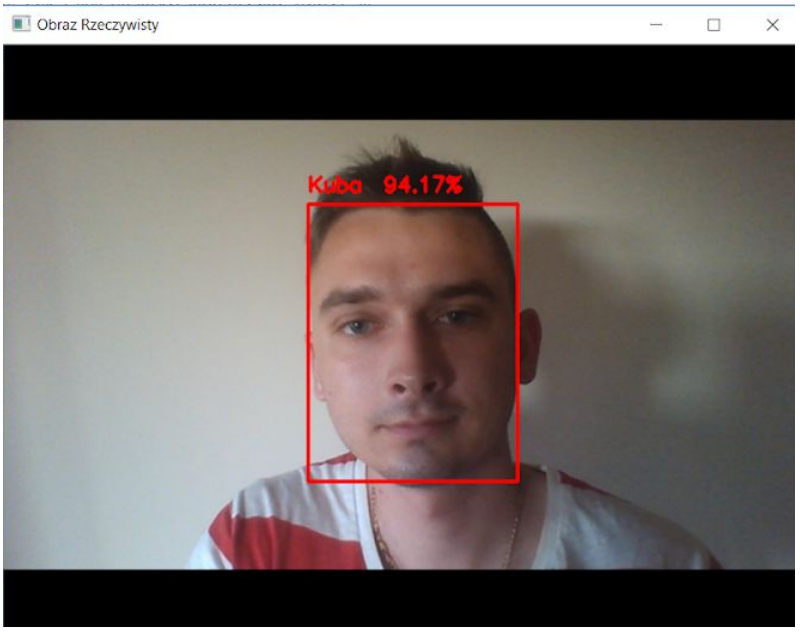
```
168
169
170  import matplotlib.pyplot as plt
171
172  plt.figure(figsize=(20,10))
173  plt.subplot(1, 2, 1)
174  plt.suptitle('Optimizer : Adam', fontsize=10)
175  plt.ylabel('Loss', fontsize=16)
176  plt.plot(history.history['loss'], label='Training Loss')
177  plt.plot(history.history['val_loss'], label='Validation Loss')
178  plt.legend(loc='upper right')
179
180  plt.subplot(1, 2, 2)
181  plt.ylabel('Accuracy', fontsize=16)
182  plt.plot(history.history['acc'], label='Training Accuracy')
183  plt.plot(history.history['val_acc'], label='Validation Accuracy')
184  plt.legend(loc='lower right')
185  plt.show()
186
187  # compute predictions
188  predictions = model.predict_generator(generator=validation_generator)
189  y_pred = [np.argmax(probas) for probas in predictions]
190  y_test = validation_generator.classes
191  class_names = validation_generator.class_indices.keys()
192
```

Zdjęcie 1. Kod programu do otrzymania wykresu 1-go.

```python
from sklearn.metrics import confusion_matrix
import itertools


def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm.Blues):
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.figure(figsize=(10, 10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


# compute confusion matrix
cnf_matrix = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)

# plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, title='Normalized confusion matrix')
plt.show()
```

