UNIWERSYTET WSB MERITO W GDAŃSKU

WYDZIAŁ INFORMATYKI I NOWYCH   
TECHNOLOGII

**ANALIZA PORÓWNAWCZA KONWOLUCYJNYCH SIECI NEURONOWYCH INCEPTION RESNET V2 I XCEPTION W ZASTOSOWANIU ICH DO KATEGORYZOWANIA ZMIAN SKÓRNYCH**

Praca inżynierska

na kierunku Informatyka

Praca napisana pod kierunkiem

dr inż. Pawła Tomkiewicza

Marek Małek, 56572, Gdańsk 2024

Spis treści

[WSTĘP 2](#_Toc154851582)

[Rozdział 1 – Historia Sieci Neuronowych 4](#_Toc154851583)

[1.1 Wprowadzenie 4](#_Toc154851584)

[1.2 Konwolucyjne Sieci Neuronowe (CNN) 5](#_Toc154851585)

[1.3 Mechanizm propagacji wstecznej 6](#_Toc154851586)

[1.4 Operacja splotu 7](#_Toc154851587)

[Rozdział 2 – Architektury InceptionResNetV2 i Xception 8](#_Toc154851588)

[2.1 Architektura InceptionResNetV2 8](#_Toc154851590)

[2.2 Architektura Xception 9](#_Toc154851591)

[Rozdział 3 – Proces Trenowania Modeli 11](#_Toc154851592)

[3.1 Zbiór danych i EDA 11](#_Toc154851594)

[3.2 Trenowanie i rezultaty modeli bazowych 12](#_Toc154851595)

[3.3 Opis procesu uruchamiania treningów 14](#_Toc154851596)

[3.4 Trenowanie sieci neuronowych 16](#_Toc154851597)

[3.5 Usprawnianie sieci neuronowych 20](#_Toc154851598)

[3.6 Porównanie wyników 20](#_Toc154851599)

[Rozdział 4 – Statystyczna Analiza Porównawcza 21](#_Toc154851600)

[4.1 Opis procesu wyboru najlepszej instancji InceptionResNetV2 21](#_Toc154851602)

[4.2 Opis procesu wyboru najlepszej instancji Xception 21](#_Toc154851603)

[4.3 Porównanie wyników 21](#_Toc154851604)

[BIBLIOGRAFIA 22](#_Toc154851605)

[Opracowania książkowe 22](#_Toc154851606)

[Netografia 22](#_Toc154851607)

[SPIS RYSUNKÓW 23](#_Toc154851608)

# WSTĘP

Celem pracy jest wykonanie analizy porównawczej dwóch, klasycznych już architektur konwolucyjnych sieci neuronowych, którymi są InceptionResNetV2 oraz Xception. Obie te architektury służyć będą za rdzeń kilkunastu wariacji mających na celu osiągnięcie wyników lepszych, niż byłoby to możliwe przy użyciu niezmodyfikowanych sieci i surowych danych. W jej skład wchodzić będą następujące części:

* Krótka, eksploracyjna analiza danych (en. EDA[[1]](#footnote-1)) – jej celem jest sprawdzenie rozkładu klas obrazów
* Wyszukiwanie modelu bazowego – aby wyraźnie zobaczyć przewagę sieci neuronowych w danym zagadnieniu, dobrze jest najpierw zastosować prostsze i szeroko znane techniki oraz sprawdzić jaką one mają wydajność
* Sprawdzenie jak dobrze na zestawie danych HAM10000[[2]](#footnote-2) działa architektura InceptionResNetV2[[3]](#footnote-3), na razie ignorując dysbalans klas; sprawdzenie, czy będzie w stanie pobić model bazowy
* Dostrajanie wybranej architektury tak, aby lepiej radziła sobie na zbiorze danych HAM10000
* Wytrenowanie na tym samym zestawie klas architektury Xception[[4]](#footnote-4) i porównanie jej wyników z InceptionResNetV2

Poza wymienionymi składowymi, wykonane zostaną także skrypty ułatwiające zarządzanie kodem oraz automatyzujące wielokrotne uruchamianie procesów treningowych. Jest to potrzebne, aby móc wyciągnąć poprawne wnioski, ponieważ w szkoleniu każdej sieci neuronowej jest miejsce na element losowości. Stąd dla zrozumienia, która modyfikacja miała pozytywny wpływ na trening potrzebne jest kilkukrotne uruchomienie go oraz uśrednienie wyników, a następnie analiza statystyczna.

