AKADEMIA GÓRNICZO-HUTNICZA

im. Stanisława Staszica w Krakowie

WYDZIAŁ ZARZĄDZANIA

STUDIA STACJONARNE II stopnia

KIERUNEK: Informatyka i Ekonometria

PRZEDMIOT: Uczenie maszynowe

PROWADZĄCY: dr inż. Marcin Pietroń

Projekt zaliczeniowy

Agnieszka Pytel

Budowa sieci CNN do klasyfikacji obrazów rentgenowskich

Projekt w języku angielskim:

https://www.kaggle.com/agnieszkapytel/cnn-chest-x-ray-classification

Kraków, 20.05.2019

1) Wstęp

Motywacja

Uczenie maszynowe zdobywa coraz większą popularność ze względu na możliwość wykorzystania w różnorodnych dziedzinach, łatwość implementacji oraz bardzo dobre rezultaty ich zastosowania dla danego problemu. Dzięki wciąż powiększającej się mocy obliczeniowej komputerów mocno rozwinęło się w ostatnich latach szczególnie uczenie głębokie (ang. *deep learning*), które obecnie jest jednym z najczęściej wybieranych narzędzi w pracy z obrazami, wideo czy tekstem. Zastosowanie uczenia głębokiego do klasyfikacji obrazów niejednokrotnie pozwala osiągnąć wyniki lepsze niż w przypadku klasyfikacji przez człowieka. Jest to szczególnie istotna uwaga w kontekście analizy danych medycznych, gdzie prawidłowa lub nieprawidłowa klasyfikacja niejednokrotnie jest kwestią życia i śmierci pacjenta.

Cel projektu

Celem niniejszego projektu jest stworzenie głębokiej sieci konwolucyjnej do klasyfikacji obrazów rentgenowskich w celu wykrycia u pacjenta zapalenia płuc.

2) Opis danych

Zbiór danych[1] dla rozwiązywanego problemu składa się z 5856 obrazów w formacie JPEG o różnych rozmiarach i liczbie kanałów (2 kanały dla obrazów w skali szarości oraz 3 kanały dla obrazów RGB) oraz pliku CSV zawierającego etykiety obrazów, gdzie:

- **0 –** oznacza zapalenie płuc,
- **1 –** oznacza obraz zdrowych płuc.

Obrazów zdrowych płuc jest ponad dwukrotnie więcej niż obrazów zapalenia płuc: odpowiednio 73% i 27% zbioru.

3) Użyte metody

Do rozwiązania wspomnianego problemu wykorzystano głęboką sieć konwolucyjną (ang. *CNN – convolutional neural network*), gdyż jest ona polecana w pracy z obrazami i daje bardzo dobre efekty. Jest to sieć głęboka, w odróżnieniu od prostej sieci neuronowej składa się oprócz warstwy wejściowej i wyjściowej z wielu warstw ukrytych. Głównym mechanizmem

1Link do zbioru danych: https://www.kaggle.com/parthachakraborty/pneumonia-chest-x-ray

wykorzystywanym w konwolucyjnych sieciach neuronowych jest konwolucja (nazywana także *splotem*) – operacja matematyczna na dwóch funkcjach, dająca w wyniku trzecią funkcję - może być ona interpretowana jako modyfikacja pierwszej funkcji. W przetwarzaniu obrazów korzysta się z konwolucji macierzy pikseli obrazu z filtrem, otrzymując w wyniku nową macierz pikseli, utworzonych na podstawie sąsiedztwa. Sama idea wykorzystania konwolucji w przetwarzaniu obrazów ma źródło w analizie postrzegania obrazu przez człowieka – zwracamy uwagę nie tylko na dany obiekt, ale także na jego otoczenie.

Aby rozwiązać problem niezrównoważonego zbioru danych, powielono trzykrotnie obrazy rentgenowskie zapalenia płuc, a następnie wykorzystano klasę ImageDataGenerator zaimplementowaną we frameworku Keras, która pozwala na przeprowadzenie losowych przekształceń w zadanym zakresie, takich jak: zmiana wysokości, zmiana szerokości, obrót, obrazów ładowanych podczas trenowania modelu.

W celu zapobiegnięcia nadmiernemu dopasowaniu modelu wykorzystano w sieci neuronowej warstwę Dropout, w której zadany odsetek połączeń między neuronami jest usuwany.

