AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Informatyki, Elektroniki i Telekomunikacji

Katedra Informatyki

PROJEKT INŻYNIERSKI

Rozpoznawanie ekspresji twarzy w oparciu o sieci neuronowe konwolucyjne

Jacek Obłaza, Dmytro Petruk

OPIEKUN:
Dr hab. Bogdan Kwolek

Kraków, 2016

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY(-A) ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZY PROJEKT WYKONAŁEM(-AM) OSOBIŚCIE I SAMODZIELNIE W ZAKRESIE OPISANYM W DALSZEJ CZĘŚCI DOKUMENTU I ŻE NIE KORZYSTAŁEM(-AM) ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W DALSZEJ CZĘŚCI DOKUMENTU.

. . . . . .

PODPIS

Spis treści

1. Cel prac i wizja produktu

1.1. Problem

Rozpoznawanie ekspresji twarzy jest przydatne do codziennego funkcjonowania w społeczeństwie. Pozwala nam zrozumieć emocje drugiej osoby patrząc tylko na jej twarz. Nawet komunikatory internetowe pozwalają nam przekazać prostsze lub bardziej złożone ekspresje w postaci emotikon. Coraz bardziej popularne staje się oznaczanie twarzy osób na zdjęciach, które wcześniej zostają wykryte poprzez stworzone do tego programy. Po co ograniczać się tylko do wykrywania twarzy, skoro można od razu wykryć na nich ekspresję i ustalić emocje, które okazuje osoba na zdjęciu. Klient zażyczył sobie stworzenie biblioteki, dzięki której mógłby zrealizować rozpoznawanie ekspresji twarzy na obrazach.

1.2. Przegląd rozwiązań

Na rynku istnieją gotowe rozwiązania do rozpoznawania ekspresji twarzy, ale każde z nich posiada funkcjonalności, które nie pasują do celu realizowanego projektu.

- Microsoft Cognitive Services Emotion API [1]. Teraz jest jednym z najlepszych rozwiązań na rynku i z dostępną polityką cenową. Główną jego wadą, jak i większości innych rozwiązań jest to, że działa jako SaaS (Software as a Service), więc istnieje możliwość użycia tylko przy dostępie do sieci, co nie zawsze jest możliwe.
- Kairos [2]. Wadą jest brak możliwości użycia w trybie offline.
- EmoVu [3]. Tak jak wcześniejsze rozwiązania jedyny możliwy typ pracy to online.
- Nviso [4]. Chociaż jest uważany za bardzo dobre rozwiązanie ze strony technicznej, ale nie ma możliwości edycji, czy jakiejkolwiek zmiany.
- Nvidia DIGITS + DetectNet [5], chociaż nie służy do rozpoznawania emocji, a do rozpoznawania obiektów, ale zasadniczo działa podobnie do naszej biblioteki. Pozwala na podanie zbioru treningowego i uruchomienia uczenia sieci, przy jednoczesnym braku konieczności przejmowania się szczegółami technicznymi.

Wiele z tych rozwiązań umożliwia poprawne działanie tylko w trybie online, co nie zawsze jest możliwe. Nie pozwalają też one na poprawienie osiągów, czy spersonalizowanie wykrywania ekspresji. Ostatnie rozwiązanie jest bardzo ciekawe i uniwersalne, jednak nie dostarcza gotowej sieci stworzonej do wykrywania ekspresji, tylko trzeba samemu zebrać zbiór treningowy i wytrenować sieć. My chcielibyśmy jednak, aby nasze rozwiązanie do poprawnego działania nie wymagało od użytkownika niczego więcej niż instalacji.

1.3. Wizja

Nasza biblioteka będzie potrafiła wykrywać ekspresję twarzy na dostarczonym obrazie. Biblioteka ta będzie w stanie przypisać do danej twarzy na zdjęciu jedną ze znanych sobie ekspresji. Będzie można ją zastosować w komunikatorach i czatach internetowych, aby poprzez pobranie obrazu twarzy z kamery, program odczytał ekspresję i zamienił ją na odpowiadającą emotikonę.

Powinna też umożliwiać poprawianie swoich osiągów poprzez dostarczenie zdjęć, wraz z podpisanymi na nich ekspresjami. Jest to przydatne, kiedy wiemy, że twarz danych osób będzie występowała częściej niż innych, a biblioteka nie ma dla nich zadowalających osiągów.

1.4. Studium wykonalności

Rozpoznawanie ekspresji twarzy najlepiej realizować przy pomocy narzędzia stworzonego do rozpoznawania obrazów i znajdywania na nich wspólnych cech - konwolucyjnych sieci neuronowych. Charakteryzuje jest wysoka skuteczność w rozpoznawaniu charakterystycznych elementów obrazów takich jak w naszym przypadku kąciki ust, oczy, marszczenie brwi, itp.

Samo stworzenie sieci neuronowej nie powinno stanowić żadnego problemu. Co innego z znalezieniem odpowiedniej ilości przykładów. Sieć neuronowa powinna mieć odpowiednio dużą liczbę danych treningowych, aby była jak najbardziej ogólna, a nie przeznaczona tylko dla obrazów, na których się uczyła.