W rozdziale pierwszym i drugim została zawarta część teoretyczna pracy. Opisano w nich krótko historię sieci neuronowych, w szczególności splotowych sieci neuronowych (CNN[[5]](#footnote-5)), przybliżono w nich informacje dotyczące technik, które sprawiły że są tak skuteczne w rozpoznawaniu obrazów, a także opisano dwie wybrane architektury porównując je w miejscach, w których najmocniej się różnią.

Na początku rozdziału trzeciego opisano proces i skutki trenowania modeli bazowych, które ostatecznie posłużyły jako punkt wyjścia do poszukiwań właściwych architektur neuronowych. Następnie opisano modyfikacje mające usprawnić sieci InceptionResNetV2 oraz Xception i analogicznie do początku rozdziału – opisano proces i skutki trenowania tych modeli.

W rozdziale czwartym przedstawiono proces analizy statystycznej stanowiącej sposób dojścia do opisanych dalej wniosków. Opisano tu też problemy, z którymi spotkano się w trakcie prac.

# Rozdział 1 – Historia Sieci Neuronowych

## Wprowadzenie

Większość wytwarzanego oprogramowania od samego początku była algorytmiczna. Program miał zawierać procesy i reguły biznesowe danego przedsiębiorstwa i w ten sposób automatyzować powtarzalne czynności, które można przelać na kod w postaci szeregu instrukcji. Oprogramowanie takie zawsze jest specyficzne, rzadziej dla całej domeny, w której porusza się firma, częściej dla konkretnego jej wycinku, specyfiki działania samego przedsiębiorstwa. Oczywiście istnieją gotowe produkty, takie jak SAP ERP[[6]](#footnote-6), które operują w obrębie całych domen, jednak wciąż ich zasada działania jest taka, jak mniejszych programów – do aplikacji trafiają dane zewnętrzne, które po przetworzeniu przez ściśle określone reguły biznesowe dają spodziewany rezultat. Od zawsze istniała jednak potrzeba istnienia oprogramowania posługującego się logiką rozmytą[[7]](#footnote-7) tak, aby dla pewnych klas problemów nie trzeba było tworzyć tak sztywnych i dokładnych reguł przetwarzania danych. W istocie, być może dla większości tych klas problemów nie byłoby to nawet możliwe. Dla przykładu, w praktyce niemożliwe jest napisanie dobrego klasyfikatora pisma odręcznego.

Różne algorytmy uczenia maszynowego znane były od połowy ubiegłego stulecia, a wciąż krótką historię samych sieci neuronowych podzielić można na kilka okresów:

* Wczesne koncepcje lat 40-tych i 50-tych: wtedy to Warren McCulloch i Walter Pitts tworzą pierwszy „obliczeniowy” model biologicznego neuronu[[8]](#footnote-8).
* Wynalezienie Perceptronu[[9]](#footnote-9) w późnych latach 50-tych: była to wczesna sieć neuronowa zdolna nauczyć się rozpoznawania prostych wzorców, miała jednak spore ograniczenia, np. nie była w stanie rozwiązywać problemów, które nie były liniowo rozdzielne (np. problem XOR[[10]](#footnote-10))
* „Pierwsza Zima Sztucznej Inteligencji” w późnych latach 60-tych i 70-tych: przez ograniczenia perceptronu oraz niewielkiej dostępnej wtedy badaczom mocy obliczeniowej zainteresowanie inwestorów mocno ucierpiały przez co rozwój AI bardzo spowolnił.
* Wynalezienie algorytmu Propagacji Wstecznej[[11]](#footnote-11): umożliwiło to sieciom neuronowym uczenie się bardziej skomplikowanych wzorców.
* „Druga Zima Sztucznej Inteligencji” w późnych latach 80-tych i 90-tych: nastała z powodu zbyt dużych oczekiwań i ograniczonej dostępności mocy obliczeniowej. Innymi słowy sprzęt komputerowy wciąż był zbyt słaby, aby sieci neuronowe mogły pokazać swoje prawdziwe możliwości. Wtedy też na popularności zyskały mechanizmy uczenia maszynowego niezwiązane z sieciami neuronowymi, takie jak Support Vector Machines[[12]](#footnote-12)
* Ponowna faza wzrostu zainteresowania AI oraz czas przełomów: od początku lat 2000 ponownie widać było rosnące zainteresowanie sieciami neuronowymi przez badaczy i wielkie firmy technologiczne. W okolicach początku drugiego dziesięciolecia lat dwutysięcznych zaczęły się pojawiać kolejne przełomowe odkrycia i czas ten trwa do teraz. Kilka największych z nich to np. AlexNet[[13]](#footnote-13), która to sieć stanowiła przełom w dziedzinie rozpoznawania obrazów, odkrycie architektur GAN[[14]](#footnote-14) oraz Transformer[[15]](#footnote-15) (która to jest używana np. w święcącym sukcesy ChatGPT).

