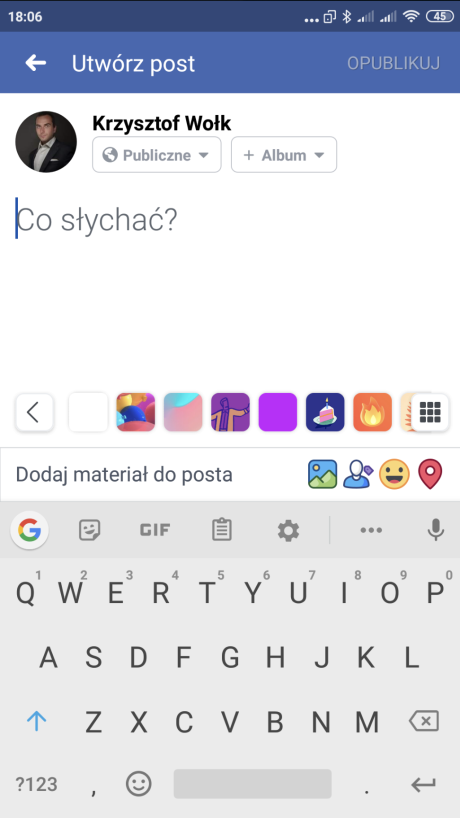
### Rozpoznawanie twarzy przy użyciu głębokiego uczenia maszynowego.

Na pewno podczas używania swojego Facebooka zauważyłeś jak szybko firma ta rozwija swoje możliwości dotyczące rozpoznawania Twoich przyjaciół na zdjęciach. Na początku to Ty oznaczałeś swoich znajomych poprzez klikanie na twarzach i wpisywanie ich imion i nazwisk. Natomiast teraz Facebook w niemal sposób magiczny jest w stanie zrobić to za Ciebie.



Facebook w automatyczny sposób potrafi oznaczyć ludzi na Twoich zdjęciach, których wcześniej zaznaczyłeś. Być może przeraża Cię to? Taka technologia nazywana jest po prostu rozpoznawaniem rysów twarzy. Facebook posługuje się takim algorytmem, który potrafi dokonać klasyfikacji naszych znajomych nawet po kilkukrotnym ich oznaczeniu. W chwili obecnej ich algorytm cechuje się precyzją na poziomie 98%, czyli jest to poziom podobny do tego, jakim cechuje się w tym zakresie człowiek.

Zatem sprawdźmy jak współcześnie działa rozpoznawanie rysów twarzy. Rozpoznawanie rysów Twoich znajomych jest zbyt łatwym zadaniem. Postawmy więc większe wyzwanie tej technologii i spróbujmy rozróżnić dwie panie: Dorotę Gardias (Polska prezenterka telewizyjna) i Morenę Baccarin (aktorka)



# Posługiwanie się Uczeniem Maszynowym w rozwiązaniu bardzo skomplikowanego problemu

Do tej pory do rozwiązania różnych problemów używaliśmy uczenia maszynowego, gdzie posługiwaliśmy się algorytmem, który uczył maszynę po dostarczeniu danych wejściowych, a następnie otrzymywaliśmy odpowiedni wynik. W taki sposób szacowaliśmy na przykład cenę pojazdu czy rozpoznawaliśmy obiekt na obrazie (ptak).

Rozpoznawanie rysów twarzy to jednak całkiem inne zagadnienie, które złożone jest z wielu powiązanych ze sobą problemów:

1. Patrzenie na pojedyncze zdjęcie związane jest z umiejętnością rozpoznawania twarzy jako osobnego obiektu.
2. Na wygląd rysów twarzy ma wpływ wiele czynników jak na przykład mimika twarzy, oświetlenie czy pozycja głowy, a niezależnie od tych czynników jest to nadal twarz tej samej osoby.
3. Ważna jest umiejętność rozróżniania cech, które są unikatowe dla danej twarzy jak na przykład wielkość oczu, podłużność twarzy, wielkość nosa czy ust.
4. Umiejętność porównywania tych cech szczególnych dla danej twarzy do innych ludzi, po to, aby być w stanie przypisać tej osobie imię i nazwisko.

Mózg człowieka jest dostosowany do dokonywania klasyfikacji i rozróżnień w sposób automatyczny i natychmiastowy. Poza tym ludzie są tak przyzwyczajeni do rozpoznawania twarzy, że rozpoznają ją na różnych obiektach, kompozycjach itp.