Drugim problemem jest proces uczenia sieci. Jest to proces bardzo czasochłonny i musi być często powtarzany, aby znaleźć jak najoptymalniejsze parametry sieci. Proces ten może trwać nawet kilka dni, dlatego trzeba zacząć go odpowiednio wcześnie.

Po zapoznaniu się dokładniej z sieciami neuronowymi konwolucyjnymi doszliśmy do wniosku, że najpierw należy przetestować sieć na dwóch emocjach i sprawdzić ile zajmuje jej wytrenowanie i jak duży musi być zbiór treningowy. Powoli zwiększając liczbę rozpoznawanych

1.5. Analiza ryzyka

Wstępne zagrożenia jakie przypuszczamy, że mogą wystąpić to:

- Błędne nauczenie sieci neuronowej, w tym przeuczenie, może wystąpić kiedy będziemy uczyli sieć zbyt długo na zbiorze treningowym lub sieć będzie miała nieodpowiednią ilość warstw, ilość neuronów w warstwach, rodzaje warstw, rozmiar warstwy wejściowej. Aby temu zaradzić musimy testować sieci z różnymi parametrami i wybierzemy z nich te z najlepszymi wynikami. Jeżeli po wielu próbach osiągi dalej nie będą nas zadowalały będziemy zmuszeni wrócić do mniejszej ilości rozpoznawanych ekspresji.
- Niewystarczający rozmiar zbioru treningowego danych to kolejny z problemów jakie możemy napotkać. Będziemy szukać odpowiednio dużego zbioru danych (zakładamy wstępnie co najmniej 2000 zdjęć), aby ten problem nie wystąpił. Jednak gdy okaże się, że to dalej jest za mało będziemy rozszerzać istniejący zbiór o zdjęcia zebrane przez nas.
- Czas rozpoznawania ekspresji nie może być też zbyt długi. Zakładamy, że chcemy aby proces ten przebiegał w czasie rzeczywistym. Zapobiegniemy tego poprzez testy, testy i jeszcze raz testy. Obraz wejściowy będziemy zmniejszać do rozmiarów takich, aby człowiek był w stanie rozpoznać ekspresję przedstawioną na danej twarzy. Jeśli to nie pomoże będziemy musieli ograniczyć się do dostarczania sieci samych fragmentów najważniejszych części twarzy takich jak oczy, usta, czy nos. Razem ze zmniejszaniem rozmiaru wejścia sieci, rozmiar kolejnych warstw także będzie malał, co powinno tym bardziej zapobiec problemowi z czasem.
- Nieodpowiednie narzędzia. Przed przystąpieniem do prac należy sprawdzić dostępne narzędzia do tworzenia sieci neuronowych. Jeżeli jakieś narzędzie nie będzie odpowiednie na jego miejsce znajdziemy inne, które spełni nasze wymagania.

2. Zakres funkcjonalności

2.1. Kontekst użytkownika produktu

W każdym przypadku użycia główną rolę gra aktor nazywany "Użytkownikiem". Może on chcieć wykonać jedną z dwóch głównych czynności oferowanych przez naszą bibliotekę:

- Rozpoznać ekspresję twarzy przedstawionej na podanym obrazie w oparciu o sieć wytrenowaną na przykładach treningowych
- Dotrenować sieć dostarczając własne obrazy twarzy z podpisanymi na nich ekspresjami. Po dotrenowaniu użytkownik będzie mógł zrealizować wcześniejszy punkt, ale wykorzystując sieć, która jest dostosowana do jego potrzeb i lepiej radzi sobie z twarzami osób dostarczonych przez niego do procesu dotrenowywania.

2.2. Wymagania funkcjonalne

1. Rozpoznawanie ekspresji twarzy na podstawie dostarczonego pliku graficznego
2. Możliwość poprawiania osiągów na podstawie własnych danych

2.3. Wymagania niefunkcjonalne

1. Działanie w czasie rzeczywistym
2. Działanie w trybie offline
3. Posiadanie API
4. Działanie na co najmniej dwóch systemach operacyjnych jakimi są Windows oraz Linux

3. Wybrane aspekty realizacji

3.1. Przyjęte założenia

Jak wiadomo obrazy można zapisać w różnych formatach graficznych. Przyjęliśmy, że nasza biblioteka będzie obsługiwała formaty: JPEG (.jpg, .jpeg, .jpe) oraz PNG (.png).

Docelowe API biblioteki:

- load_img(img_path)

Funkcja służąca do załadowania zdjęcia w obsługiwanym formacie, wykryciu na nim twarzy i przekształceniu do postaci macierzy odpowiadającej wejściu sieci neuronowej.

- build_cnn(model_path)

Funkcja do tworzenia architektury sieci na podstawie dostarczonego modelu z zapisanymi wagami.