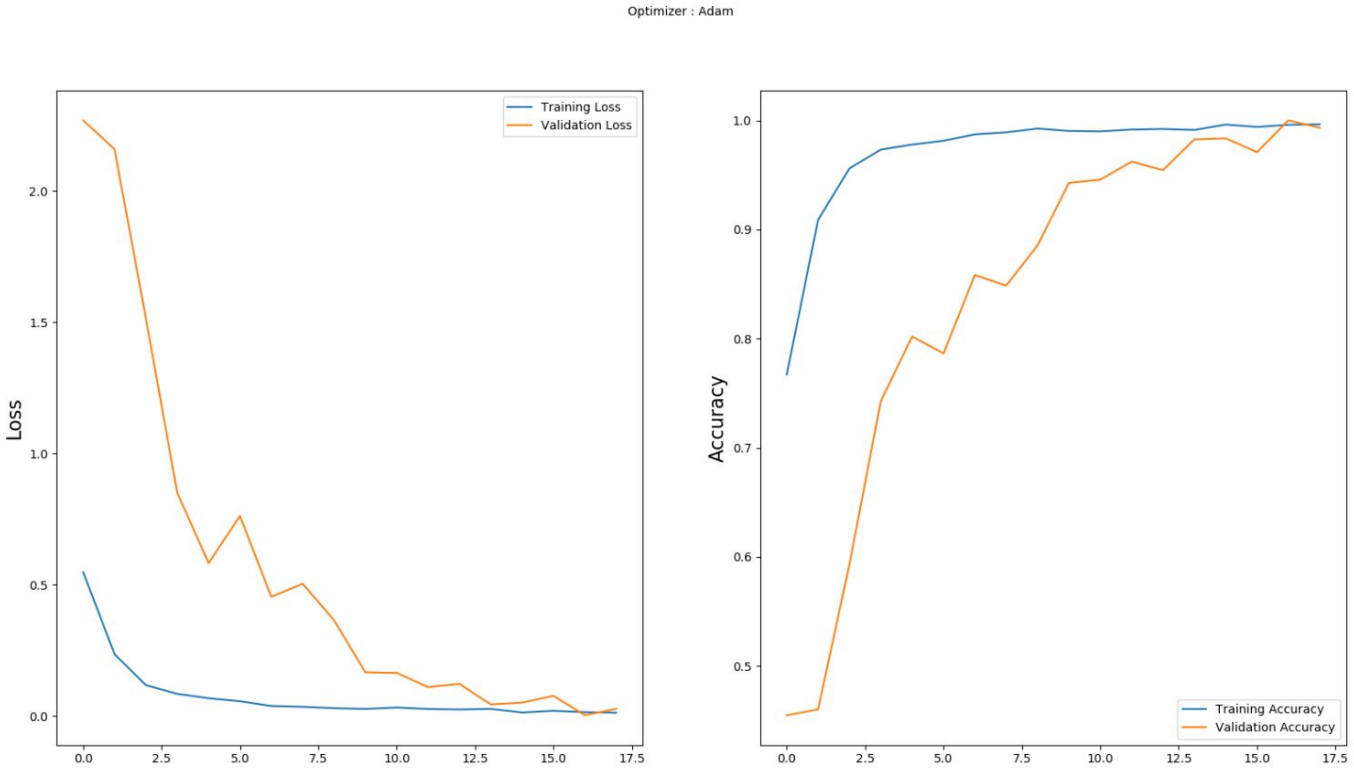
Zdjęcie 2. Kod do otrzymania wykresu 2-go.

Efekt końcowy:

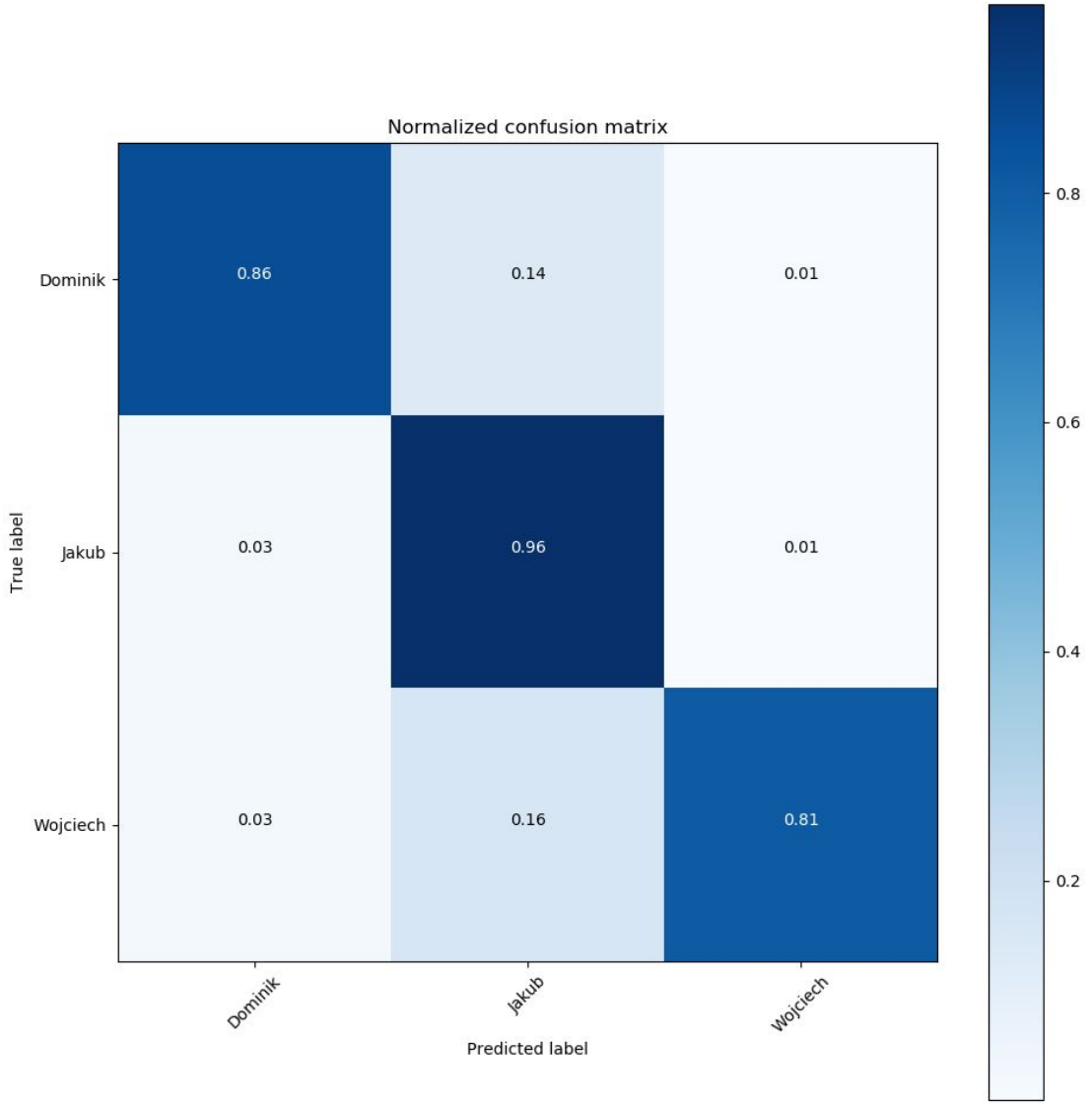


7. Wnioski

Zbudowana przez nas sieć bardzo dobrze radzi sobie z rozpoznawaniem osób, których zdjęcia znajdują się w bazie danych. Nie ma również praktycznie żadnego znaczenia czy osoba rozpoznawana znajduje się w ciemnym czy jasnym pomieszczeniu. Poniżej zamieszczamy wykresy przedstawiające postęp uczenia (wykres 1), a także wykres pokazujący w jaki sposób nasz model klasyfikuje poszczególne osoby (wykres 2).



Wykres 1. Postęp uczenia.



Wykres 2. Sposób klasyfikacji modelu.

Na obecną chwilę występuje problem w momencie próby rozpoznania osoby spoza bazy. Program przypisuje bardzo duży współczynnik prawdopodobieństwa do którejś z klas, nawet jeśli rozpoznawany obraz do niej nie należy. Dzieję się tak prawdopodobnie ze względu na zbyt małą liczbę warstw konwolucyjncych lub nie do końca odpowiednio dobrane ich parametry. Zostaje wyciągnięte zbyt mało cech, w szczególności tych znaczących jak zarost, odległość pomiędzy oczami, usta itp. Wielokrotne próby poprawienia tego problemu (zmiana parametrów sieci, liczby warstw) niestety nie dawały zamierzonego efektu. Jeżeli udałoby się to naprawić na pewno należałoby również spróbować zmniejszyć barierę procentową wspomnianą wcześniej, która jest stosunkowo wysoka (gdyż dwie losowe osoby, biorąc pod uwagę ich charakterystyczne elementy twarzy, nie powinny być do siebie podobne w tak wysokim stopniu).

Dodatkowo zastosowanie w ostatniej warstwie funkcji **softmax** oznacza, że sieć na wyjściu wyrzuca prawdopodobieństwo przynależności do danej klasy, a suma tych prawdopodobieństw wynosi **1**. Gdy rozpoznaniu twarzy podlega twarz spoza bazy program i tak stara się ją przyporządkować do którejś z klas. Biorąc pod uwagę, że w bazie danych uwzględnione są trzy osoby to zawsze przynajmniej jedna klasa osiągnie min. 33% prawdopodobieństwa przynależności co już jest dość dużym wynikiem.