Komputery zaś przynajmniej na chwilę obecną nie są w stanie dokonywać tak zagmatwanych uogólnień, czyli muszą zostać nauczone owej umiejętności niemal krok po kroku, rozkładając problem na mniejsze etapy/zadania. Zatem konieczne będzie stworzenie platformy, dzięki której komputer będzie rozpoznawał rysy twarzy posiłkując się przechodzeniem przez poszczególne etapy, które są następujące:

* Etap 1: Znalezienie twarzy na obrazie jako obiektu
* Etap 2: Analizowanie rysów twarzy
* Etap 3: Porównania do innych znanych twarzy
* Etap 4: Dokonanie przewidywania prawdopodobieństwa

W artykule jednak nie będziemy się zagłębiali w każdy poszczególny algorytm, gdyż jest ich tyle, że wyszłaby piękna książka z ich opisu, a nie mamy tutaj na to miejsca. Przedstawiony zostanie natomiast ogólny zamysł, który jest podstawą działania owych algorytmów. Będziemy także w stanie stworzyć własny system służący do rozpoznawania rysów twarzy w języku Python oraz przy użyciu OpenFace i dlib.

*Etap 1: Znalezienie twarzy jako obiektu*

Pierwszym krokiem w procesie rozpoznawania twarzy będzie rozpoznanie samego zarysu twarzy wśród innych obiektów na zdjęciu. Rozpoznawanie rysów jest bardzo pomocną funkcją wielu aparatów fotograficznych. Doskonale zapewne wiesz/widziałeś, w jaki sposób ona działa.

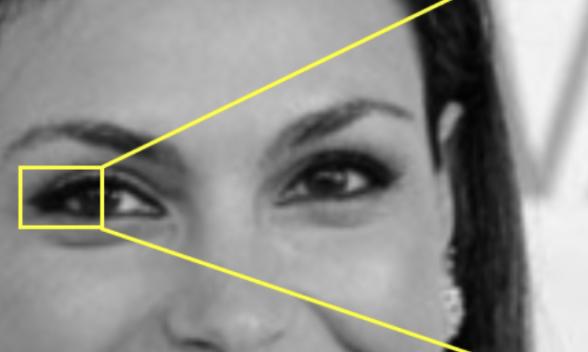


Rozpoznawanie rysów twarzy stało się dla wszystkich dostępne, gdy po roku dwutysięcznym Paul Viola i Michael Jones wynaleźli sposób na rozpoznawanie twarzy, za co też otrzymali nagrodę Marr Prize na konferencji International Conference on Computer Vision w 2003 roku. Ich sposób był na tyle szybki, że można go było zastosować w aparatach fotograficznych. Obecnie mamy jeszcze doskonalsze rozwiązania, w tym artykule posłużymy się metodą wynalezioną w 2005 roku, noszącą nazwę Histogram of Oriented Gradients (HOG).

Aparat cyfrowy w sposób automatyczny rozpoznaje zarysy twarzy, ustawiając ostrość na twarz przed zrobieniem zdjęcia. Naszym zadaniem będzie znalezienie takich obszarów na zdjęciu, które będziemy chcieli przenieść do kolejnego etapu w procesie uczenia. Zatem aby rozpoznawać twarze na zdjęciach rozpoczniemy od zmiany zdjęcia, które jest kolorowe w czarno białe, gdyż do detekcji rysów twarzy kolor nie jest potrzebny.



W kolejnym kroku przyjrzymy się każdemu pojedyńczemu pikselowi, jeden po drugim, a także pikselom w otoczeniu danego piksela. Mamy znaleźć jak ciemny jest badany pojedyńczy piksel w porównaniu z tym, który z nim sąsiaduje. W taki sposób możemy sprawdzić kierunek zaciemniania obrazu (w tym przypasku w stronę prawego górnego rogu).



**Analizując jeden pixel i pixele w jego bezpośrednim otoczeniu widzimy, że zdjęcie zaciemnia się w stronę prawego górnego rogu.**