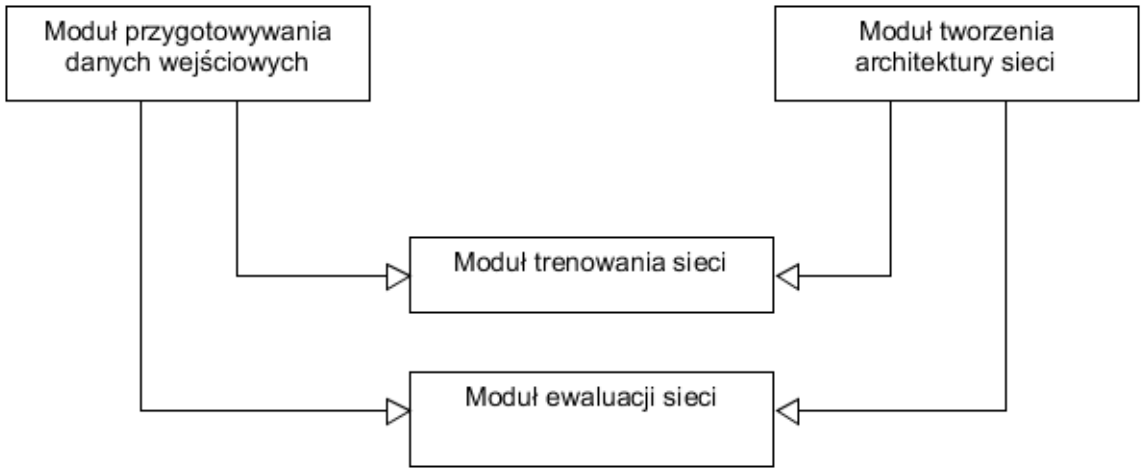
- evaluate(network, faces_matrix)

Funkcja predykująca ekspresję na dostarczonych twarzach za pomocą dostarczonej sieci.

- train_net(datadir, imagedir, labeldir, network, epochs=1000, save_each=False)

Funkcja służąca do trenowania sieci, wykorzystując jako zbiór treningowy obrazy wraz z ich podpisami zapisane w odpowiednio przygotowanej strukturze katalogów.

3.2. Struktura i zasada działania systemu



Rys. 1. Struktura biblioteki

Na rysunku 1. widać wszystkie moduły naszej biblioteki oraz zależności między nimi.

1. Moduł przygotowywania danych wejściowych odpowiedzialny jest za wczytanie obrazu z podanej ścieżki, przekształcenie go do skali szarości, wykrycie na nim twarz, żeby w końcu przeskalować sam fragment zawierający twarz do macierzy rozmiaru 96x96x1. Odpowiada

też za wczytanie danych potrzebnych do trenowania sieci z odpowiedniej struktury katalogów.

2. Moduł tworzenia architektury sieci służy do przygotowania odpowiedniej struktury sieci. Potrzebne jest jego wywołanie zarówno kiedy chcemy trenować sieć, jak też użyć jej do klasyfikacji przynajmniej jednego obrazu. To on odpowiada za to aby zawsze tworzona sieć miała taką samą budowę. Pozwala też wczytać odpowiednie wagi krawędzi z wcześniej zapisanego pliku.
3. Moduł trenowania sieci odpowiada za poprawne trenowanie sieci na podstawie danych dostarczonych przez moduł przygotowywania danych wejściowych. Trenowanie podzielone jest na epoki, których liczbę można podać jako argument funkcji trenującej. Jeżeli nie zostanie podany, funkcja będzie wykonywać się, dopóki różnica wartości funkcji błędu dla dwóch kolejnych epok nie będzie większa niż 0.0001.
4. Moduł ewaluacji sieci to moduł, z którym użytkownik ma najwięcej interakcji. To w nim przeprowadzane jest rozpoznawanie ekspresji twarzy na danym obrazie, co jest najważniejszą funkcjonalnością systemu.

3.3. Wykorzystane rozwiązania technologiczne

Język programowania

Całość kodu projektu napisana jest w języku **Python**. Jest on prosty, szybko się w nim tworzy małe projekty oraz pozwala na szybkie prototypowanie. Co jednak było najważniejszym kryterium podczas decydowania się na taki język jest to, że większość narzędzi do tworzenia sieci neuronowych jest właśnie dla niego stworzona.

Dużym minusem mogło okazać się jego wolne działanie, jednak jest ono wystarczająco szybkie dla potrzeb tego projektu.

Narzędzie do tworzenia sieci neuronowych

Na rynku znajduje się wiele narzędzi pozwalających na zbudowanie, wytrenowanie i testowanie własnej sieci neuronowej. Wiele z nich umożliwia także tworzenie głębokich sieci neuronowych, w tym sieci konwolucyjnych. Biblioteki brane pod uwagę podczas tworzenia tego projektu:

1) **TensorFlow** [6]
2) **Theano** [7]
3) **Caffe** [8]
4) **Lasagne** [9]
5) **Keras** [10]

TensorFlow i Theano pozwalają na wydajną definicję, optymalizację i ewaluację wyrażeń matematycznych używających wielowymiarowych macierzy. Obie pozwalają na współbieżne trenowanie sieci z wykorzystaniem GPU. TensorFlow do tego pozwala w prosty sposób wizualizować sieć za pomocą TensorBoard. Są dobrymi narzędziami do budowania całego projektu od podstaw. Ich niskopoziomowość jest ich dużym atutem, jak i dużą wadą ze względu na złożoność procesu tworzenia sieci neuronowej.