## Konwolucyjne Sieci Neuronowe (CNN)

Chociaż historia rozpoznawania obrazów przy pomocy sieci neuronowych, przynajmniej na poziomie koncepcji, sięga lat 80-tych, to pierwszym najgłośniejszym zastosowaniem tego typu sieci była architektura LeNet-5, stworzona w późnych latach 90-tych przez obecnie szeroko znanego badacza sztucznej inteligencji – Yanna LeCuna. Celem jej stworzenia było umożliwienie Amerykańskiemu Biuru Pocztowemu automatycznego odczytu kodów pocztowych z kopert. Była to dość prosta (z dzisiejszego punktu widzenia) architektura, której składowe to warstwa wejścia o wymiarach 32x32x1 (gdzie pierwsze dwie liczby to wysokość i szerokość obrazów, a ostatnia to ilość kanałów), warstwa splotowa, warstwa uśredniająca, jeszcze jedna warstwa splotowa, po niej uśredniająca i ostatnia splotowa, po której znajdują się dwie warstwy gęsto połączone oraz warstwa wyjścia, służąca do klasyfikacji liczb. Warstwy splotowe wykrywają wzorce przestrzenne, natomiast warstwy uśredniające zmniejszają rozmiar danych pochodzących z ich poprzedników. Zostało to zilustrowane na Rysunku 1.

Rysunek 1 – Architektura Sieci LeNet-5

Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

Źródło: https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

Architektura ta stała się punktem wyjścia dla twórców bardziej złożonych sieci neuronowych takich jak AlexNet, VGGNet, czy ResNet.

Być może największym wkładem Yanna LeCuna w rozwój sztucznej inteligencji było zastosowanie mechanizmu tzw. Propagacji Wstecznej. Nie była to całkowicie nowa idea, jednak była świeża w świecie wielowarstwowych modeli konwolucyjnych. Jako, że jest to technika leżąca u podstaw każdego rodzaju sieci, nie tylko CNN, zostanie ona opisana jako pierwsza.

## Mechanizm propagacji wstecznej

Mechanizm ten składa się z kilku kroków:

* Przejście wprzód: warstwa po warstwie, aż do wyjścia przekazywane są dane. W przypadku sieci konwolucyjnej będą to coraz bardziej zmienione – zmniejszone, a na końcu spłaszczone do postaci wektora, dane obrazu.
* Obliczenie straty: funkcja straty (która jest jednym z tzw. hiperparametrów) oblicza różnicę między predykcją modelu, a stanem faktycznym – tym, co chciano by, aby model zwrócił.
* Przejście w tył (propagacja wsteczna): w tym miejscu algorytm oblicza gradienty funkcji straty względem każdej z wag w sieci. Każda z owych wag przyczynia się do uzyskania ostatecznego rezultatu, dlatego też sygnał błędu musi zostać rozpropagowany wstecz.
* Gradienty używane są do aktualizacji wag sieci.
* Cały cykl powtarza się na nowo, w każdym przejściu minimalizując wynik funkcji straty.

## Operacja splotu

Tak jak mechanizm propagacji wstecznej jest kluczowym elementem dla wszystkich rodzajów sieci neuronowych, to operacja splotu jest takim elementem w sieciach konwolucyjnych. Dzięki niej, sieci splotowe są w stanie lepiej, od np. sieci gęsto połączonych wychwytywać złożone wzorce, ale też, do pewnego stopnia są w stanie znajdywać je w różnych miejscach obrazów wejściowych. Składowymi operacji splotu są:

* Filtry (lub kernele): są to małe macierze wag, które podobnie jak wagi warstw gęstych ustawione są początkowo na losowe wartości.
* „Aplikowanie” filtrów na danych wejściowych: w tym momencie każda z wag mnożona jest przez wartość danego piksela. Kiedy mnożenia na obecnej pozycji filtra zostają zakończone, ich wyniki są do siebie dodawane – wynik tych operacji to wynik działania filtra na bieżącym obszarze obrazu.
* „Przesuwanie” filtrów: w tym momencie następuje przesunięcie kerneli o określoną ilość pikseli, aż do „końca” obrazu.
* Wyjściem z tej operacji są tzw. mapy cech, które reprezentują to, co w danym przejściu udało się wykryć warstwie splotowej.
* Na końcu, na każdej mapie cech wywoływana jest nieliniowa funkcja aktywacji (np. ReLU), po to, żeby model był zdolny wychwytywać skomplikowane wzorce.

