Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Klasyfikowanie emocji na twarzy człowieka z wykorzystaniem sieci neuronowych

Anna Modrzejewska
Nr indeksu: 236642

Praca inżynierska napisana
pod kierunkiem
dra Krzysztofa Majchra

Politechnika
Wrocławska
Wrocław 2019

Spis treści

Wstęp

Wyraz twarzy człowieka odgrywa ważną rolę na wielu płaszczyznach, takich jak spersonalizowane rekomendacje, interakcja człowieka z komputerem, interpretacja obrazów. Z tego powodu automatyczne rozpoznanie emocji stało się przedmiotem badań dla wielu ludzi przez ostatnie 30 lat. Występujący tu problem wykrycia i klasyfikacji wzorca na obrazie jest jednym z podstawowych obszarów, w których uczenie maszynowe znajduje zastosowanie.

Celem pracy jest stworzenie architektury sieci neuronowej oraz wytrenowanie modelu zdolnego do rozpoznania podstawowych emocji człowieka, bazując na zdjęciu jego twarzy. Głównym problemem pracy będzie odpowiedni dobór parametrów przy tworzeniu sieci tak, aby zminimalizować funkcję straty na zbiorze walidacyjnym. Na koniec otrzymany model zostanie użyty do przetworzenia strumienia wideo na żywo.

Praca składa się z sześciu rozdziałów. Rozdział pierwszy stanowi niniejszy wstęp.

W rozdziale drugim omówiono teoretyczne podłoże (kluczowe pojęcia) dotyczące konwolucyjnych sieci neuronowych oraz procesu trenowania modelu. Przedstawiono planowany przebieg prac.

W rozdziale trzecim przedstawiono szczegółowo proces projektowania modelu. Dokładnie opisano zbiór danych służących do trenowania modelu, a także sposób ich wstępnego przetworzenia. Przedstawiono kolejne kroki budowania sieci neuronowej, a następnie dobrane parametry przy trenowaniu i kompilacji modelu wraz z opisem i uzasadnieniem. Omówiono proces przetwarzania strumienia wideo na żywo przy użyciu otrzymanego wcześniej modelu.

W rozdziale czwartym pokazano rezultat pracy w postaci wyników testów wraz z wnioskami.

W rozdziale piątym opisano technologie implementacji projektu: wybrany język programowania, biblioteki.

Ostatni rozdział zawiera krótke podsumowanie uzyskanego wyniku oraz dalsze możliwości ulepszenia go, a także zaproponowano ścieżki użycia.

Analiza problemu

W niniejszym rozdziale omówiono teoretyczne podłoże problemu maszynowej analizy zdjęcia w celu umiejscowienia w przestrzeni uczenia maszynowego. Przedstawiono cykl pracy dla danego zagadnienia. Omówiono sposób użycia projektowanego systemu.

2.1 Omówienie teoretyczne

Uczeniu maszynowym można podzielić na dwa rodzaje ze względu na dostarczone dane: nadzorowane i nienadzorowane. W przypadku uczenia nadzorowanego zbiór danych treningowych jest poprawnie oetykietowany, to znaczy dla danej wejściowej znana jest dana wyjściowa, w odróżnieniu do uczenia nienadzorowanego, gdzie model znajduje regularności wśród dostarczonych danych bez wiedzy a priori dotyczącej poprawnych etykiet. W tym przypadku potrzebny jest znacznie większy zbiór danych niż przy uczeniu nadzorowanym.

Problemy uczenia nadzorowanego można z kolei podzielić na problemy klasyfikacji i regresji. Różnią się one rodzajem zwracanego wyniku. W przypadku klasyfikacji jest to przypisanie do jednej z kategorii, natomiast w przypadku regresji jest to dokładna, wyliczona wartość. Niniejsza praca skupia się na problemie klasyfikacji.

Jako że jest to rozpoznawanie wzorców na zdjęciach, zostanie zaprojektowana konwolucyjna sieć neuronowa (CNN), która jest najpopularniejsza w tego typu zagadnieniach. Konwolucja, w maszynowym przetwarzaniu obrazów, to nałożenie na obraz odpowiednio skonstruowanych filtrów, co pozwala na jego dokładniejszą analizę i wyodrębnienie z niego cech.

2.2 Etapy pracy

Proces tworzenia klasyfikatora będzie stosował typowy machine learningowy cykl pracy:

1. Zbadanie zbioru danych
2. Przygotowanie danych wejściowych
3. Zbudowanie modelu
4. Trenowanie modelu
5. Testowanie modelu
6. Ulepszenie modelu i powtórzenie procesu

Po otrzymaniu modelu z akceptowalną dokładnością zostanie stworzony program do analizy strumienia wideo w czasie rzeczywistym. Będzie on wykrywał na wideo wycinki klatki obejmujące twarze, konwertował je do postaci odpowiedniej dla wytrenowanego klasyfikatora, a następnie wypisywał na wyjściu emocje znalezione z największym prawdopodobieństwem.

Poszczególne etapy projektu zostały szczegółowo omówione w kolejnych rozdziałach.

Projekt modelu

W tym rozdziale zostanie omówiony zbiór danych, na których będzie trenowany model, a także zostanie zaproponowana architektura modelu wraz z użytymi parametrami. Na koniec zostanie opisany sposób działania programu do klasyfikacji emocji.