Caffe jest bardzo potężnym narzędziem, jednak po wstępnych testach i implementacjach przykładowych sieci konwolucyjnych okazało się, że jej API jest nieoczywiste. Tak samo jak poprzednie biblioteki pozwala na uczenie sieci przy pomocy GPU.

Keras i Lasagne są do siebie podobne pod względem API i możliwości. Wybór między nimi nie był jednak trudny. Obie tak samo są nakładką na Theano, która sprawia, że staje się ono wyskopoziomowym narzędziem. Do Lasagne można jednak znaleźć więcej poradników, przykładowych sieci i pomocy w sieci, co zaważyło o wyborze tej właśnie biblioteki.

Narzędzie do rozpoznawania obrazów

Przed przystąpieniem do trenowania i ewaluacji sieci należy przygotować obraz wejściowy. Zdjęcia czegoś więcej niż samej twarzy wprowadziłyby niepotrzebny element zaburzający wyniki klasyfikacji. Mogłoby się okazać, że sieć klasyfikuje emocję jako strach, jeśli na obrazie poza twarzą występuje jakiś inny konkretny obiekt. Aby ograniczyć obraz do samej twarzy należy najpierw ją na nim znaleźć, do czego posłużyła bilioteka **OpenCV** [11]

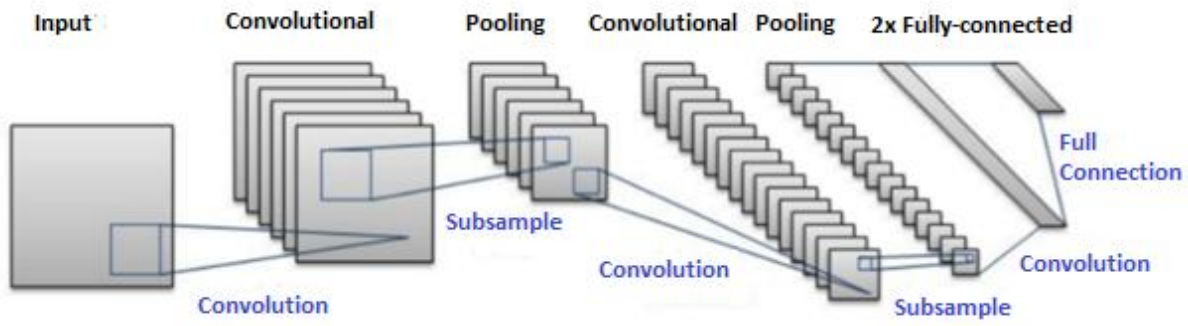
OpenCV to biblioteka funkcji wykorzystywanych podczas obróbki obrazu, zapoczątkowana przez Intela. Biblioteka ta jest wieloplatformowa, można z niej korzystać zarówno na Mac OS X, Windows, jak i Linux. Jej autorzy skupiają się na przetwarzaniu obrazu w czasie rzeczywistym.

Z tej biblioteki był użyty głównie mechanizm rozpoznawania twarzy algorytmem Viola-Jones, który działa na podstawie cech kaskad Haar'a i został opisany w artykule z 2001 roku ("Rapid Object Detection using a Boosted Cascade of Simple Features" [12]). Jest bazowany na metodach uczenia maszynowego, gdzie funkcja kaskad jest trenowana na wielu obrazach pozytywnych i negatywnych (obrazy pozytywne - zawierają szukany obiekt, a negatywne - nie zawierają). Następnie jest wykorzystany do detekcji obiektu na nowych obrazach. OpenCV zawiera już wytrenowaną konfigurację detektora, więc została ona użyta do detekcji twarzy. [13]

3.4. Szczegóły realizacji

3.4.1. Architektua sieci

W naszej bibliotece zastosowaliśmy sieć o architekturze LeNet.



Rys. 2. Budowa sieci LeNet

Sieć składa się kilku warstw. Pierwszą z nich jest input layer, która w naszym przypadku ma wymiary 96x96x1 (wysokość, szerokość i jedna skala kolorów - skala szarości). Testowaliśmy też inne rozmiary wejścia, jednak dla 128x128 czas jednej epoki przekraczał 10 minut, a 64x64 były zbyt małe, aby na obrazie dało się określić ekspresję twarzy. Kolejne warstwy to convolutional layer razem z pooling layer.

W convolutional layer zastosowaliśmy 32 filtry, każdy o wymiarach 5x5. Znaczy to tyle, że po obrazie wejściowym przesuwamy 32 okna o wymiarach 5x5, każde z innym zestawem, różnych wag w każdej komórce. Wynikiem działania pojedynczego okna jest suma iloczynów wagi i odpowiadającej jej komórki w obrazie wejściowym. Przesuwając tak piksel po pikselu otrzymamy 32 obrazy o wymiarach 92x92.

Pooling layer służy do zmniejszenia rozmiaru przetwarzanego obrazu. Zastosowany przez nas wariant zmniejsza go czterokrotnie, gdyż zastosowaliśmy filtr o rozmiarach 2x2, a odległość, o którą jest przesuwany ustawiliśmy na 2. Funkcją, która jest stosowana dla każdej czwórki liczb pod filtrem, jest funkcja MAX, zwracająca wartość maksymalną spośród argumentów.