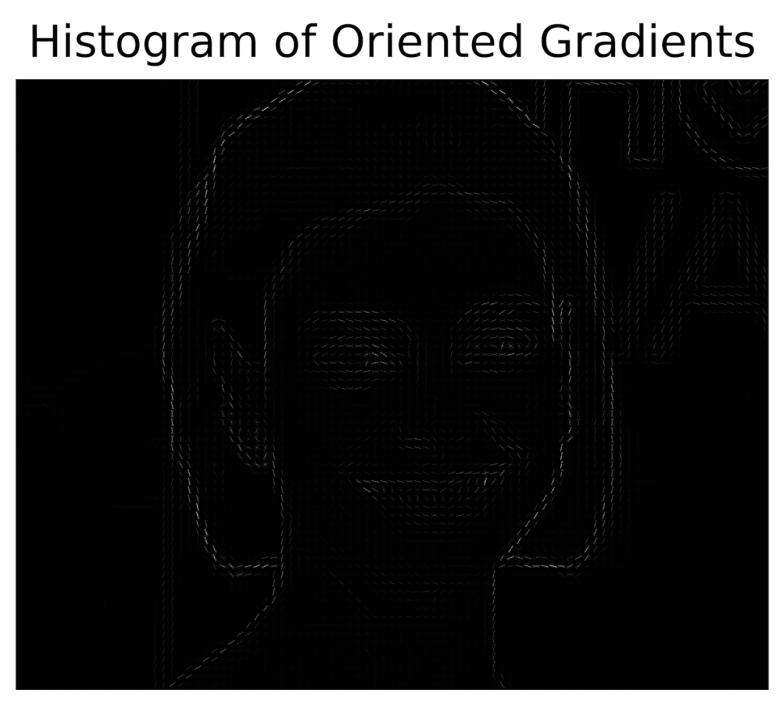
Jeśli etap ten zostanie powtórzony dla każdego piksela na zdjęciu to każdy piksel danych światła zostanie zastąpiony strzałką. Strzałki określane są jako gradient i wskazują na przepływ światła przez cały obraz, począwszy od jaśniejszej strony zdjęcia.

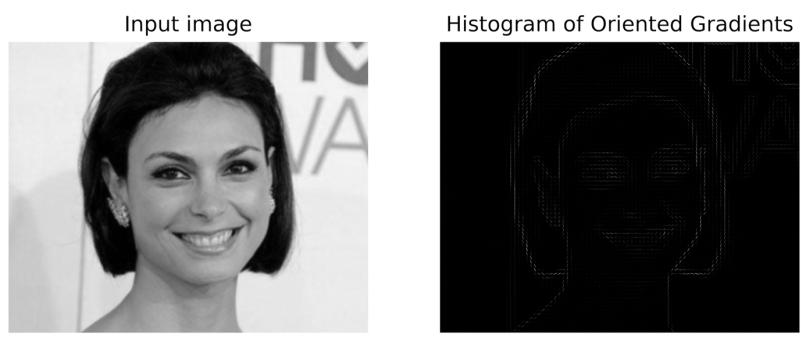
Zastępowanie pikseli gradientem nie jest przypadkowe, ma poważny powód, mianowicie piksele analizowane bezpośrednio dla zdjęć o większej ekspozycji świetlnej i mniejszemu naświetleniu dostarczają nam różne wartości dla zdjęcia tej samej twarzy. Jeśli będą więc analizowane zmiany w gradiencie – obydwa zdjęcia, niezależnie czy zdjęcia jest jasne czy ciemne to otrzymamy to samo odwzorowanie. Więc takie podejście pomaga w rozwiązaniu problemu.

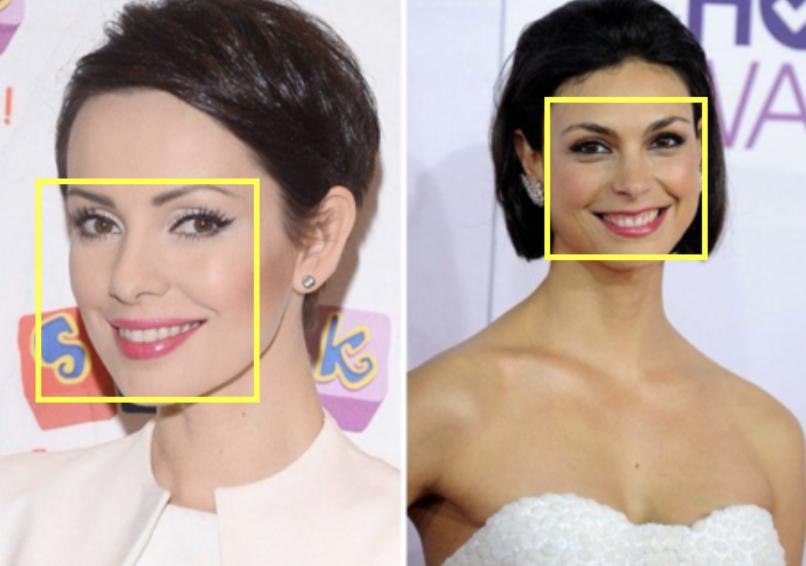
Zachowując wartości gradientu dla każdego pojedynczego piksela gromadzimy ogromną ilość danych. W naszym zadaniu o wiele lepszym rozwiązaniem było by zarejestrowanie podstawowego przepływu światła oraz przejść od jasnego do ciemnego na obrazie, po to, aby uzyskać podstawowy zarys zdjęcia. W celu osiągnięcia tego obraz zostanie podzielony na mniejsze części o wymiarach 16x16px, a następnie zostanie obliczone które punkty gradientu znajdujące się w poszczególnym kwadracie wskazują w różnych kierunkach (ile wskazuje w danym kierunku np. w górę). Po takich obliczeniach piksele zostają zastąpione strzałkami, które dominują wskazanie w danym kierunku.