3.1 Dane

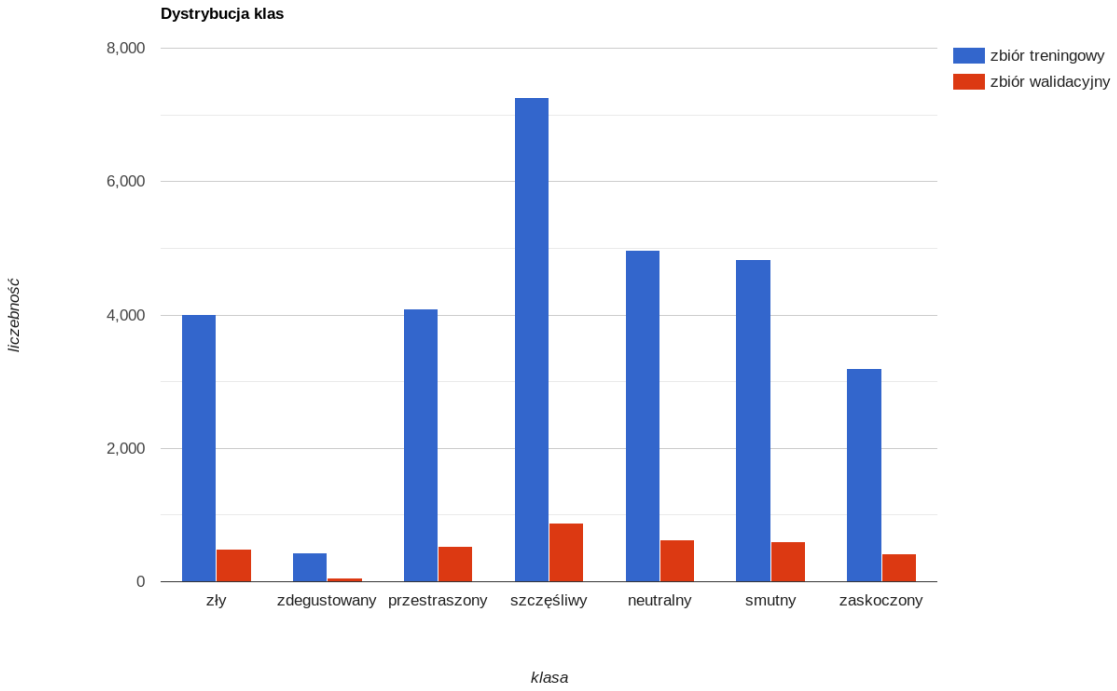
Model zostanie trenowany na zbiorze danych FER2013 udostępnionym przez Pierre-Luca Carriera i Aarona Courville'a do konkursu na platformie Kaggle [9]. Składa się on z 32298 zdjęć podzielonych na dwa zbiory - treningowy (75% zbioru) i testowy (25% zbioru). Każde zdjęcie jest czarno-białe, ma wymiary 48 pikseli na 48 pikseli i ukazuje twarz człowieka w jednej z siedmiu ekspresji: szczęśliwy, neutralny, zdziwiony, zły, smutny, przestraszony, zdegustowany. Zdjęcia są uprzednio oetykietowane poprawnymi klasami. Przykładowe dane widnieją na rysunku 3.1:

Zdziwiony Szczęśliwy Neutralny

Zły Smutny Przestraszony

Rysunek 3.1: Przykładowe zdjęcia ze zbioru FER2013

Dystrybucję zdjęć w poszczególnych klasach przedstawia wykres na rysunku 3.2.



Rysunek 3.2: Dystrybucja klas

Można zauważyć zdecydowaną przewagę zdjęć ukazujących szczęście nad pozostałymi klasami oraz małą liczbę wystąpień zdegustowania, natomiast pozostałe klasy są dobrze zbalansowane.

3.2 Wstępne przetworzenie

3.2.1 Formatowanie danych

Przed dostarczeniem zdjęć do wytrenowania modelu, zostały one odpowiednio sformatowane. Po wczytaniu zdjęć z dysku, zostały one przekonwertowane do postaci tensorów. Następnie poddano je normalizacji - wartości od 0 do 255 przeskalowano do wartości od 0 do 1, jako że sieci neuronowe wykazują lepsze działanie na małych wartościach wejściowych.

Zdjęcia podczas trenowania pomieszano oraz podzielono na mniejsze zestawy (batches) w celu zrównoważenia wykorzystywania pamięci.

3.2.2 Powiększenie zbioru danych

Do trenowania danych na podstawie posiadanych zdjęć z bazy FER2013, wygenerowano inne, lekko zmodyfikowane w porównaniu z oryginałami. Działaniami, którym zostały poddane zdjęcia, są:

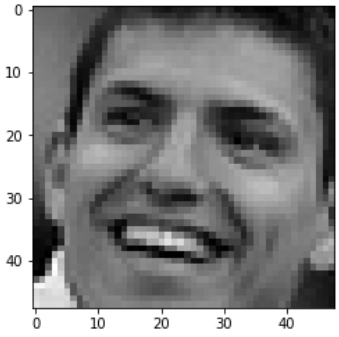
- przesunięcie obrazu do 20% pionowo
- przesunięcie obrazu do 20% poziomo

- powiększenie obrazu do 20%,
- rotacja do 20 stopni,
- odbicie lustrzane.