4) Użyte narzędzia i technologie

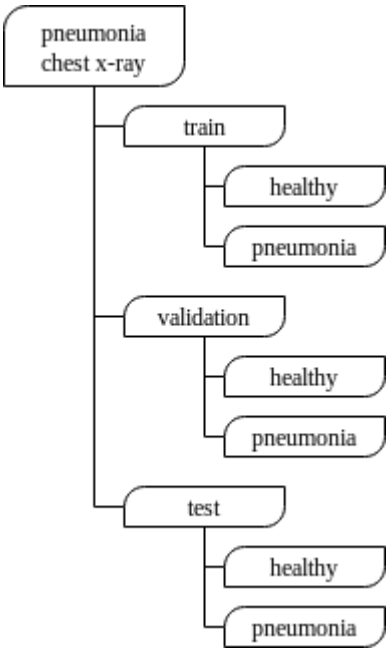
Kod projektu został napisany w języku Python, a całość opracowana w postaci notatnika Jupyter. Do wstępnego przetwarzania obrazów oraz budowy konwolucyjnej sieci neuronowej wykorzystano framework do uczenia głębokiego Keras oraz TensorFlow – bibliotekę do uczenia maszynowego. Do operacji na obrazach i pliku CSV oraz manipulacji danymi użyto pakietów numpy, pandas, matplotlib. Do operacji na plikach i katalogach wykorzystano pakiety os i shutil.

Ze względu na niewystarczającą moc obliczeniową dostępnego sprzętu fizycznego, wykorzystano do obliczeń zasoby udostępniane przez platformę Kaggle (w tym procesor GPU współpracujący z frameworkami Keras i TensorFlow). W związku z tym kod projektu został wykonany i zapisany na platformie Kaggle w postaci notatnika Jupyter. Wykonanie kodu na komputerze osobistym wymagałoby skonfigurowania środowiska do korzystania z GPU (zalecany jest procesor firmy Nvidia) zamiast CPU, gdyż w przeciwnym razie czas trenowania modelu znacząco by się wydłużył.

5) Opis algorytmu

Opis algorytmu w języku angielskim wraz z obliczeniami i wynikami jest przedstawiony w notatniku Jupyter, do którego link można znaleźć na stronie tytułowej niniejszego projektu oraz w Podsumowaniu.

W ramach preprocessingu obrazy ze zbioru rozdzielono do poszczególnych folderów na potrzeby wykorzystania funkcji *flow_from_directory*, która pozwala na ładowanie już wstępnie przetworzonych obrazów (zmiana rozmiaru, zadane losowe przekształcenia) na bieżąco podczas trenowania modelu, bez konieczności uprzedniego wczytania całego zbioru danych do pamięci, co bardzo obciążyłoby procesor. Struktura folderów przedstawiona jest na Rysunku 1.



Rysunek 1. Struktura folderów utworzona na potrzeby trenowania modelu.

Kolejność plików i etykiet w pliku CSV nie odpowiada kolejności obrazów w głównym folderze, dlatego utworzono dwie listy z nazwami plików należących do obu klas.

Zbiór danych został podzielony na zbiór treningowy, walidacyjny i testowy w następujących proporcjach: odpowiednio 80%, 10% i 10%. Ostateczna liczba obrazów w folderach przedstawionych powyżej to:

- w folderze *train*: 7220 obrazów,
- w folderze *validation*: 901 obrazów,
- w folderze *test*: 901 obrazów.

Zaleca się, aby rozmiar wsadów zadawanych modelowi do walidacji i predykcji był podzielnikiem liczby obrazów, w celu zapewnienia, że każdy obraz będzie przetwarzany przez model dokładnie jeden raz. Ustalono zatem, że rozmiar wsadu wyniesie 53, a liczba kroków walidacji oraz predykcji – 17.

Wielkość wsadu do trenowania została ustawiona na liczbę będącą potęgą liczby 2, w tym przypadku to 64, ponieważ procesory są zoptymalizowane pod kątem operacji binarnych. Rozmiar wejściowy obrazów wynosi 128 x 128 pikseli. Wszystkie obrazy były ładowane w skali szarości.