# Rozdział 2 – Architektury InceptionResNetV2 i Xception



## Architektura InceptionResNetV2

Największą zmianą wprowadzoną przez autorów tej architektury w stosunku do jej poprzedników jest użycie modułów incepcyjnych oraz połączeń skrótowych (residual connections). Obie te techniki istniały już wcześniej, jednak użycie ich kombinacji umożliwiło utworzenie dużo głębszych struktur, które oprócz cechowania się wyższą dokładnością, mniej cierpiały z powodu problemu tzw. zanikającego gradientu.

Problem ten został zauważony przez badaczy, kiedy próbowali uczyć bardzo głębokie sieci. Opisany wcześniej mechanizm propagacji wstecznej może cechować się tym, że wagi warstw położonych najdalej są w procesie treningu dostosowywane w najmniejszym stopniu, co z kolei skutkuje tym, że warstwy te przestają się uczyć i sieć przykłada nieproporcjonalnie wyższą uwagę to położonych wyżej. To z kolei sprawia, że stosowanie głębokich sieci neuronowych może nie przynieść żadnych korzyści, lub wręcz sprawić, że ich dokładność będzie niższa od ich prostszych rodzajów. W poszukiwaniu architektur zdolnych rozpoznawać coraz bardziej skomplikowane wzorce, narodziła się konieczność zapobieżenia temu zjawisku. Połączenia skrótowe (pomijające) działają tak, że warstwa poprzedzająca warstwę bezpośrednio nad nią, zwraca swój rezultat do warstwy położonej jeszcze wyżej – oba wyniki są dodawane i w ten sposób mechanizm propagacji wstecznej może w bardziej wydajny sposób rozpropagować gradient funkcji straty.

Jeśli zaś chodzi o ideę, która stała za zastosowaniem modułów incepcyjnych, to badaczom chodziło o to, aby móc wykrywać skomplikowane wzorce w różnych skalach. Stąd, moduły incepcyjne przeprowadzają operację splotu używając jednocześnie filtrów o różnych wielkościach (np. 5x5, 3x3).

Na rysunku 2 zaprezentowane zostały oba omawiane moduły. Jest to bardzo podstawowa prezentacja tej idei, a w jej konkretnych implementacjach występują różne bloki incepcyjne oraz pomijające, bardziej rozbudowane od poniższych.

Po lewej stronie widać, że blok incepcyjny rozgałęzia się na kilka „ramion” przechodzących przez dane filtrami różnych wielkości. Na końcu wyniki tych równoległych operacji łączone są do postaci jednej mapy cech.

Rysunek 2 - przykład modułu incepcyjnego

Obraz zawierający tekst, diagram, linia, zrzut ekranu

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, linia, Czcionka

Opis wygenerowany automatycznie

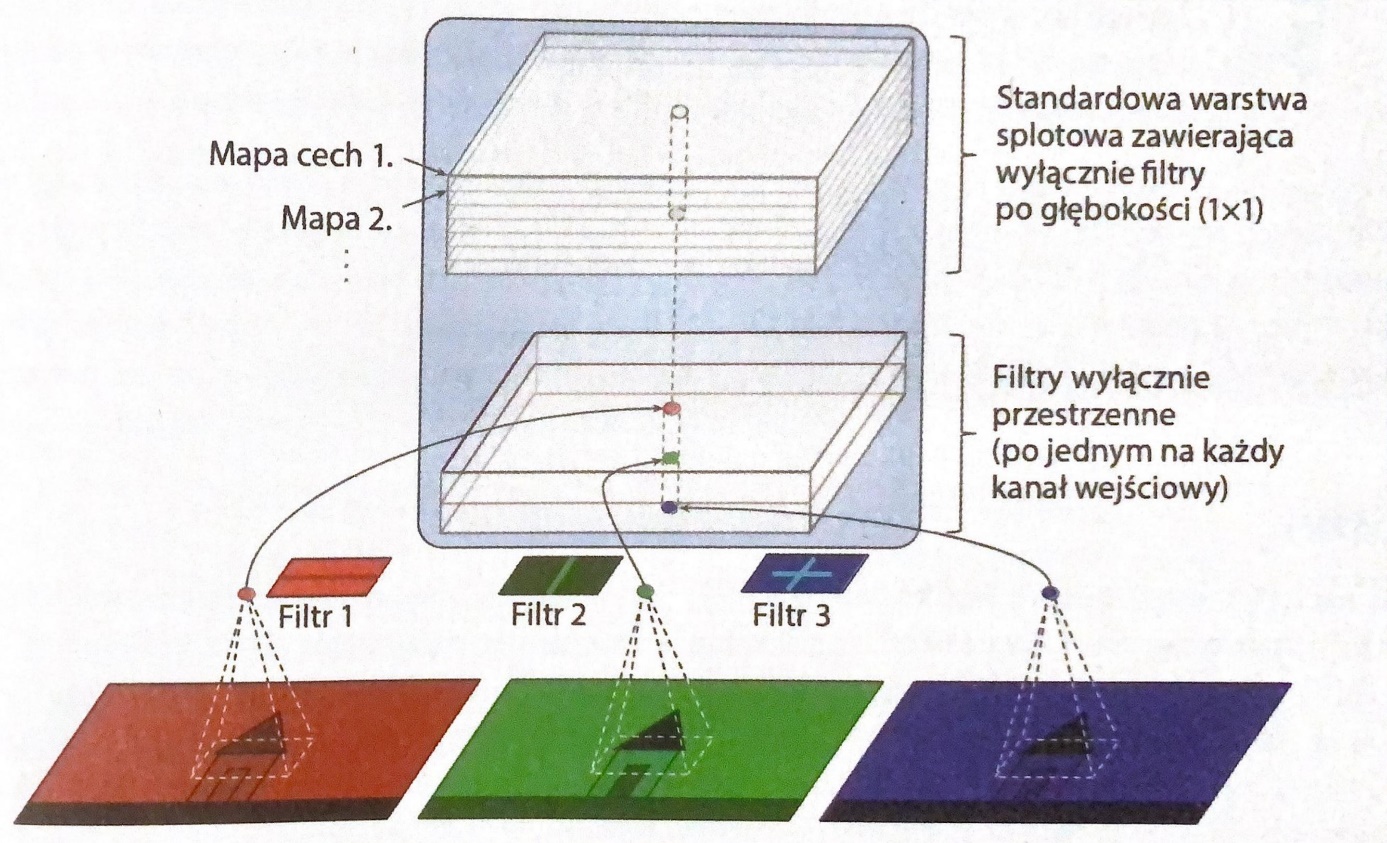
Źródło: https://www.researchgate.net/figure/Basic-architecture-of-inception-block\_fig4\_330511306  
Źródło: https://arxiv.org/pdf/1602.07261v2.pdf? https://arxiv.org/pdf/1602.07261v2.pdf?