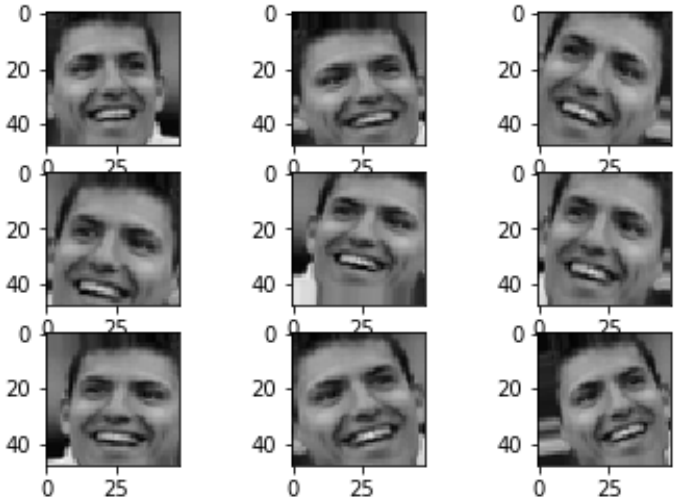
Wynikowe zdjęcia pozostają wiarygodne, czarno-białe i o wymiarach 48px na 48px. Różne rodzaje transformacji zdjęcia przedstawia rysunek 3.4. Dzięki takiemu procesowi zbiór danych do trenowania znacząco się powiększy, co w tym przypadku pozwoli na lepsze dopasowanie się końcowego modelu oraz zmniejszy ryzyko wystąpienio overfittingu, czyli zjawiska występującego, kiedy klasyfikator skupia się na zapamiętywaniu uczonych danych, zamiast na poszukiwaniu wzoru. Model traci wtedy zdolność do generalizacji i źle analizuje dane, które nie były użyte do trenowania.

Transformacje są dokonywane losowo i zastosowywane podczas procesu uczenia. Taki proces powiększania zbioru treningowego nazywa się *data augmentation*.

Zbiór walidacyny należy jedynie przekonwertować do postaci tensorów, podzielić na mniejsze zestawy i przeskalować do wartości [0,1]. Nie ma potrzeby powiększania tego zbioru danych.



Rysunek 3.3: Oryginał



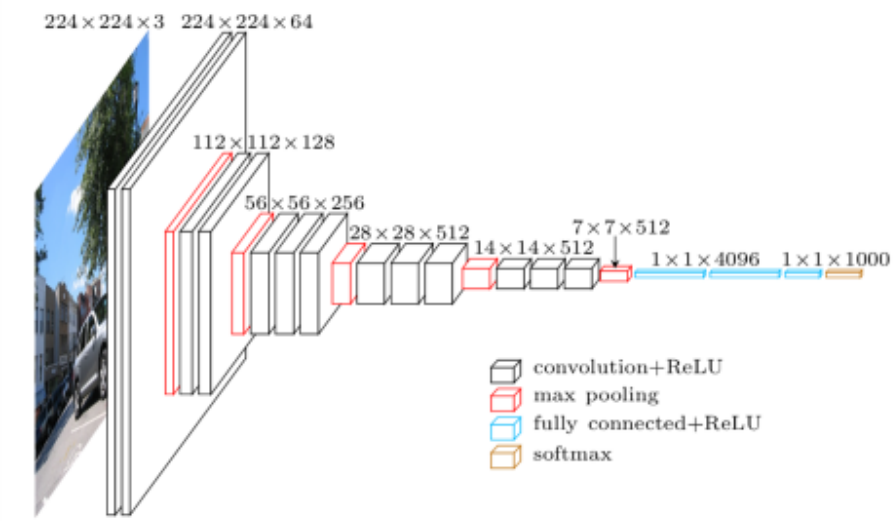
Rysunek 3.4: Zdjęcia po transformacji

3.3 Tworzenie sieci neuronowej

Zostały stworzone dwie wersje modelu, jedna całkowicie od podstaw, druga z wykorzystaniem wytrenowanego wcześniej modelu.

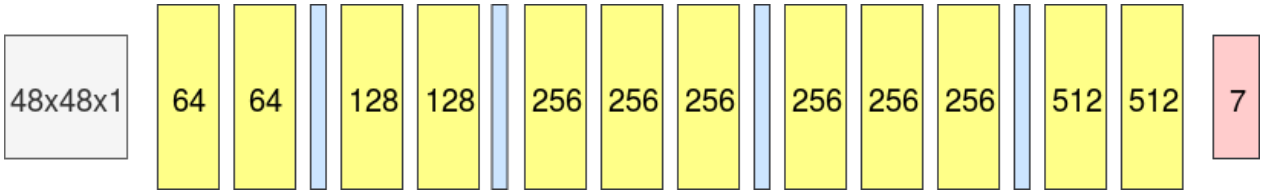
3.3.1 Od podstaw

Model od podstaw został zaprojektowany na podstawie architektury VGG16 (rysunek 3.5).



Rysunek 3.5: Wizualizacja architektury VGG16, źródło: `www.cs.toronto.edu` [5]

Poglądowy rysunek zaproponowanej architektury znajduje się na rysunku 3.6, gdzie żółte bloki oznaczają warstwy konwolucyjne, liczby w nich reprezentują liczbę filtrów dla danej warstwy, natomiast niebieskie bloki to warstwy poolingu. Charakterystyczna dla architetury VGG jest wzrastająca liczba filtrów w warstwach konwolucyjnych. Kolejne moduły są przedzielone warstwami stosującymi max-pooling i dropout, które redukują wejściowy rozmiar dla kolejnego modułu o połowę. Wzrasta wtedy dwukrotnie liczba filtrów w warstwie konwolucyjnej.

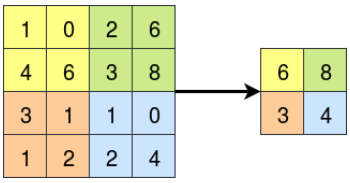


Rysunek 3.6: Wizualizacja zaproponowanej architektury

Zastosowano tu techniki takie jak MaxPooling, Dropout, funkcje aktywacji ReLU i softmax. Zostały one opisane poniżej:

- **max-pooling**
  jest formą kompresji wynikowej mapy cech z warstwy konwolucyjnej, która wydobywa reprezentatywne informacje na zdjęciu. Dzieli wejściowy obraz na regiony i z każdego wybiera maksymalną wartość. Sposób działania obrazuje rysunek 3.7. Pozwala to na zapamiętanie relatywnego umiejscowienia cech na rysunku w odróżnieniu do dokładnej lokalizacji, co redukuje znacznie rozmiar zdjęć przesyłanych dalej

i czas trenowania. Ignoruje mniej znaczące cechy i niewielkie przesunięcia, ponieważ po zastosowaniu maxpoolingu na zdjęciu wynikowa mapa będzie taka sama.