Kolejne dwie warstwy mają identyczne parametry, dzięki czemu przekształcamy nasz obraz wejściowy do macierzy cech o rozmiarach 21x21x32.

Następną warstwą jest fully-connected layer o 256 neuronach i zastosowanym dropout'cie 50% losowych neuronów poprzedniej warstwy. Zapobiega to przeuczeniu sieci i znacząco zmniejsza liczbę połączeń, skutkując zwiększeniem szybkości procesu uczenia.

Ostatnią warstwą również jest fully-connected layer, lecz tym razem o liczbie neuronów odpowiadającej ilości rozpoznawanych ekspresji twarzy. Również w połączeniu między tą warstwą a poprzednią zastosowaliśmy dropout 50%.

4. Organizacja pracy

4.1. Podział prac

Projekt wykonywany był przez dwie osoby, których zadania prezentuje Tabela 1.

Tabela 1. Podział prac w zespole

Dmytro Petruk	Jacek Obłaza
- Przeprowadzenie analizy dostępnych narzędzi - Przygotowanie zbioru treningowego - Implementacja wczytywania danych - Trenowanie i ewaluacja sieci	- Implementacja wykrywania twarzy i dostosowywania jej do wejścia sieci - Implementacja trenowania sieci neuronowej - Trenowanie i ewaluacja sieci

4.2. Przyjęta metodyka i kolejność prac

Jako metodologię projektowania systemu wybraliśmy połączenie Agile i Lean Programming, co pozwoliło na działanie iteratywne, inkrementalne i ewolucyjne w rozbudowie produktu wraz z krótkimi cyklami dla eliminacji działań nie przyczyniających się do realizacji wyznaczonych celów. Całość była podzielona na kilka iteracji wynikających głównie z długiego czasu trenowania sieci.

1. Analiza wymagań i wybór narzędzi
2. Stworzenie pierwszego prototypu sieci
3. Trenowanie i poprawki
4. Szukanie odpowiedniego zbioru danych treningowych
5. Trenowanie i poprawki
6. Trenowanie i poprawki
7. Trenowanie i poprawki
8. Trenowanie i poprawki
9. Porządkowanie i refaktoryzacja kodu

4.2.1. Analiza wymagań

Na tym etapie przeprowadziliśmy analizę rynku, dostępnych narzędzi oraz wymagań produktu. To tutaj podjęliśmy decyzje o tym, aby dodać możliwość dotrenowywania sieci oprócz samego rozpoznawania ekspresji twarzy. Praktycznie od samego początku wiedzieliśmy, że językiem programowania będzie Python, ze względu na jego specyfikę i niewielki rozmiar projektu. Bibliotekę do wykrywania twarzy na obrazie znaleźliśmy od razu po wpisaniu w wyszukiwarkę "python face recognition". OpenCV spełniło nasze oczekiwania i nie szukaliśmy już dalej. Nie jest to główny problem naszego systemu, ani też nie zależy od niego czas działania, więc nie było potrzeby sprawdzania, czy istnieją na rynku inne, być może lepsze rozwiązania.

Największym problemem był wybór biblioteki do tworzenia sieci. Znaleźliśmy ich dużo więcej niż zostało to wspomniane, jednak do dalszej analizy wybraliśmy tylko 5. Do testów wzięliśmy Caffe, Lasagne i Keras. W każdej z tych bibliotek stworzyliśmy sieć z przykładu i porównaliśmy je pod kątem przejrzystości kodu, szybkości działania i ilości iteracji wymaganych do osiągnięcia zadowalającego nas wyniku. Wszystkie testu przeprowadzane były wtedy na zbiorze MNIST. O ile czas i ilość iteracji były do siebie bardzo zbliżone (różnice na poziomie milisekund), to przejrzystość kodu Caffe nie powalała. Zostały więc tylko Lasagne i Keras. Obie bardzo podobne do siebie, jednak czynnikiem przemawiającym za Lasagne była ilość przykładów i tutoriali, które można znaleźć w sieci.

4.2.2. Stworzenie pierwszego prototypu sieci

W tej fazie zaprojektowaliśmy wstępną architekturę sieci. Po analizie sieci znalezionych w poradnikach doszliśmy do wniosku, że najodpowiedniejsza będzie LeNet. Jest ona bardzo uniwersalna i prosta, więc idealnie nadaje się jako pierwsza sieć konwolucyjna.

4.2.3. Trenowanie i poprawki

Z siecią przygotowaną w poprzedniej iteracji przystąpiliśmy do trenowania jej. W pierwszej iteracji treningu zastosowaliśmy zbiór złożony ze stu naszych zdjęć ze smutną oraz z radosną ekspresją twarzy (po 25 danej ekspresji na osobę). Trening sieci na tak małym zbiorze przebiegał bardzo szybko. Jedna epoka trwała mniej niż 5 sekund, a funkcja już po 500 iteracja miała wartość mniejszą od 0.0001.