Aby znaleźć twarz w takim przekształconym obrazie musi zostać wyszukana część obrazu, która jest jak najbardziej podobna do wzorca HOG, otrzymanego z ekstrakcji danych początkowych uczących ten algorytm rozpoznawania rysów twarzy. Posługując się zatem rozwiązaniem przedstawionym w niniejszym artykule jesteśmy w stanie z dużą łatwością wykryć twarze na niemal każdym zdjęciu.







Przy użyciu języka Python i dlib możesz w następujący sposób tworzyć i przeglądać odwzorowanie obrazu w HOG.

import dlib

import sys

from skimage import io

#pobieramy obrazek z linii poleceń

file\_name = sys.argv[1]

#tworzenie detektora hog z wbudowanej klasy w bibliotece dlib

face\_detector = dlib.get\_frontal\_face\_detector()

win = dlib.image\_window()

#konwertujemy obrazek do postaci macierzowej

image = io.imread(file\_name)

#uruchomienie algorytmu hog

detected\_faces = face+detector(image, 1)

print("znalazłem {} twarzy w pliku {}".format(len(detected\_faces), file\_name))

#otwarcie okienka z podglądem rysunku

win.set\_image(image)

#pętla analizująca wszystkie twarze wykryte na obrazku

for i, face\_rect in enumerate(detected\_faces):

#wykryte twarze są zwracane jako zestaw koordynatów

print("- twarze #{} znalezione po lewej: {} na górze: {} po prawej: {} na dole: {}".format(i, face\_rect.left(), face\_rect.top(), face\_rect.right(), face\_rect.bottom()))

#obrysuj każdą twarz prostokątem

win.add\_overlay(face\_rect)

#aby zakończyć wciśnij <enter>

dlib.hit\_enter\_to\_continue()

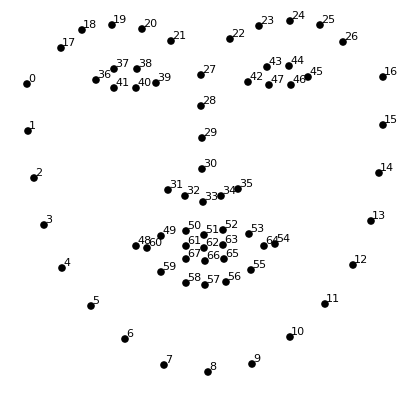
*Etap 2: Pozowanie i uchwycenie twarzy.*

Pierwszy krok, jak widać był dość pracochłonny (uchwycenie twarzy na zdjęciu). Teraz zaś musimy rozwiązać nieco inną kwestię, mianowicie dla komputera ta sama ludzka twarz w momencie wystąpienia w innej pozie czy odwrócona, będzie twarzą zupełnie innej osoby.

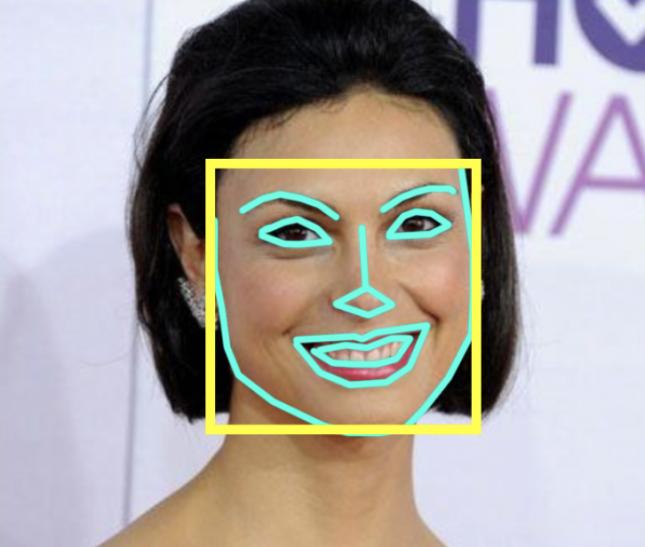
 

Człowiek dość szybko rozpozna, że na obydwu zdjęciach jest ta sama osoba, komputer natomiast nie jest w stanie tego dokonać. Aby się pozbyć tego problemu musimy więc tak zmodyfikować zdjęcie, aby oczy i usta zawsze były na zdjęciach w tym samym miejscu, co bardzo ułatwi rozpoznawanie rysów twarzy w kolejnych etapach.