Rysunek 3.7: Działanie maxpoolingu z filtrem (2,2)

- **dropout**
  jest techniką redukującą overfitting poprzez usuwanie niektórych wierzchołków podczas poszczególnych faz trenowania. Prawdopodobieństwo usunięcia wierzchołka jest podawane podczas tworzenia sieci jako parametr, w zaproponowanym przykładzie jest to zaledwie 0.2 w początkowych warstwach, jako że po usunięciu wejściowych wierzchołków informacje zostaną stracone, natomiast w kolejnych warstwach ustalono wartość 0.5.

- **ReLU**
  jest funkcją aktywacji dla każdej warstwy sieci z wyjątkiem ostatniej. Jest postaci:

$$f(x) = \max(0, x) \tag{3.1}$$

  Podczas procesu uczenia, wagi na wierzchołkach są aktualizowane w ramach propagacji wstecznej korzystając z metody gradientu prostego. Dzięki swojej właściwości, takiej że $\lim_{x\to\infty} f(x) \to \infty$ zapobiega problemowi znikającego gradientu [4] (gradient staje się na tyle znikomy, że skutecznie tłumi zmiany wag), co mogłoby zatrzymać proces uczenia. Ponadto funkcja jest prosta obliczeniowo, co skraca czas trenowania.

- **softmax**
  jest funkcją aktywacji w ostatniej warstwie, zwracająca wektor rozkładu prawdopodobieństw dla wszystkich klas. Jest zdefiniowana wzorem:

$$\sigma(z_i) = \frac{\exp(z_i)}{\sum_j \exp(x_j)} \tag{3.2}$$

  Funkcja normalizuje wartości każdego elementu wektora z, zastosowując na nim funkcję eksponencjalną i dzieląc przez sumę eksponent każdego elementu z z. Dzięki temu otrzymane w ten sposób prawdopodobieństwa sumują się do 1.

3.3.2 Z wykorzystaniem wytrenowanego wcześniej modelu

Problem rozpoznawania emocji na twarzy można rozwiązać również za pomocą tzw. przeniesienia uczenia (ang. *transfer learning*). Proces ten polega na wykorzystaniu wiedzy wytrenowanego modelu do rozwiązania innego, pokrewnego mu problemu. Jest to przydatne m. in. dla małego zbioru danych lub niewystarczającej mocy procesora.

W rozpatrywanym przypadku można wykorzystać model VGGFace [6]. Jest oparty na architekturach w trzech wariantach, takich jak: VGG16, Resnet50 lub Senet50. Został wytrenowany na zbiorze danych liczącym ponad 3 miliony zdjęć twarzy. Oryginalnie model został nauczony identyfikować twarze. Jako że opiera się on na konwolucyjnej sieci neuronowej, początkowe warstwy sieci zostały nauczone rozpoznawać prostsze kształty takie jak krawędzie, środkowe warstwy rozpoznają bardziej skomplikowane kształty, ostatnia warstwa jest odpowiedzialna za klasyfikację. Można wykorzystać wiedzę modelu do rozpoznawania bardziej skomplikowanych zależności na twarzy (ang. *feature extraction*) i przenieść ją do identyfikacji emocji. W tym przypadku ostatnia warstwa modelu Resnet50 została pominięta (jak na przykładzie w pseudokodzie 3.1) i

zamieniona na warstwę w pełni połączoną z funkcją aktywacji *softmax*. Z uwagi na mały zbiór danych, które są jednak bardzo zbliżone do tych ze zbioru VGGFace, należy 'zamrozić' aktualizowanie wag warstw modelu, aby uniknąć overfittingu, a trenować jedynie ostatni moduł klasyfikujący, jak opisane na stronie [2].

Pseudokod 3.1: Użycie modelu VGGFace do klasyfikacji emocji

```
1 num_class = 7;
2 model = VGGFace(include_top = False, model='resnet', input_shape=(224, 224, 3));
3 model.add(Dense(num_class, activation='softmax'));
4 foreach layer ∈ model.layers do
5     layer.trainable = False;
```

Wykorzystany model został wyuczony na zdjęciach w kolorystyce RGB o wymiarach (244, 224), dlatego użyte zdjęcia zostały powiększone do takiego rozmiaru.

3.4 Trenowanie modelu

Modele zostają skompilowane przy użyciu wybranej funkcji straty oraz optymalizatora. Następnie cały zbiór danych treningowych oraz danych wygenerowanych na podstawie zdjęć treningowych zostaje poddany procesowi trenowania przy górnej granicy odpowiednio 100 i 10 iteracji. Po każdej iteracji na zbiorze walidacyjnym wyliczana jest funkcja straty oraz dokładność.

Poszczególne parametry zostały opisane w poniższych podrozdziałach.

3.4.1 Optymalizator

Pierwszy model został wytrenowany przy użyciu optymalizatora Adam, natomiast drugi przy użyciu SGD. Wybór tych optymalizatorów został podjęty eksperymentalnie i wybrano te, które zwracały lepsze wyniki.