Po prawej stronie widać schemat bloku zawierającego połączenie pomijające. W tym przykładzie, dla uproszczenia użyto dwóch warstw splotowych, jednak nic nie stoi na przeszkodzie, aby w ich miejscu pojawił się blok incepcyjny taki, jak na schemacie z lewej strony, co też z resztą ma miejsce w kodzie modelu InceptionResNetV2.

## Architektura Xception

Architektura Xception została zaproponowana jako wariacja podejścia prezentowanego przez Inception, z tą różnicą, że bloki incepcyjne zastąpione zostały rozdzielnymi po głębokości warstwami splotowymi. Klasyczna sieć splotowa wykorzystuje filtry starające się jednocześnie wykrywać wzorce przestrzenne i międzykanałowe, natomiast w rozdzielnej warstwie splotowej użyto założenia, że wzorce przestrzenne i międzykanałowe mogą być modelowane oddzielnie. Między tą architekturą a kolejnymi podejściami do pomysłów zawartych w linii modeli Inception istnieje też sporo innych różnic, jednak ta jest kluczowa i największa. Dzięki użyciu operacji splotu w opisany powyżej, alternatywny sposób, Xception uznawane jest za generalnie skuteczniejszą architekturę. Rysunek 3 zawiera diagram kluczowego elementu:

Rysunek 3- wykorzystanie filtrów po głębokości



Źródło: Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow

# Rozdział 3 – Proces Trenowania Modeli



## Zbiór danych i EDA

Jako zbiór danych wybrano dostępny w portalu kaggle.com HAM10000. Zawiera on 10 000 obrazów zmian skórnych dla 7 kategorii. Oprócz samych zdjęć, obecne są także pewne informacje o pacjentach, takie jak wiek, płeć, umiejscowienie zmiany i metoda jej wykrycia. Ostatecznie nie zdecydowano się na ich użycie, aby lepiej skupić się na rdzeniu rozpatrywanego problemu, czyli rozpoznawaniu obrazów. Rysunek 4 prezentuje histogramy zliczające wystąpienia określonych zmian skórnych wśród płci, gdzie na niebiesko zaznaczono mężczyzn, a na pomarańczowo kobiety:

Rysunek 4 - EDA

Obraz zawierający tekst, zrzut ekranu, diagram, Wykres

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Ciekawą obserwację być może stanowi fakt, że dla większości zmian skórnych obecnych w zbiorze HAM 10000 ich ofiarami nieco częściej padają mężczyźni. Nie ma to jednak wpływu na efektywność treningu, wybranych architektur.

