Sieci Neuronowe i Deep Learning - Projekt zaliczeniowy

RAPORT

Tematem naszego projektu jest stworzenie modelu klasyfikacji dla zbioru danych zawierającego litery języka migowego. Klasyfikacja obrazów polega na automatycznym rozpoznawaniu i przypisywaniu etykiet do obrazów na podstawie ich zawartości. Skupimy się na podejściu wykorzystującym konwolucyjną sieć neuronową (CNN), przy użyciu biblioteki PyTorch.

Zbiór danych użyty w projekcie pochodzi ze strony kaggle i można go znaleźć pod adresem:

https://www.kaggle.com/datasets/datamunge/sign-language-mnist

Format zestawu danych jest wzorowany na klasycznym MNIST. Każdy przypadek jest reprezentowany przez etykietę (0-25) jako przypisanie

jednoznaczne do każdej litery alfabetu A-Z (brak przypadków dla 9=J lub 25=Z ze względu na ruchome gesty). Rozmiar danych jest mniej więcej dwa razy mniejszy niż rozmiar standardowego zestawu MNIST, ale są podobne pod względem struktury - pierwsza kolumna zawiera etykietę, a pozostałe piksele od 1 do 784, reprezentujące pojedynczy obraz o rozmiarze 28x28 pikseli z wartościami odcieni szarości między 0 a 255.

Oryginalne obrazy gestów ręki obejmowały powtórzenia gestu przez różnych użytkowników na różnych tłach. Dane MNIST języka migowego pochodzą z rozszerzenia niewielkiej liczby (1704) kolorowych obrazów, które nie były przycięte do obszaru ręki. Aby stworzyć nowe dane, użyto ‘image pipeline’ opartego na ImageMagick, który obejmował przycinanie tylko rąk, konwersję na odcienie szarości, zmianę rozmiaru, a następnie utworzenie co najmniej 50 różnych wariantów w celu zwiększenia ilości. Strategia modyfikacji i rozszerzenia obejmowała filtry ('Mitchell', 'Robidoux', 'Catrom', 'Spline', 'Hermite'), oraz 5% losową pikselizację, +/- 15% jasność/kontrast, i na końcu obrót o 3 stopnie. Ze względu na mały rozmiar obrazów, te modyfikacje efektywnie zmieniają rozdzielczość i separację klas w ciekawy i kontrolowany sposób.

Pracę rozpoczęliśmy od wczytania danych i ich wstępnej analizy: sprawdzenia wymiarów danych, ewentualnych brakujących wartości, liczby elementów w zbiorach treningowym i testowym oraz wyświetliliśmy kilka przykładowych zdjęć z naszego zestawu danych. Dane nie posiadają braków, ani widocznych anomalii, a wyświetlone obrazy wydają się być poprawne. Zestaw danych podzieliliśmy na zbiór treningowy, walidacyjny i testowy zawierających 20445, 7000, 7172 elementy. Etykiety treningowe, walidacyjne i testowe przekonwertowaliśmy na tensory typu long, a tablice pikseli zostały przekonwertowane na tensory typu float. Następnym krokiem było utworzenie obiektów TensorDataset, oraz obiektów typu DataLoader. Dzięki tym czynnościom mogliśmy przystąpić do trenowania oraz testowania modeli opartych o nasze dane.

Budowa modelu:

Pierwszym bazowym modelem, który zbudowaliśmy była sieć CNN, składająca się z dwóch warstw konwolucyjnych: pierwsza z 1 kanałem wejściowym i 32 kanałami wyjściowymi z funkcją aktywacji ReLU, następnie MaxPoolingiem i druga warstwa konwolucyjna, zawierająca 32 kanały wejściowe i 64 wyjściowe, również z funkcją aktywacji ReLU oraz

MaxPoolingiem. Jądro konwolucyjne miało rozmiar 5x5, a padding ustawiliśmy na wartość 2. Po warstwach konwolucyjnych dodaliśmy warstwę spłaszczającą, a następnie warstwę w pełni połączoną, z funkcją aktywacji ReLU i dropoutem w celu redukcji nadmiernego dopasowania. Warstwa ukryta składała się z 1024 neuronów, a model kończy się warstwą w pełni połączoną, która klasyfikuje wejście do jednej z 25 klas. Do trenowania modelu użyliśmy funkcję straty CrossEntropyLoss, optymalizator Adam oraz learning rate ustawiony na 0.001.