3.4.2 Funkcja straty

Jako że jest to problem klasyfikacji, w której jest możliwych kilka ($\geqslant 3$) rozłącznych klas, a poprawnym wynikiem jest przyporządkowanie do tylko do jednej z nich, najlepiej sprawdzi się tu kategoryczna entropia krzyżowa (ang. *categorical crossentropy*). Jest zdefiniowana wzorem:

$$L(y, \hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} \cdot \log(\hat{y_{ij}})) \tag{3.3}$$

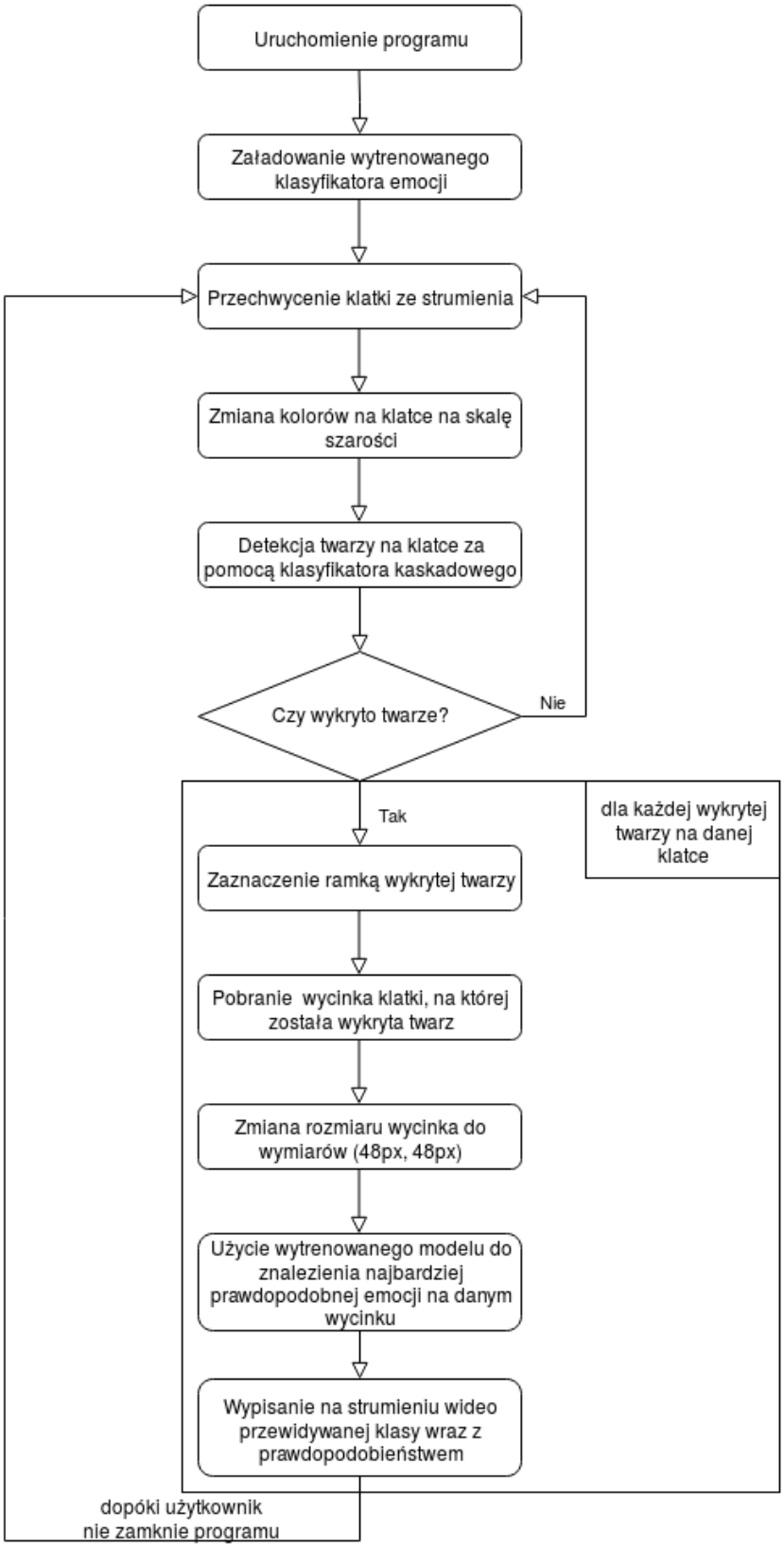
gdzie $\hat{y}$ oznacza predykcję. Funkcja porównuje przewidziany rozkład prawdopodobieństw z prawdziwym wektorem (gdzie np. wektor $[0, 0, 1, 0, 0, 0, 0]$ oznacza należenie do 3. klasy). Im bardziej przewidywane wartości są zbliżone do rzeczywistych, tym mniejsza strata.

3.4.3 Wcześniejsze przerwanie

Podczas procesu trenowania model stale monitoruje wyliczoną na zbiorze walidacyjnym wartość funkcji straty. W przypadku, gdy przez 12 kolejnych iteracji (epochs) funkcja straty nie zanotuje pomniejszenia, zmniejsza się learning rate i trenuje dalej. Modele często zyskują na redukcji learning rate'u w momencie stagnacji. Jeśli przez kolejne 50 iteracji funkcja straty dalej się nie zmniejsza, model kończy trenowanie i zapisywane są wagi z iteracji, która osiągnęła najlepszy wynik.

3.5 Przetwarzanie strumienia wideo

Wytrenowany model został użyty do predykcji emocji na strumieniu wideo na żywo pobieranym z kamerki komputerowej. Działanie programu obrazuje diagram aktywności na rysunku 3.8. Do detekcji twarzy zastosowano open-source'owy klasyfikator kaskadowy [13] [11].



Rysunek 3.8: Diagram aktywności programu

Wyniki

W tym rozdziale zostały opisane wyniki, jakie osiągnęły wytrenowane modele, a także przedstawiono wynik działania programu klasyfikującego emocje w czasie rzeczywistym.