Po takim treningu należało przetestować działanie sieci. Zrobiliśmy kolejne 12 zdjęć, tym razem już z mniej przekonywującą ekspresją i użyliśmy ich jako zbioru testowego. Wyniki dawane przez sieć były bardzo jednostronne (w granicach 98% nasycenia daną ekspresją). Ekspresje na 10 zdjęciach zostały rozpoznane prawidłowo, co nas satysfakcjonowało. Doszliśmy do wniosku, że źle rozpoznane ekspresje na 2 zdjęciach były spowodowane zbyt małym i mało zróżnicowanym zbiorem treningowym.

4.2.4. Szukanie odpowiedniego zbioru danych treningowych

Poszukiwany przez nas zbiór treningowy musiał spełniać następujące założenia:

- Zdjęcia muszą mieć podpisaną na nich ekspresję twarzy
- Co najmniej 2000 zdjęć
- Co najmniej 4 różne ekspresje

Zbiór "cohn-kanade-images", który znaleźliśmy spełniał nasze wymagania. Jest tam ponad 10 000 zdjęć twarzy, z czego około połowa ma podpisane ekspresje. Liczba różnych ekspresji w tym zbiorze wynosi 7. Są to: złość, pogarda, zniesmaczenie, strach, radość, smutek, zaskoczenie.

4.2.5. Trenowanie i poprawki

Kolejne iteracje trenowania sieci były bardzo czasochłonne, gdyż dla tak dużego zbioru danych jedna epoka treningu trwa około 5 minut wykorzystując domowy komputer. Pod uwagę przy porównywaniu wyników braliśmy wartość funkcji błędu, ilość epok potrzebną na wytrenowanie sieci, skuteczność rozpoznawania na zbiorze testowym.

Liczyliśmy, że zmiany w parametrach warstw sieci dadzą jakąś poprawę, jednak po czterech próbach żadna nie okazała się ani szybsza w trenowaniu, ani nie miała lepszej skuteczności niż sieć stworzona na samym początku jako prototyp na podstawie przykładowej sieci z poradnika, więc zostaliśmy przy początkowych parametrach.

4.2.6. Porządkowanie i refaktoryzacja kodu

Ostatnim krokiem było uporządkowanie kodu i zrefaktoryzowanie go, tak aby funkcje działały dokładnie tak jak stanowi ich opis. Nie sprawiło to żadnego problemu, a proces ten nie zajął więcej niż godzinę.

4.3. Planowane i zrealizowane funkcjonalności

Rozpoznawanie ekspresji twarzy na podstawie dostarczonego pliku graficznego

To założenie zostało zrealizowane, jednak jak zostanie pokazane w dalszej części dokumentu nie na takim poziomie jaki nas zadowala. Biblioteka rozpoznaje ekspresje twarzy takie jak: złość, pogarda, zniesmaczenie, strach, radość, smutek, zaskoczenie.

Możliwość poprawiania osiągów na podstawie własnych danych

Ten punkt został w pełni zrealizowany. Biblioteka udostępnia mechanizm douczenia sieci neuronowej na podstawie własnych zdjęć twarzy z podpisanymi na nich ekspresjami. Jest to proces czasochłonny, jednak jak przetestowaliśmy, działający.

Działanie w czasie rzeczywistym

Działanie rozpoznawanie ekspresji twarzy działa w czasie rzeczywistym. Czas rozpoznawania trwa nie dłużej niż 0,5s, więc można ją zastosować do obrazów z kamery, jednak oczywiście nie dla każdej klatki, tylko dla 2 klatek na sekundę.

Działanie w trybie offline

Biblioteka działa całkowicie bez użycia połączenia z internetem. Jest ono jedynie potrzebne do pobrania zbioru treningowego, potrzebnego do dotrenowania sieci neuronowej.

Posiadanie API

Biblioteka udostępnia funkcje wymienione w poprzednim punkcie. Wystarczają one w pełni do zrealizowania dwóch podstawowych zadań, czyli rozpoznania ekspresji twarzy z obrazu oraz poprawienia osiągów dla niektórych twarzy.

4.4. Wykorzystane praktyki i narzędzia

Jako pomoce przy tworzeniu projektu wykorzystywaliśmy takie narzędzia jak:

- Github - narzędzie do kontroli wersji
- Trello - Kanban board, który posłużył do zapisywania zadań na kolejne etapy

Obaj korzystaliśmy już z tych narzędzi wcześniej, dzięki czemu wiedzieliśmy juz przed rozpoczęciem prac, że spełniają swoją rolę i w pełni nam wystarczają. Okazały się one bardzo pomocne do koordynowania naszych działań i zdecydowanie uprościły proces tworzenia kolejnych części projektu.

5. Wyniki projektu

5.1. Wyniki

Wynikiem naszych prac jest gotowa biblioteka. Spełnia ona założenia postawione podczas etapu planowania. Do biblioteki załączona jest dokumentacja opisująca proces powstawania kodu, sposób działania i szczegóły realizacji oraz instrukcję korzystania z biblioteki. Poprawność działania biblioteki zweryfikowana jest testami jakościowymi.