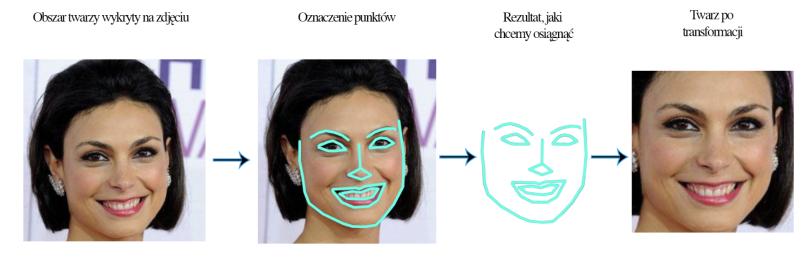
Tutaj bardzo przydatny okaże się algorytm Face Landmark Estimation, który służy do określania punktowego rysów twarzy. Z wielu rozwiązań wybrałem to stworzone w 2014 roku przez Josephine Sullivan oraz Vahida Kazemi. Koncepcją tego rozwiązania jest znalezienie sześćdziesięciu ośmiu centralnych punktów na każdej twarzy tzw. landmark. Będą nimi na przykład pozycja brody, kąciki oczy, łuki brwiowe itp. Następnie trenowany jest algorytm uczenia maszynowego, aby odnalazł on oznaczone przez nas poszczególne punkty każdej twarzy. Poniższy obraz został stworzony przez Brandona Amosa z CMU, który pracuje z użyciem OpenFace.



Przykładowy wynik oznaczenia punktów na zdjęciu testowym:



W momencie, gdy już wiemy, gdzie umiejscowione są usta i oczy, zdjęcie jest obracane, dopasowywany jest rozmiar, a także nakierowywane tak aby pozycja oczu i ust była jak najbardziej na środku. Oczywiście nie wykonujemy żadnych przekształceń w 3d, gdyż wprowadziłoby to spore zniekształcenie obrazu. Zatem używamy tylko prostych przekształceń jak: obrót, zmiana wielkości.



Teraz nie ma już znaczenia to, jak odwrócona jest twarz, gdyż jesteśmy w stanie wyśrodkować usta i oczy tak, aby były w podobnej pozycji co na naszym zdjęciu. Pozwoli nam to na przejście do kolejnego etapu i otrzymanie bardziej precyzyjnego wyniku.

Poniżej kod napisany w języku Python oraz przy użyciu dlib, pozwala na wykonanie tego etapu samodzielnie, czyli oznaczenie punktów centralnych na twarzy i wykrycie rysów.

import sys

import dlib

from skimage import io

# Można pobrać wymagany model detekcji tutaj:

# http://dlib.net/files/shape\_predictor\_68\_face\_landmarks.dat.bz2

predictor\_model = "shape\_predictor\_68\_face\_landmarks.dat"

# Pobranie nazwy pliku obrazu z wiersza poleceń

file\_name = sys.argv[1]

# Stworzenie wykrywania twarzy HOG przy użyciu wbudowanej klasy dlib.

face\_detector = dlib.get\_frontal\_face\_detector()

face\_pose\_predictor = dlib.shape\_predictor(predictor\_model)

win = dlib.image\_window()

# Pobranie nazwy pliku obrazu z wiersza poleceń

file\_name = sys.argv[1]

# Załadowanie obrazu

image = io.imread(file\_name)

# Włączenie wykrywania twarzy HOG na obrazie

detected\_faces = face\_detector(image, 1)

print("Found {} faces in the image file {}".format(len(detected\_faces), file\_name))

# Pokazanie okna z obrazem

win.set\_image(image)

# Pętla przez każdą twarz, którą znaleźliśmy na obrazku.

for i, face\_rect in enumerate(detected\_faces):

# Wykryte twarze są zwracane jako obiekt o współrzędnych

# górnych, lewych, prawych i dolnych krawędzi

print("- Face #{} found at Left: {} Top: {} Right: {} Bottom: {}".format(i,

face\_rect.left(), face\_rect.top(), face\_rect.right(), face\_rect.bottom()))

# Narysowanie kwadratu wokół każdej znalezionej twarzy. win.add\_overlay(face\_rect)

# Przewidywanie pozycji twarzy

pose\_landmarks = face\_pose\_predictor(image, face\_rect)