4.1 Ewaluacja modelu

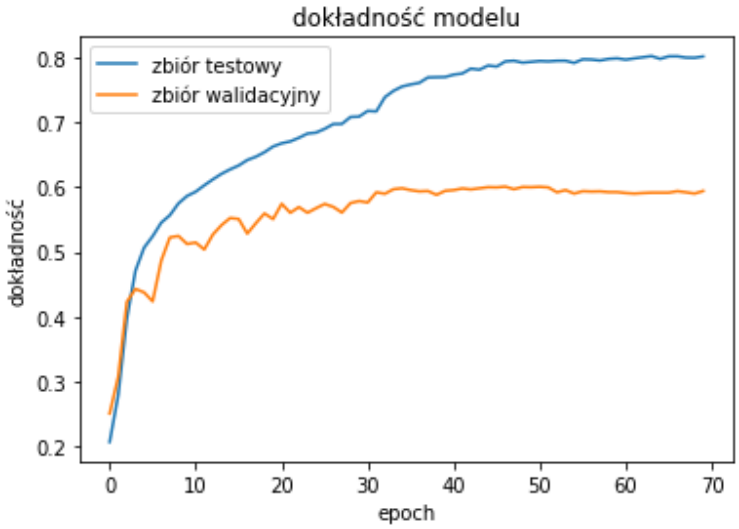
4.1.1 Model od podstaw

Trenowanie pierwszego modelu zostało zakończone po 70. przebiegu całego zbioru danych treningowych przez proces uczenia, ponieważ funkcja straty nie zmalała od 20. przebiegu, pomimo zmniejszania learning rate'u. Otrzymano w ten sposób najlepiej sprawdzający się model przy użyciu wybranych parametrów. Zanotowane w tym momencie wartości straty oraz dokładności przedstawia tabela 4.1.

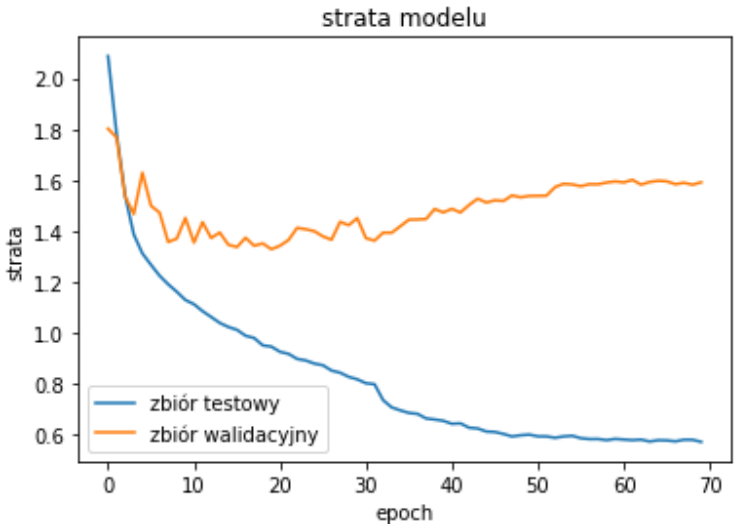
Na zbiorze treningowym	dokładność	65,17%
	strata	0,93
Na zbiorze walidacyjnym	dokładność	58,75%
	strata	1,32

Tablica 4.1: Otrzymana skuteczność modelu

Wykresy na rysunkach 4.1 i 4.2 przedstawiają odpowiednio dokładność oraz funkcję straty w zależności od numeru przebiegu trenowania.



Rysunek 4.1: Dokładność modelu od podstaw



Rysunek 4.2: Strata modelu od podstaw

Wykresy zostały wygenerowane przy pomocy biblioteki Matplotlib [10]. Można zauważyć, że znacząco model ulepszał się do około 20 przebiegu procesu uczenia się całego zbioru, natomiast później aż do zakończenia trenowania oscylował w granicy tego samego wyniku.

Tabela 4.2 pokazuje otrzymaną macierz pomyłek. Można zauważyć, że model często klasyfikuje zdegustowanie lub przerażenie jako złość oraz myli neutralny wyraz twarzy ze smutnym lub szczęśliwym, jednak może to wynikać z za małych różnic widocznych między tymi emocjami, które również człowiek może pomylić.

		Przewidywane						
		Zły	Zdegustowany	Przerażony	Szczęśliwy	Neutralny	Smutny	Zaskoczony
Rzeczywiste	Zły	283	9	44	7	57	80	11
	Zdegustowany	15	23	9	1	0	5	2
	Przerażony	77	2	187	19	61	120	62
	Szczęśliwy	24	1	20	755	42	21	16
	Neutralny	97	9	70	153	131	109	57
	Smutny	60	3	64	31	127	301	8
	Zaskoczony	16	1	54	21	10	9	305