## Trenowanie i rezultaty modeli bazowych

Przez modele bazowe, w kontekście tej pracy należy rozumieć modele niewykorzystujące sieci neuronowych. Wybrano to podejście, aby unaocznić, że prostsze algorytmy uczenia maszynowego nie są skuteczne w obliczu zbioru danych tak dużego i skomplikowanego jak HAM 10000. Chociaż były w stanie osiągnąć zbieżność szybciej od sieci neuronowych, to ich dokładność nigdy nie przekroczyła 41%.

Pierwszy z wybranych algorytmów to KNeighborsClassifier, obecny w pythonowej bibliotece scikit-learn (w skrócie sklearn). W świecie data science jest on dość popularny i często wybierany jest jako klasyfikator nawet dla bardziej złożonych problemów – przez swoją skuteczność i prostotę. Rysunek 5 pokazuje punkt wyjściowy, czyli trening tego klasyfikatora bez żadnych modyfikacji:

Rysunek 5 - pierwsze podejście do treningu KNN

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Dokładność lekko poniżej 41% była niezadawalająca, dlatego też spróbowano usprawnić ten wynik dodając do procesu treningu przeszukiwanie siatki. Polega ono na tym, że interfejs sklearn pozwala użytkownikowi na stworzenie tzw. pipeline, w którym to kolejne kroki uruchamiane są z różnymi kombinacjami parametrów, w ten sposób próbując przetestować każdą taką kombinację i na końcu wybierając najlepszą z nich. Najpopularniejszym podejściem używanym przy treningu modeli dostarczanych przez sklearn jest użycie PCA. Jest to technika redukcji wymiarowości, która powinna nieco uprościć (lub odszumić) dane, dekorelując je i zmniejszając liczbę ich wymiarów. Rysunek 6 pokazuje skutki treningu z użyciem PCA:

Rysunek 6 - trenowanie KNN z pomocą PCA

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Jak widać, nastąpiła nieznaczna poprawa, wciąż jednak było to zbyt mało. Jako kolejny klasyfikator wybrano SGD, również z biblioteki sklearn. Występują między nimi zasadnicze różnice, które da się streścić w słowach. KNeighborsClassifier może działać dobrze w sytuacji, kiedy granice decyzyjne są nietrywialne (a więc zbiór danych jest złożony), ale gorzej radzi sobie z dużymi zbiorami danych i ich wysoką wymiarowością, natomiast SGD lepiej radzi sobie wieloma danymi, ale wymaga większej dozy strojenia hiperparametrów. Efekty szkolenia drugiego z nich widoczne są na rysunku 5:

Rysunek 7 - trenowanie SGD

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Jest to najlepszy wynik, który udało się osiągnąć przy użyciu tego klasyfikatora, jednocześnie jak widać gorszy nawet od uzyskanego przez KNeighborsClassifier.

## Opis procesu uruchamiania treningów

Trening kolejnych wariacji obu wybranych modeli odbywał się na komputerze osobistym o dobrych parametrach. Aby otrzymać statystycznie istotne wyniki, każdą z wariacji uruchomiono po 20 razy. Z jednej strony pozwoliło to mimo pewnej losowości wyników odnaleźć najsilniejszą architekturę, a z drugiej najsilniejszą jej instancję. Niestety, takie podejście poskutkowało bardzo długotrwałym treningiem, który trwał około 2 miesięcy. Z tego też powodu należało napisać skrypt pozwalający zarządzać uruchamianiem treningów, oraz zdolny je przerywać w przewidywalny sposób nie skutkujący błędami w zapisanych modelach. Każdy z nich umieszczony został w osobnym notatniku Jupyter, co ułatwiło zarządzanie, modyfikacje oraz prostą, bazującą na wykresie dokładności analizę ostatniego wyniku. Na rysunkach 8 i 9 widać kompletny kod skryptu zarządzającego uruchomieniami, natomiast kolejne podrozdziały prezentują kod kolejnych sieci neurunowych.

Rysunek 8 - pierwsza część skryptu uruchamiającego

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Po zaimportowaniu kilku wbudowanych bibliotek, oraz kilku funkcji pomocniczych, skrypt odczytuje argumenty linii poleceń takie jak:

* typ (InceptionResNetV2, lub Xception)
* żądana ilość uruchomień każdego modelu
* nazwa pliku, którego pojawienie się w folderze użytkownika oznacza konieczność zakończenia treningu

W dalszej części odbywa się odczyt zapisanych wcześniej ilości uruchomień każdego modelu. Dzieje się to dlatego, że jeśli trening został przerwany, nie chciano, aby po wznowieniu startował od uruchomienia zerowego, ale od tego, na którym ostatnio skończono.