# Narysowanie punktów twarzy na ekranie

win.add\_overlay(pose\_landmarks)

dlib.hit\_enter\_to\_continue()

i przetworzenie zdjęcia:

import sys

import dlib

import cv2

import openface

# http://dlib.net/files/shape\_predictor\_68\_face\_landmarks.dat.bz2

predictor\_model = "shape\_predictor\_68\_face\_landmarks.dat"

# Pobranie nazwy pliku obrazu z wiersza poleceń

file\_name = sys.argv[1]

# Stworzenie HOG przy użyciu dlib

face\_detector = dlib.get\_frontal\_face\_detector()

face\_pose\_predictor = dlib.shape\_predictor(predictor\_model)

face\_aligner = openface.AlignDlib(predictor\_model)

# Pobranie nazwy pliku obrazu z wiersza poleceń

file\_name = sys.argv[1]

# Załadowanie pliku

image = cv2.imread(file\_name)

# Włączenie HOG na naszym zdjęciu

detected\_faces = face\_detector(image, 1)

print("Found {} faces in the image file {}".format(len(detected\_faces), file\_name))

# Pętla po wszystkich twarzach znalezionych na zdjęciu

for i, face\_rect in enumerate(detected\_faces):

# Wykryte twarze zwracane są jako współrzędne

# górnych, lewych, prawych i dolnych krawędzi

print("- Face #{} found at Left: {} Top: {} Right: {} Bottom: {}".format(i, face\_rect.left(), face\_rect.top(), face\_rect.right(), face\_rect.bottom()))

# Otrzymanie pozycji twarzy

pose\_landmarks = face\_pose\_predictor(image, face\_rect)

# Użycie openface do obliczenia i wykonania wyrównania twarzy

alignedFace = face\_aligner.align(534, image, face\_rect, landmarkIndices=openface.AlignDlib.OUTER\_EYES\_AND\_NOSE)

# Zapisanie przetworzonego zdjęcia do pliku

cv2.imwrite("aligned\_face\_{}.jpg".format(i), alignedFace)

*Etap 3: Kodowanie twarzy*

W kolejnym etapie przechodzimy do sedna zadania, mianowicie rozróżniania twarzy. To bardzo interesujący etap. Najprostszym sposobem wydaje się być porównanie nieznanej twarzy, znalezionej w drugim etapie ze wszystkim zdjęciami ludzi już oznaczonych. Gdy znajdziemy zaznaczoną twarz przypominającą tą nieznaną możemy przyjąć, iż jest to ta sama osoba, czyż nie? Niestety takie podejście jest dość problematyczne. Zwróćmy uwagę na przykład na Facebooka, który ma miliardy użytkowników i tryliony załadowanych zdjęć, nie ma czasu na porównywanie każdego nowo wgranego zdjęcia. To zajęłoby zbyt dużo czasu, a strona musi przetwarzać dane w ciągu milisekund, a nie godzin. Dlatego też, aby tego dokonać musimy posiadać określone pomiary z każdej twarzy, dzięki czemu będziemy w stanie zmierzyć nieznaną twarz i odnaleźć tą najbardziej do niej podobną.

# Najbardziej wiarygodne sposoby otrzymania wymiarów twarzy.

Zacznijmy od tego, które wymiary są niezbędne z każdej twarzy, abyśmy mogli stworzyć własną bazę danych? kolor oczu? Wielkość nosa? Uszu? Czy jeszcze coś innego? Jak się okazuje, kolor oczu czy inne tego typu miary, oczywiste dla człowieka, dla komputera nie mają najmniejszego znaczenia, gdyż poszukuje on na zdjęciu poszczególnych pikseli. Z odkrycia badaczy wynika, iż zebranie odpowiednich pomiarów powinno być zostawione komputerowi. Głębokie uczenie jest w stanie pokonać ludzkie zdolności w tej materii, decydując które wymiary są istotne.