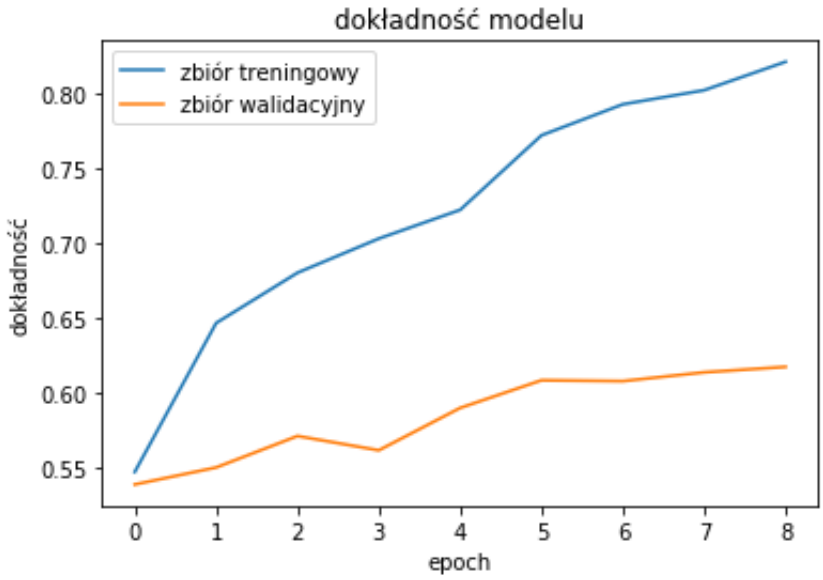
Tablica 4.2: Confusion matrix

4.1.2 Model wcześniej wytrenowany

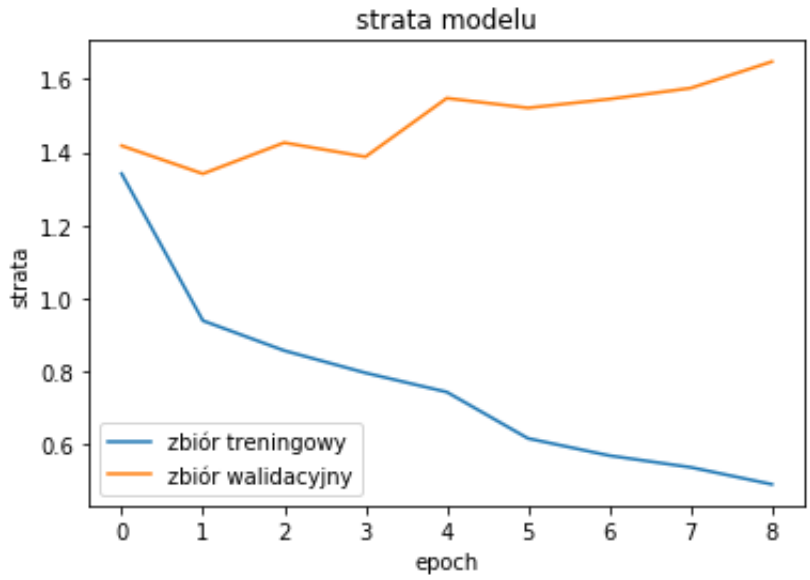
W drugim przypadku model zanotował podobne wyniki co do straty oraz dokładności. Jednak należy zauważyć, że otrzymano taki wynik już po dwóch iteracjach, w odróżnieniu od pierwszego modelu, który potrzebował kilkudziesięciu. Tabela 4.3 przedstawia otrzymaną skuteczność. Wykresy 4.4 i 4.3 przedstawiają, jak zmieniały się te wyniki podczas trenowania. Macierz na rysunku 4.4 przedstawia, ile przypadków model klasyfikował emocje zgodnie z rzeczywistą klasą, a ile klasyfikował do złej.

Na zbiorze treningowym	dokładność	83,44%
	strata	0,53
Na zbiorze walidacyjnym	dokładność	59,65%
	strata	1,34

Tablica 4.3: Otrzymana skuteczność modelu VGGFace



Rysunek 4.3: Dokładność modelu z wykorzystaniem VGGFace



Rysunek 4.4: Strata modelu z wykorzystaniem VGGFace

Z wykresu można odczytać, że od 3. iteracji wartość funkcji straty zaczęła narastać, co może świadczyć o overfittingu modelu.

		Przewidywane						
		Zły	Zdegustowany	Przerażony	Szczęśliwy	Neutralny	Smutny	Zaskoczony
Rzeczywiste	Zły	310	1	71	15	31	59	4
	Zdegustowany	9	40	2	0	1	2	1
	Przerażony	62	2	310	13	43	68	30
	Szczęśliwy	13	0	9	806	29	9	13
	Neutralny	93	11	108	153	97	107	57
	Smutny	57	2	61	25	76	370	3
	Zaskoczony	15	0	29	19	8	4	341

Tablica 4.4: Confusion matrix modelu z wykorzystaniem VGGFace

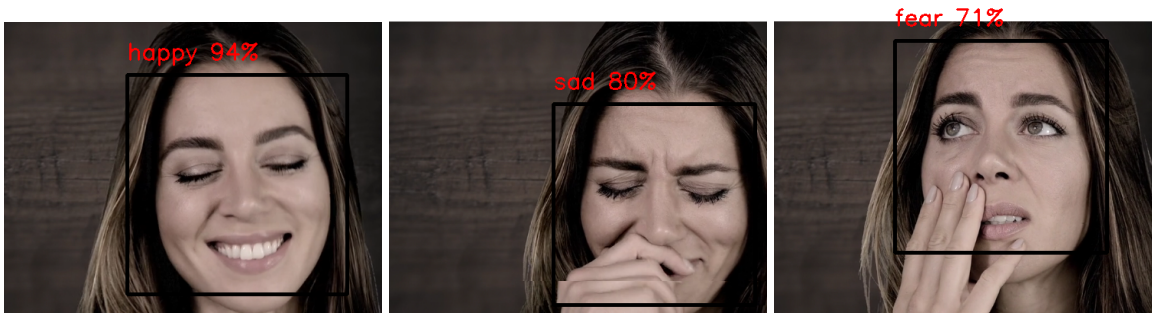
Można zobaczyć, że model osiągnął nieznacznie lepsze wyniki niż w pierwszym przypadku, ale wciąż myli ze sobą negatywne emocje (przerażenie ze złością oraz smutkiem), a także często utożsamia neutralny wyraz twarzy ze szczęściem, smutkiem lub przerażeniem. Jednak, patrząc na zdjęcia znajdujące się w zbiorze danych, widać, że zdjęcia sklasyfikowane jako neutralne często posiadają cechy wspólne zdjęć smutnych czy szczęśliwych wyrazów twarzy (np. lekko przekrzywione w dół lub górę usta).