Kod pracy wraz ze zbiorem treningowym znajdują się w repozytorium pod adresem: https://github.com/bavaria95/thesis

Wykorzystując część zdjęć ze zbioru "cohn-kanade-images", która nie była użyta w zbiorze treningowym przeprowadziliśmy testy. Każdy wiersz odpowiada średniej wartości stopni nasycenia każdą z ekspresji dla wszystkich obrazów reprezentujących tę samą ekspresję.

Tabela 2. Wyniki dla zbioru testowego

	złość	pogarda	zniesmaczenie	strach	radość	smutek	zaskoczenie
złość	95,82	0,01	1,05	0,01	1,82	0,39	0,90
pogarda	0,00	100,00	0,00	0,00	0,00	0,00	0,00
zniesmaczenie	0,15	0,04	94,53	1,12	1,36	0,47	2,33
strach	3,94	0,00	0,00	92,06	3,98	0,02	0,00
radość	0,03	0,02	1,07	0,43	97,45	0,12	0,89
smutek	0,09	0,02	0,03	0,03	0,19	96,47	3,17
zaskoczenie	0,04	0,03	1,25	0,19	2,68	1,05	94,77

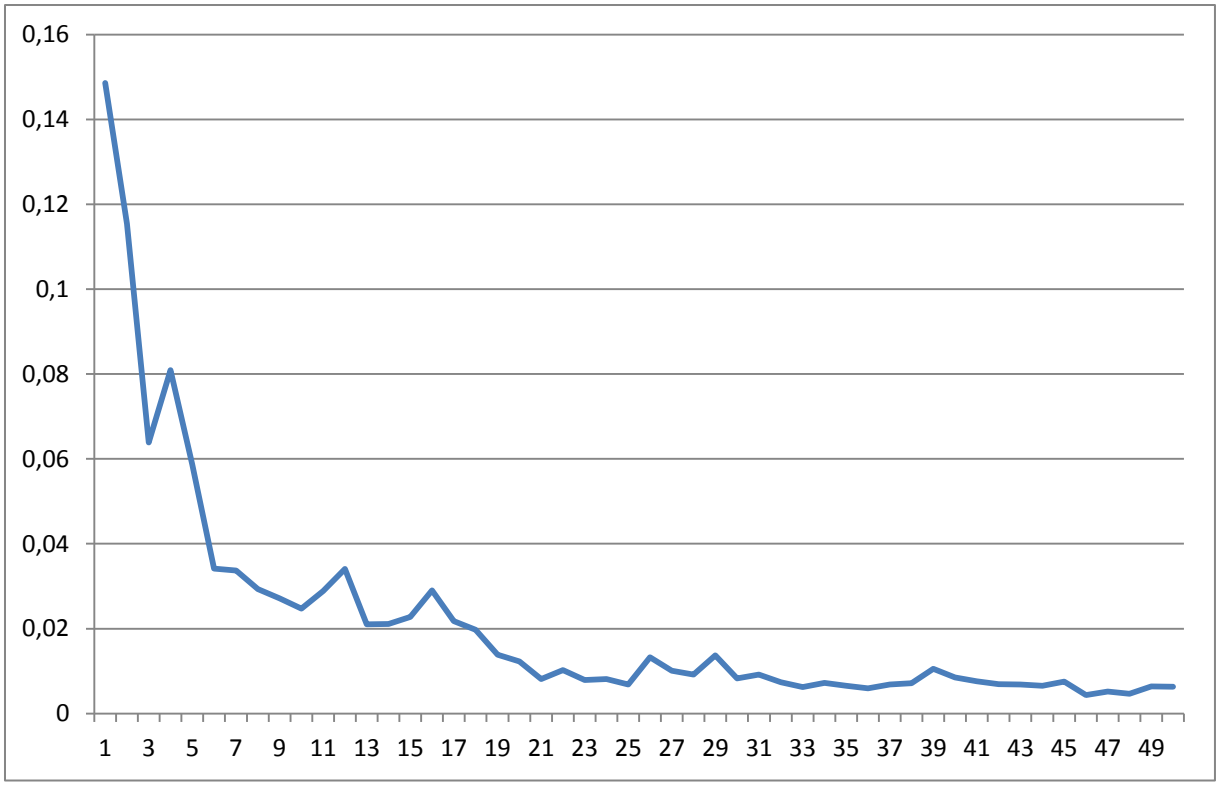
Jak widać sieć jest nauczona poprawnie dla danych ze wspomnianego zbioru. Postanowiliśmy więc przetestować sieć na naszych własnych twarzach. Zrobiliśmy po 5 zdjęć czterech emocji i poddaliśmy je takim samym testom jak powyższy.

Tabela 3. Wyniki dla własnych zdjęć

		Rozpoznana ekspresja						
		złość	pogarda	zniesmaczenie	strach	radość	smutek	zaskoczenie
Poprawna ekspresja	zniesmaczenie	56,68	15,00	5,64	21,86	0,00	0,53	0,03
	radość	0,06	23,48	17,90	22,90	35,66	0,00	0,00
	smutek	0,00	1,89	2,47	79,02	0,18	0,00	16,74
	zaskoczenie	0,11	0,00	44,05	0,01	0,00	0,00	55,83

Wyniki te nie są tak dobre jak poprzednie. Wiele ekspresji jest źle rozpoznawana. Nie wiemy czy to ze względu na to, że sami nie potrafimy dobrze ich przedstawić, czy może dlatego, że zbiór treningowy był zbyt mały i niezróżnicowany. W takim przypadku należałoby dotrenować sieć z wykorzystaniem naszych zdjęć.