Rysunek 9 - druga część skryptu uruchamiającego

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Następną częścią skryptu uruchamiającego jest pętla, która iterując po kolejnych notatnikach, uruchamia je zadaną ilość razy, w każdej iteracji sprawdzając, czy zażądano zakończenia treningu. Widać tu też, że mierzony jest czas wykonania każdego notatnika, tak, aby pokazać w linii komend szacowany czas zakończenia treningu całości.

## Trenowanie sieci neuronowych

Jako, że budowanie własnych architektur sieci neuronowych jest bardzo trudne, zdecydowano się na użycie sugerowanego przez wiele podręczników alternatywnego podejścia – skorzystano z tzw. transfer learning. Oznacza to, że użyte zostały gotowe modele, udostępniane przez bibliotekę TensorFlow. Następnie starano się użyć samodzielnie zbudowanych modułów, mających usprawnić działanie modeli bazowych oraz w określony sposób nadawano wartości pewnym istotnym dla procesu treningu parametrom modeli.

Rysunek 10 pokazuje strukturę folderów projektu. Najbardziej interesująca jest rozwinięta część, ponieważ pokazuje nazwy plików wskazujące na to, jakie modyfikacje bazowych architektur poczyniono. Widać tu też dwukrotną przewagę ilości modeli architektury InceptionResNetV2 (w skrócie: Inception) Xception. Różnica ta miała dwa powody:

* Oszczędność czasu – trenowanie modeli Xception było porównywalnie wolne do treningu modeli Inception, stąd dołożenie kolejnych 6 wariacji skutkowałoby kolejnym miesiącem treningu.
* Xception, jako nowsza architektura, lepiej sprawdzająca się w konkursach klasyfikacji obrazów powinien pobić architekturę Inception nawet bez żadnych usprawnień. Postanowiono to sprawdzić.

Rysunek 10 - struktura projektu

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Wariacje architektury InceptionResNetV2 znajdują się w kolejnych notatnikach, których wyjściowymi wersjami są cztery opisane niżej:

1. whole\_model\_trainable – jest to wariacja wyjściowa, bez modyfikacji, ani ustawionych parametrów początkowych. Żadne warstwy nie zostały w tym przypadku zamrożone, a więc w procesie treningu każda waga może zostać zmodyfikowana.
2. only\_bottom\_half\_layers\_trainable – jw. z tą różnicą, że model jest zamrożony „od połowy”. Górne warstwy sieci neuronowej znają reprezentacje bardziej wysokopoziomowych cech, takich jak różnego rodzaju linie i krzywe, natomiast dolne – ich składowe. Zamrożenie modelu „od połowy” miało poskutkować tym, że zawarta w modelu wysokopoziomowa wiedza nie zostanie naruszona, zmienią się natomiast składowe kierujące model do określonych wniosków.
3. whole\_model\_trainable\_with\_attention\_module\_on\_top – moduł uwagi zostanie opisany w kolejnym podrozdziale, oprócz niego model jest identyczny z pierwszym.
4. only\_bottom\_half\_layers\_trainable\_with\_attention\_module\_on\_top – taki sam jak model drugi, prócz dodanego modułu uwagi.

Dalsze notatniki do powyższych dodają kolejno: odpowiadającą rozkładowi klas inicjalizację wag, wagi klas oraz technikę augmentacji danych znaną pod nazwą centrowania próbki. Każda z nich zostanie opisana w podrozdziale dotyczącym poczynionych usprawnień.

Kolejny interesujący folder to /plots, gdzie notatniki Jupyter składowały wykresy wygenerowane na koniec treningu każdej wariacji. Służyły one wizualnej inspekcji wyników, ale nie brały udziału w ostatecznej analizie, która była czysto statystyczna. W folderach /src/constants/ /src/functions/ oraz /src/image\_manipulation umieszczone zostały fragment kodu, z których notatniki później korzystały.

Wszystkie notatniki posiadały identyczną strukturę, którą widać na rysunku 11. Najpierw importowane są potrzebne biblioteki systemowe, oraz własne, następnie nadawane są wartości stałym, które używane są na końcu przez dołączoną w pierwszym kroku funkcję run\_model. Funkcja ta jest sercem każdego notatnika, ponieważ to ona definiuje kolejne kroki treningu, a jej interfejs pozwala na przekazanie do niej tzw. fabryk modeli – jako że funkcja ta używana jest wszędzie, a każdy notatnik reprezentuje inny model, to musiała zostać napisana w ogólny sposób, czyli tak, aby mogła pracować na każdym rodzaju kombinacji architektura/wariacja. Rysunek 11 zawiera tylko jej wycinek, ponieważ kod jest dość długi.