4.2 Działanie programu

Program działa w dwóch wersjach, jedna przetwarza strumień wideo z kamerki, druga pobiera strumień z ekranu komputera. Obie wersje znajdują twarze i klasyfikują emocje w czasie rzeczywistym wykonywania programu. Zdjęcia na rysunkach 4.5 i 4.6 przedstawiają zrzuty ekranu z działania programu na dwóch przykładowych wideo udostępnionych na portalu Pixabay [3]. Program po wykryciu twarzy obramowuje ją oraz podaje wytrenowanemu modelowi do analizy. Zwracany jest wektor prawdopodobieństw wszystkich klas, a ta najpewniej oszacowana jest wypisywana nad znalezioną twarzą wraz z prawdopodobieństwem.

(a) Zły (b) Zaskoczony

Rysunek 4.5: Działanie programu na 1. wideo



(a) Szczęśliwy (b) Smutny (c) Przestraszony

Rysunek 4.6: Działanie programu na 2. wideo

Implementacja i środowisko

5.1 Opis technologii

Program napisano w języku Python w wersji 3.6, szczegółowy opis można znaleźć w [12]. Do implementacji sieci użyto biblioteki Keras [8], natomiast do przetwarzania strumienia wideo wykorzystano bibliotekę OpenCV [7]. Ponadto, do badania jakości wytrenowanego modelu użyto bibliotek MatPlotLib, Sklearn oraz NumPy.

5.2 Opis środowiska

5.2.1 Środowisko trenowania modelu

Wszystkie czynności związane z procesem trenowania zostały przeprowadzone w serwisie Google Colaboratory, ponieważ dostarcza on bezpłatny dostęp do układów GPU. Takie rozwiązanie znacznie skróciło proces trenowania modelu w porównaniu do pracy z procesorem standardowego laptopa, jak również umożliwiło wielokrotne testowanie sieci w celu zoptymalizowania doboru parametrów. Dodatkowo środowisko to zapewnia wiele zainstalowanych pakietów, w tym użyte w niniejszej pracy język Python, menadżer pakietów PIP oraz biblioteki Tensorflow, Keras i MatPlotLib.

5.2.2 Środowisko uruchomienia programu

Program do przetwarzania emocji z wykorzystaniem gotowego modelu był testowany w systemie operacyjnym Linux w wersji 16.04. Do uruchomienia programu potrzebny jest zainstalowany język Python oraz biblioteki OpenCV, NumPy i Keras.

Podsumowanie

W niniejszej pracy omówiono kluczowe zagadnienia konwolucyjnych sieci neuronowych. Pokazano metody, którymi można posłużyć się, pracując z ograniczoną liczbą danych, takich jak *data augmentation* czy *transfer learning*.

Zaproponowane modele osiągnęły dokładność w granicy 60%, co jest wynikiem o 15% niższym od zwycięskiego modelu *State-of-art* [1], jednak wystarczającym do tego zagadnienia. Można uzyskać lepszą generalizację m. in. poprzez zebranie dużo większej liczby zdjęć dobrze rozdystrybuowanych oraz staranniejszy dobór parametrów przy tworzeniu i trenowaniu sieci. W przypadku *transfer learningu* można tak zmodyfikować pierwsze warstwy wykorzystanego modelu, by przyjmowały na wejściu do trenowania obrazy o rozmiarach (48, 48). Aktualne rozwiązanie ze skalowaniem zdjęć mogło skutkować utratą ważnych cech.

Wytrenowany model może znaleźć zastosowanie głównie w aplikacjach wspomagających urządzenie w roli asystenta oraz w programach sterowanych wyrazem twarzy.

Bibliografia

[1] Facial expression recognition using convolutional neural networks: State of the art. https://arxiv.org/abs/1612.02903.

[2] Keras.blog.io. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

[3] Pixabay. http://pixabay.com/.

[4] Towards non-saturating recurrent units for modelling long-term dependencies. https://arxiv.org/pdf/1902.06704.pdf.

[5] www.cs.toronto.edu. https://www.cs.toronto.edu/~frossard/post/vgg16/.

[6] Vgg face descriptor. http://www.robots.ox.ac.uk/~vgg/software/vgg˙face/, 2015.

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[8] F. Chollet, i in. Keras. https://keras.io, 2015.

[9] I. Goodfellow, D. Erhan, P.-L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, Y. Bengio. Challenges in representation learning: A report on three machine learning contests, 2013.

[10] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[11] R. Lienhart, J. Maydt. An extended set of haar-like features for rapid object detection. wolumen 1, 09 2002.

[12] G. Van Rossum, F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Paramount, CA, 2009.

[13] P. Viola, M. Jones. Rapid object detection using a boosted cascade of simple features. wolumen 1, strony I–511, 02 2001.

Zawartość płyty CD

Poniżej przedstawiono strukturę katalogów wraz z ich zawartością znajdujących się na płycie CD dodanej jako załącznik do niniejszej pracy inżynierskiej.

Folder `weights` zawiera wagi wytrenowanych modeli. Folder `test` zawiera programy przetwarzające strumienie wideo - pobierane z kamery lub ekranu. Plik `haarcascade.xml` zawiera użyty do detekcji twarzy open-source'owy klasyfikator kaskadowy [11]. Folder `src` zawiera kody źródłowe do wytrenowania modeli.

```
/
├── weights
│   ├── vggface.h5
│   └── fromscratch.h5
├── test
│   ├── camerastream.py
│   ├── screenstream.py
│   └── haarcascade.xml
└── src
    ├── fromscratch.py
    └── vggfacemodel.py
```