Do dotrenowania sieci wykorzystaliśmy cały zbiór treningowy, na którym była uczona wcześniej, jak również zdjęcia zrobione do wcześniejszych testów, których wyniki przedstawione są w tabeli 3. Jako podstawę sieci użyliśmy sieci wcześniej nauczonej na naszym zbiorze treningowym.



Wykres 1. Funkcja błędu douczania sieci od epoki

Sieć trenowana była przez 50 epok jednak, jak widać z wykresu 1, na którym przedstawiono wartość funkcji błędu w zależności od epoki, podobne wartości były osiągane już w 20 epoce. Tak niska wartość początkowa funkcji błędu wynika z tego, że tylko dla 1% danych sieć nie była trenowana wcześniej.

Po dotrenowaniu wykonaliśmy ponownie test na własnych obrazach. Użyliśmy innych zdjęć niż poprzednio, jednak tym razem po 2 zdjęcia na każdą z wcześniejszych ekspresji. Jak można odczytać z tabeli 4. wyniki uległy zdecydowanej poprawie. Każda emocja rozpoznawana jest prawidłowo, co świadczy o poprawnym działaniu dotrenowywania sieci.

Tabela 4. Wyniki dla własnych zdjęć po dotrenowaniu sieci

		Rozpoznana ekspresja						
		złość	pogarda	zniesmaczenie	strach	radość	smutek	zaskoczenie
Poprawna ekspresja	zniesmaczenie	0,19	0,00	90,62	0,86	0,45	0,07	7,80
	radość	0,26	0,02	0,29	2,10	96,52	0,82	0,00
	smutek	3,47	0,01	0,01	0,03	1,18	95,29	0,00
	zaskoczenie	0,15	1,41	0,00	2,38	0,12	1,00	94,95

5.2. Ograniczenia

Podczas prac nad biblioteką ujawniło się kilka aspektów, które należałoby dopracować, jednak brakło na to czasu i czasami pomysłu. Należą do nich:

5.2.1 Wymagany zbiór treningowy podczas douczania sieci

Aby douczanie przebiegło pomyślnie i sieć nie straciła umiejętności klasyfikacji ekspresji wcześniej już nauczonych twarzy podczas douczania musi być załadowany zbiór treningowy,

na którym sieć była już uczona. Jest to o tyle niewygodne, że wraz z biblioteką musi być dostarczany ten zbiór, a zajmuje on ponad 1GB. Można próbować dotrenowywać sieć posiadając sam jej model, jednak musi się to odbywać małą liczbę epok (3 lub mniej) i nie ma gwarancji, że sieć nie straci umiejętności rozpoznawania ekspresji dla niektórych twarzy, czy też nawet będzie dla nich rozpoznawać nieprawidłowo.

5.2.2. Złe rozpoznawanie dla niektórych twarzy

Jak pokazały testy na naszych twarzach biblioteka miała problemy z rozpoznaniem smutnej ekspresji i klasyfikowała ja jako strach. Wymagane jest wtedy dotrenowanie sieci, jednak jest to niewygodne, aby dla każdej nowej twarzy, dla której biblioteka nie będzie poprawnie działać dotrenowywać ją. Zawarliśmy ten punkt w analizie ryzyka i obawialiśmy się go najbardziej. Jak widać słusznie. Testowaliśmy jednak kilka różnych parametrów sieci i dla tych konkretnych osiągi były najlepsze. Należałoby zatem zmniejszyć liczbę rozpoznawanych ekspresji, jednak brakło już czasu na kolejne testy i trenowanie, gdyż trwa to w najlepszym wypadku 2 tygodnie. Innym rozwiązaniem jest rozszerzenie zbioru treningowego o kolejne obrazy wraz z podpisami, jednak wtedy proces uczenia byłby jeszcze bardziej czasochłonny.

5.3. Propozycje dalszych prac

Kolejnym krokiem powinno być testowanie osiągów sieci w zależności od wielu różnych parametrów i ilości rozpoznawalnych ekspresji. Poprawi to na pewno jakość klasyfikacji ekspresji.

Inną propozycją jest rozszerzenie zbioru treningowego o kolejne zdjęcia twarzy z podpisanymi ekspresjami. Aktualny zbiór to około 2000 obrazów nadających się do treningu, co jest jak widać niewystarczającą ilością.

6. Bibliografia

[1] https://www.microsoft.com/cognitive-services/en-us/emotion-api
[2] https://www.kairos.com
[3] http://emovu.com
[4] http://www.nviso.ch
[5] https://developer.nvidia.com/digits
[6] https://www.tensorflow.org
[7] http://deeplearning.net/software/theano
[8] http://caffe.berkeleyvision.org
[9] https://lasagne.readthedocs.io
[10] https://keras.io
[11] http://opencv.org
[12] https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
[13] http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html