Rysunek 11 - struktura notatnika Jupyter

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Źródło: opracowanie własne

Rysunek 12 - ciało funkcji run\_model

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Źródło: opracowanie własne

## Usprawnianie sieci neuronowych

Jak zostanie pokazane w rozdziale opisującym analizę porównawczą, chociaż między modelami bez dużych usprawnień, a takimi, które te usprawnienia zawierają nie było wielkich różnic, to statystycznie dowiedzione zostało, że proponowane usprawnienia miały rezultat w postaci wyższej dokładności zmienionych modeli. W przypadku medycyny jest to bardzo istotne, ponieważ wzrost skuteczności modelu o 1% może oznaczać ocalenie dodatkowych setek tysięcy istnień ludzkich.

Pierwszym takim usprawnieniem było dodanie modułu uwagi. Pomysł na niego zainspirowany został architekturą sieci SENet. W tym modelu moduły uwagi mają nieco inną nazwę – Squeeze and Excitation – oraz wbudowane są wewnątrz sieci, natomiast w rozważanym przykładzie taki moduł został dodany między wyjście z architektury InceptionResNetV2, a warstwy klasyfikatora. Kolejne kroki działania tego mechanizmu to:

1. Użycie warstwy GlobalAveragePooling2D interfejsu Keras oraz następująca po niej operacja Reshape: warstwa GAP oblicza średnią każdej mapy cech, redukując jej wymiary do jednej liczby, zachowując jednak wymiar głębokości, a więc wynikiem działania takiej warstwy jest tensor 1x1xGŁĘBOKOŚĆ – jego wartość można zinterpretować jako skompresowaną reprezentację danych wejściowych; taką, która pozwala mocniej wybić się globalnym cechom. W terminologii architektury SENet jest to etap nazwany „squeeze”.
2. Użycie dwóch warstw gęsto połączonych: celem pierwszej jest jedynie zwiększenie mocy poznawczej sieci, natomiast druga, poprzez użycie funkcji softmax, zamienia wyuczone w dwóch wcześniejszych krokach reprezentacje na wektor wag zawierający wartości w przedziale [0, 1]. W terminologii SENet jest to krok nazwany „excitation”.
3. Mapy cech dostarczone modułowi uwagi mnożone są przez wynik działania drugiej warstwy Dense. W tym miejscu następuje rekalibracja tych map cech. Dzięki niej, sieć może uwypuklić istotne mapy cech i wytłumić mniej istotne.

## Porównanie wyników

# Rozdział 4 – Statystyczna Analiza Porównawcza



## Opis procesu wyboru najlepszej instancji InceptionResNetV2

## Opis procesu wyboru najlepszej instancji Xception

## Porównanie wyników

# BIBLIOGRAFIA

## Opracowania książkowe

## Netografia

# SPIS RYSUNKÓW

Rysunek 1 – Architektura Sieci LeNet-5 6

Rysunek 2 - przykład modułu incepcyjnego 9

Rysunek 3- wykorzystanie filtrów po głębokości 10

Rysunek 4 - EDA 11

Rysunek 5 - pierwsze podejście do treningu KNN 12

Rysunek 6 - trenowanie KNN z pomocą PCA 13

Rysunek 7 - trenowanie SGD 14

1. https://en.wikipedia.org/wiki/Exploratory\_data\_analysis [↑](#footnote-ref-1)
2. https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000 [↑](#footnote-ref-2)
3. https://arxiv.org/pdf/1602.07261v2.pdf [↑](#footnote-ref-3)
4. https://arxiv.org/pdf/1610.02357.pdf [↑](#footnote-ref-4)
5. https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html [↑](#footnote-ref-5)
6. https://pl.wikipedia.org/wiki/SAP\_ERP [↑](#footnote-ref-6)
7. https://towardsdatascience.com/a-very-brief-introduction-to-fuzzy-logic-and-fuzzy-systems-d68d14b3a3b8 [↑](#footnote-ref-7)
8. https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 [↑](#footnote-ref-8)
9. https://maelfabien.github.io/deeplearning/Perceptron/#the-classic-model [↑](#footnote-ref-9)
10. https://www.niser.ac.in/~smishra/teach/cs460/2020/lectures/lec19/ [↑](#footnote-ref-10)
11. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd [↑](#footnote-ref-11)
12. https://pl.wikipedia.org/wiki/Maszyna\_wektor%C3%B3w\_no%C5%9Bnych [↑](#footnote-ref-12)
13. https://proceedings.neurips.cc/paper\_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf [↑](#footnote-ref-13)
14. https://arxiv.org/pdf/1406.2661.pdf [↑](#footnote-ref-14)
15. https://arxiv.org/pdf/1706.03762.pdf [↑](#footnote-ref-15)