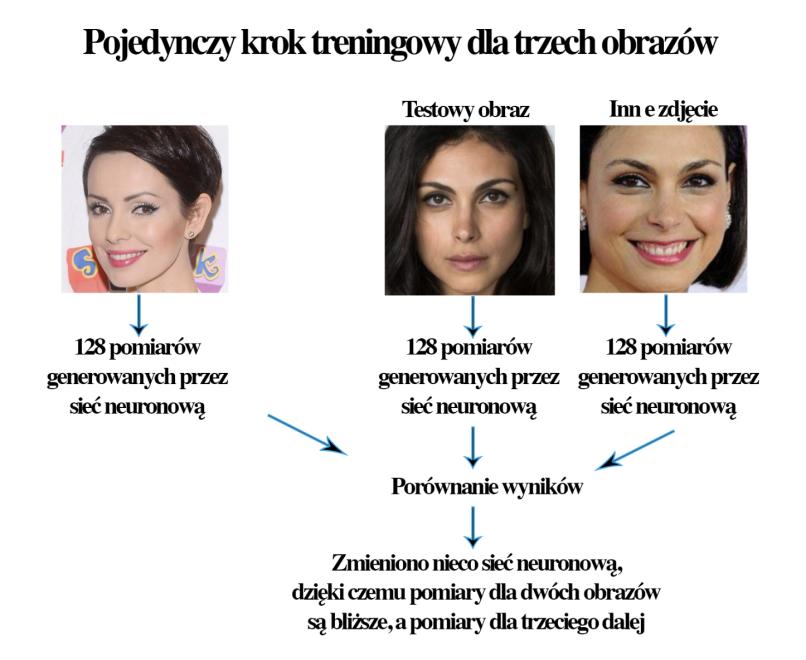
Rozwiązaniem jest trenowanie głębokiej konwolucyjnej sieci neuronowej. W tym przypadku jednak zamiast rozpoznawania obiektów na zdjęciu, algorytm jest uczony generowania 128 wymiarów dla każdej twarzy.

*Krok trenujący: Równoczesna analiza “trojaczków”*

Proces ten działa na zasadzie równoczesnej obserwacji trzech twarzy:

1. Załadowanie zdjęcia przykładowego znanej osoby
2. Załadowanie innego zdjęcia znanej osoby
3. Załadowanie zdjęcia zupełnie innej osoby

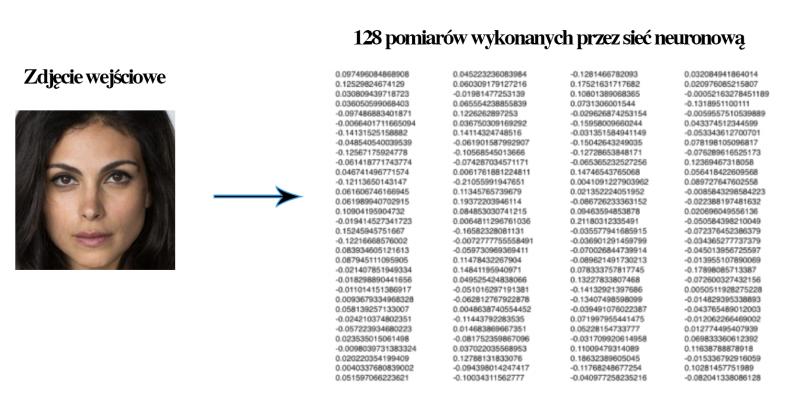
Algorytm sprawdza i generuje odpowiednie wymiary dla każdego zdjęcia, a następnie dopasowuje sieć neuronową tak, aby upewnić się, że wymiary, które są generowane dla zdjęcia pierwszego oraz drugiego będą do siebie zbliżone, a dla drugiego i trzeciego będą bardziej od siebie oddalone.



Jeśli krok będzie powtarzany dla milionów zdjęć wielu tysięcy różnych osób miliony razy to nasza sieć neuronowa będzie uczyła się generowania tych 128 wymiarów dla twarzy w sposób prawidłowy. Kilka różnych zdjęć, przedstawiających tą samą osobę dadzą w przybliżeniu podobne wymiary. Te 128 wymiarów nazywa się mianem osadzania (embedding). Takie podejście związane z wykrywaniem rysów twarzy, przedstawione w niniejszym artykule zostało stworzone w 2015 roku przez firmę Google.

Taki proces, w którym trenowana jest konwolucyjna sieć neuronowa w celu uzyskania danych wyjściowych przedstawionych w postaci zaznaczeń twarzy (embeddings) potrzebuje oprócz dużej ilości danych także dużej mocy, niezbędnej, aby te dane przetworzyć te dane (np. karta nVidia Tesla, aby otrzymać precyzyjne dane porzebuje dwudziestu czterech godzin treningu). Wynikiem takiego uczenia jest możliwość detekcji cech niemal każdej twarzy, nawet takiej, którą zobaczy pierwszy raz. Możemy się nazwać szczęściarzami, gdyż ludzie z OpenFace dokonali tego dla nas, gdyż opublikowali kilka już wyuczonych sieci, których każdy może użyć.

Zostało nam do wykonania tak naprawdę tylko jedno zadanie, mianowicie załadowanie zdjęć twarzy do tych już wyuczonych sieci, po to, aby uzyskać pomiary dla każdego z nich. Poniżej wymiary, które zostały otrzymane dla zdjęcia testowego.