Zastosowany w tym przypadku model CNN składał się z 3 warstw konwolucyjnych *Conv2D* (każda z następującymi po sobie warstwami normalizacji wsadu *BatchNormalization*, zapewniającej stały zakres liczbowy po przekształceniach konwolucyjnych, i *MaxPooling,* umożliwiającej zmniejszenie liczby wymiarów danych), warstwy *Dropout*, wspomnianej w rozdziale 3 projektu, warstwy spłaszczającej *Flatten* i 2 warstw gęstych *Dense*. Szczegółowe podsumowanie modelu wraz z liczbą parametrów do dostrojenia można znaleźć w notatniku Jupyter z kodem niniejszego projektu.

Dla każdej z warstw poza ostatnią jako funkcję aktywacji wykorzystano ReLU (wyprostowana jednostka liniowa), która filtruje wartości ujemne, zatem jej wyjście zawsze jest nieujemne. Funkcja aktywacji ostatniej warstwy to sigmoida, używana w klasyfikacji binarnej. Jej wyjście jest liczbą z zakresu <0,1> wskazującą prawdopodobieństwo przynależności do klasy pozytywnej (oznaczonej przez model jako klasa „1").

Model był trenowany pod kątem minimalizacji funkcji straty, którą w tym przypadku jest binarna entropia krzyżowa (ang. *binary crossentropy*), stosowana w klasyfikacji binarnej. Jako metrykę wydajności modelu wybrano dokładność (ang. *accuracy*). Jako algorytm optymalizacji, określającej, w jaki sposób wagi mają być modyfikowane na podstawie funkcji straty, został wybrany SGD – *stochastic gradient descent*, czyli stochastyczny spadek wzdłuż gradientu.

Model trenowano przez 50 epok z wykorzystaniem wywołań zwrotnych udostępnianych w pakiecie Keras, aby proces trenowania był bardziej efektywny. Są to funkcje, które umożliwiają wykonywanie pewnych operacji podczas treningu:

- *ReduceLROnPlateau* - pozwala na dynamiczną redukcję szybkości uczenia się, gdy nie ma poprawy wyniku funkcji straty w danej liczbie epok,
- *EarlyStopping* - pozwala przerwać trenowanie modelu, kiedy nie ma poprawy wyniku funkcji straty w danej liczbie epok,

- *ModelCheckpoint* - pozwala zapisać najlepsze wagi modelu zanim zaczną się pogarszać w wyniku nadmiernego dopasowywania modelu do danych treningowych.

Kroki na epokę obliczono jako iloraz liczby obrazów treningowych i wielkości wsadu treningowego, ustalonej wcześniej ręcznie.

Obliczono liczbowe prognozy dla obrazów w zestawie testowym w celu stworzenia macierzy pomyłek, metryk klasyfikacji i pokazania prognoz losowo wybranej próbki obrazów z klasy oznaczającej zapalenie płuc.

Szczegółowe wyniki trenowania modelu oraz prognoz są umieszczone w notatniku Jupyter z kodem projektu. Osiągnięto dokładność predykcji dla zbioru testowego na poziomie 98%.

6) Podsumowanie

W modelu stworzonym w ramach niniejszego projektu osiągnięto w dokładność 98%, jednak liczba ta jest inna przy każdym uruchomieniu kodu. Wyniki działania modelu nie są powtarzalne ze względu na to, że przy wykorzystaniu do obliczeń GPU oraz zastosowaniu losowych przekształceń obrazów przez ImageDataGenerator nie można uniknąć pewnej losowości.

Metody uczenia głębokiego są obecnie mocno rozwijane ze względu na ogromny potencjał związany z wykorzystaniem ich w wielu dziedzinach, nie tylko tych ściśle związanych z informatyką. Problemy związane z przetwarzaniem tekstu, obrazów, wideo czy mowy mają szansę wkrótce zostać zautomatyzowane, co pozwoli m. in. oszczędzić czas oraz uniknąć błędów popełnianych przy takich operacjach przez ludzi, a wynikających choćby z fizycznego zmęczenia, które nie dotyczy komputerów. Wśród różnych dziedzin również medycyna może bardzo skorzystać na wykorzystaniu głębokiego uczenia nie tylko do diagnostyki chorobowej, ale także do bardziej eksperymentalnych problemów, np. badań nad DNA.

Link do zbioru danych:
https://www.kaggle.com/parthachakraborty/pneumonia-chest-x-ray

Link do jądra Kaggle (notatnik Jupyter z wykonanym kodem):
https://www.kaggle.com/agnieszkapytel/cnn-chest-x-ray-classification