Wykonaliśmy kolejno następujące modyfikacje naszego modelu:

1. Najpierw zmieniliśmy learning rate na 0.01, co okazało się nietrafionym pomysłem; dokładność takiego modelu wynosiła 2.01%.
2. W modelu “model2” zastosowaliśmy funkcję aktywacji LeakyReLU zamiast wcześniej użytej ReLU po pierwszej warstwie w pełni połączonej, learning rate ustawiony na 0.001.
3. “Model3” jest bardziej złożony niż poprzednie, ponieważ do modelu bazowego dodaliśmy dodatkową w pełni połączoną warstwę oraz zmniejszyliśmy prawdopodobieństwo Dropout.
4. W modelu “model4” podobnie jak w poprzednim, zastosowaliśmy dwie w pełni połączone warstwy, ale z innymi funkcjami aktywacji w warstwach konwolucyjnych: zamiast ReLU użyliśmy Sigmoid. Prawdopodobieństwo Dropout pozostało na poziomie 0.25 oraz lr = 0.001.
5. “Model5” jest wykonany analogicznie do modelu bazowego, zmniejszona została jedynie wartość learning rate na 0.00001
6. W modelu “model6”, podobnie jak w “model3” dodaliśmy w pełni połączoną warstwę oraz zmniejszyliśmy prawdopodobieństwo Dropout, przy wartości learning rate 0.00001
7. “Model7” wykonaliśmy analogicznie do modelu4, learning rate na poziomie 0.00001
8. W modelu “model8” funkcję aktywacji po pierwszej warstwie konwolucyjnej zmieniliśmy ponownie na ReLU, funkcja aktywacji po drugiej warstwie ustawiona na Sigmoid, pozostałe parametry bez zmian, learning rate równy 0.00001

9. W modelu “model9” wykorzystaliśmy parametry modelu3, zmieniliśmy liczbę neuronów w warstwach ukrytych (1024 zmieniliśmy na 512).
10. “Model10” został wykonany analogicznie do modelu4, podobnie jak w poprzednim przypadku zmieniliśmy jedynie liczbę neuronów z 1024 na 512.
11. W modelu “model11” postanowiliśmy sprawdzić zmianę optymalizatora. Wykorzystaliśmy model10 i ustawiliśmy optymalizator na SGD. Dokładność tak wykonanego modelu wynosiła zaledwie 2.01%.

Uczenie każdego z modeli przeprowadzaliśmy przez 10 epok. W poniższej tabeli znajduje się porównanie wyników na zbiorach testowych:

MODEL	TEST ACC
model (bazowy)	87.16
model1	2.01
model2	86.36
model3	87.74
model4	91.82
model5	89.65
model6	91.52
model7	59.56
model8	82.28
model9	91.65
model10	91.59
model11	2.01

Po 10 epokach, model bazowy uzyskał dokładność 0.9840 na zbiorze treningowym i 0.9996 na zbiorze walidacyjnym. Pierwsza modyfikacja tj. zmniejszenie wartości learning rate zdecydowanie pogorszyła te wyniki, osiągając odpowiednio 0.0449, oraz 0.0469 na zbiorach treningowym i

walidacyjnym. Model2 osiągnął bardzo podobne wyniki do modelu bazowego - 0.9858 i 0.9969. Kolejna modyfikacja uzyskała odpowiednio 0.9890 i 0.9999. Model4 miał dokładność 0.9834 oraz 0.9953, a model5 0.9826 i 0.9999. Zmniejszenie wartości parametru dropout w modelu6 poskutkowała dokładnością 0.9964 na zbiorze treningowym i 0.9999 na walidacyjnym. Model7 okazał się o wiele gorszy uzyskując jedynie 0.5634 na zbiorze treningowym i 0.6181 na walidacyjnym. Dwa kolejne modele tj. model8 i model9 uzyskały odpowiednio 0.9122 oraz 0.9926 na zbiorze treningowym i 0.9371 oraz 1.0000 na walidacyjnym. Przedostatni model również uzyskał bardzo wysoką dokładność - 0.9858 i 0.9997. Ostatni natomiast okazał się bardzo słaby z wynikami 0.0471 i 0.0464.

Podsumowanie:

Finalnie postanowiliśmy wybrać model10 jako najlepszy. Wykorzystanie CNN w kontekście postawionego problemu okazało się skutecznym narzędziem. Dostosowanie struktury sieci oraz jej parametrów pozwoliło nam na osiągnięcie zadowalających rezultatów. Na koniec wyświetliliśmy 12 przykładów ze zbioru wraz z predykcjami oraz wybrane przykłady (ze zbioru testowego), które zostały sklasyfikowane błędnie.

Kraków, 15.06.2024

Karolina Grzech, Krystian Bułat