Nie interesuje nas to, które części twarzy ze zdjęcia zostaną wybrane przez komputer, gdyż on sam wybiera te 128 pomiarów, a sposób selekcji nas nie obchodzi. Zależy nam na tym, aby pomiary, które zostały wygenerowane przez sieć były niemal identyczne. Za pomocą poniższego kodu możesz samodzielnie dokonać powyższego, pozwala nam na to OpenFace, który dostarcza skrypt, który generuje oznaczenia dla zdjęć w folderze, a następnie zapisuje je jako plik w formacie .CSV.

local ffi = require 'ffi'

local batchNumber, nImgs = 0

torch.setdefaulttensortype('torch.FloatTensor')

function batchRepresent()

local loadSize = {3, opt.imgDim, opt.imgDim}

print(opt.data)

local cacheFile = paths.concat(opt.data, 'cache.t7')

print('cache lotation: ', cacheFile)

local dumpLoader

if paths.filep(cacheFile) then

print('Loading metadata from cache.')

print('If your dataset has changed, delete the cache file.')

dumpLoader = torch.load(cacheFile)

else

print('Creating metadata for cache.')

dumpLoader = dataLoader{

paths = {opt.data},

loadSize = loadSize,

sampleSize = loadSize,

split = 0,

verbose = true

}

torch.save(cacheFile, dumpLoader)

end

collectgarbage()

nImgs = dumpLoader:sizeTest()

print('nImgs: ', nImgs)

assert(nImgs > 0, "Failed to get nImgs")

batchNumber = 0

for i=1,math.ceil(nImgs/opt.batchSize) do

local indexStart = (i-1) \* opt.batchSize + 1

local indexEnd = math.min(nImgs, indexStart + opt.batchSize - 1)

local batchSz = indexEnd-indexStart+1

local inputs, labels = dumpLoader:get(indexStart, indexEnd)

local paths = {}

for j=indexStart,indexEnd do

table.insert(paths,

ffi.string(dumpLoader.imagePath[dumpLoader.testIndices[j]]:data()))

end

repBatch(paths, inputs, labels, batchSz)

if i % 5 == 0 then

collectgarbage()

end

end

if opt.cuda then

cutorch.synchronize()

end

end

function repBatch(paths, inputs, labels, batchSz)

batchNumber = batchNumber + batchSz

if opt.cuda then

inputs = inputs:cuda()

end

local embeddings = model:forward(inputs):float()

if opt.cuda then

cutorch.synchronize()

end

if batchSz == 1 then

embeddings = embeddings:reshape(1, embeddings:size(1))

end

for i=1,batchSz do

labelsCSV:write({labels[i], paths[i]})

repsCSV:write(embeddings[i]:totable())

end

print(('Represent: %d/%d'):format(batchNumber, nImgs))

end

*Etap 4: Dopasowanie imienia do obrazu twarzy*

Etap czwarty, jest etapem ostatnim i tak naprawdę najłatwiejszym w całym procesie. Zadaniem tutaj jest odnalezienie w bazie zdjęć znanych osób, które mają≥ najbardziej zbliżone wymiary do zdjęcia testowego. Możemy tutaj użyć podstawowego algorytmu klasyfikatora np. przy użyciu SVM. Pozostaje nam wytrenowanie klasyfikatora, który pobierze pomiary ze zdjęcia testowego i określi, które osoby są najbardziej podobne. Takie zastosowanie klasyfikatora jest bardzo szybkie i zajmuje milisekundy, ale rezultatem jego użycia będzie uzyskanie imienia danej osoby.

*VOILA!* W ten oto sposób poznaliśmy główne kroki tego, w jakli sposób może zostać rozpoznana ludzka twarz. Oczywiście nie jest to jedyna metoda. Istnieją rozwiązania znacznie prostsze, które jednak ze względu na brak dokładności zostały wyparte, jak i rozwiązania bardziej zaawansowane, wykorzystywane chociażby w zabezpieczeniach genetycznych, gdzie problem jest znacznie bardziej zaawansowany, ponieważ nie tylko chcemy wykryć ludzką podobiznę, ale również odróżnić prawdziwą twarz od zdjęcia, maski itp. Ważne jest także rozpoznanie, gdy na przykład dana osoba ma makijaż, okulary, nakrycie głowy itp. Bez względu na to wszystko, podstawą do dalszych rozważań jest właśnie rozpoznanie twarzy na obrazie, o jakim pisaliśmy i odseparowanie go od reszty obrazu.