Rozpoznawanie Liter Przy Pomocy Konwolucyjnej Sieci Neuronowej

Barbara Doncer, Wojciech Dróżdż

30 marca 2022

1 Cel obliczeń

Celem utworzonej sieci neuronowej jest poprawne przewidywanie odręcznie napisanych liter. Problem jest zaskakująco trudny dla tradycyjnych algorytmów wizji komputerowej ze względu na liczne charaktery pisma, rozmiary i orientacje liter.

2 Opis danych

2.1 Źródło danych

W celu treningu sieci neuronowej zostały wykorzystane dane ze zbioru *EMNIST*, jest to rozszerzona wersja popularnego zbioru *MNIST*, która posiada więcej danych od oryginału.

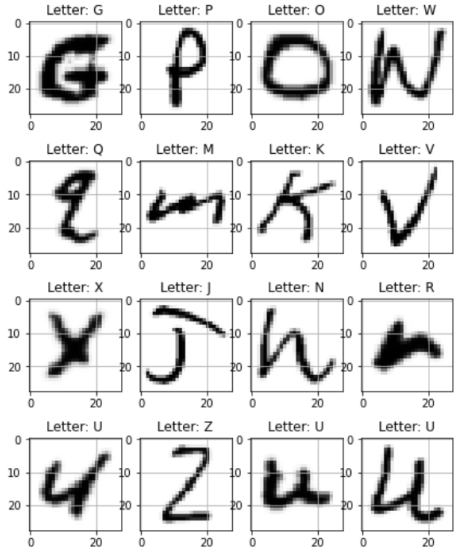
2.2 Grupy danych w zbiorze

- *ByClass i ByMerge* - Oba podzbiory zawierają niezbalansowane i kompletne dane zbioru, włączne z cyframi, które dominują dataset *EMNIST*
- *Balanced* - Zbalansowany podzbiór zbioru ByMerge, zawiera również cyfry
- *Letters* - Zbiór użyty w projekcie - jest zbalansowany i zawiera tylko litery
- *Digits* - Zbalansowany zbiór zawierający tylko cyfry

2.3 Format danych

Dane są dostarczone w formacie *CSV*. Każdy wiersz ma 785 kolumn: pierwsza kolumna to identyfikator litery przedstawionej w danym wierszu, pozostałe 784 kolumny to jednowymiarowa reprezentacja zdjęcia o rozmiarach *28x28* pikseli.

2.4 Przykład danych



Rysunek 1: Przykład użytych danych

2.5 Przygotowanie danych

Dane nie wymagały dużych ilości przygotowań. Po oddzieleniu zdjęć liter od podpisów, za pomocą funkcji *numpy.reshape* tablica jednowymiarowa o szerokokości 784 pikseli została zamieniona na tablicę 2D o wymiarach *28x28* pikseli. Następnie w celu normalizacji wartości w tabeli zostały podzielone przez wartość 255.

3 Użyte oprogramowanie

W celu wykonania projektu zostały użyte:

- Język *Python* - Najpopularniejszy język do tworzenia sieci neuronowych
- *Keras* - Biblioteka do uczenia maszynowego zbudowana na podstawie *Tensofrlow*
- *Numpy* - Biblioteka do zaawansowanych obliczeń numerycznych
- *Matplotlib* - Biblioteka do wizualizacji danych
- *OpenCV i Tkinter* - Biblioteki nie były użyte w czasie treningu, ale zostały wykorzystane w celu stworzenia aplikacji korzystającej ze stworzonego modelu.

4 Parametry Symulacji

4.1 Ilość użytych danych

W celu treningu zostały wykorzystane następujące ilości danych:

- Obiekty treningu - Do treningu zostało wykorzystane *88799* podpisanych zdjęć. *20%* zbioru zostało wydzielone na cel walidacji modelu
- Obiekty testu - Do testów wykorzystano *14799* podpisanych zdjęć

4.2 Struktura sieci neuronowej

Została wykorzystana sieć neuronowa o następujących warstwach:

- Wejściowa warstwa konwolucyjna 2D - 32 filtry, rozmiar filtra: *(3x3)*, funkcja aktywacji ReLU
- Warstwa łącząca 2D - rozmiar *2x2*

- Warstwa konwolucyjna 2D - 32 filtry, rozmiar filtra: *(3x3)*, funkcja aktywacji ReLU
- Warstwa łącząca 2D - rozmiar *2x2*
- Warstwa konwolucyjna 2D - 32 filtry, rozmiar filtra: *(3x3)*, funkcja aktywacji ReLU
- Warstwa łącząca 2D - rozmiar *2x2*
- Warstwa spłaszczająca model do jednego wymiaru
- Warstwa gęsta - 64 jednostki, funkcja aktywacji ReLU
- Warstwa gęsta - 128 jednostek, funkcja aktywacji ReLU
- Warstwa wyjściowa - 26 jednostek, funkcja aktywacji softmax

Ostatecznie sieć ma 137178 parametrów do optymalizacji

4.3 Trening sieci neuronowej

Model został wytrenowany dokonując 7 iteracji po całym zbiorze danych używając funkcji straty *categorical crossentropy* i optymalizatora *Adam*. Dla danych treningowych sieć miała celność na poziomie 95%.

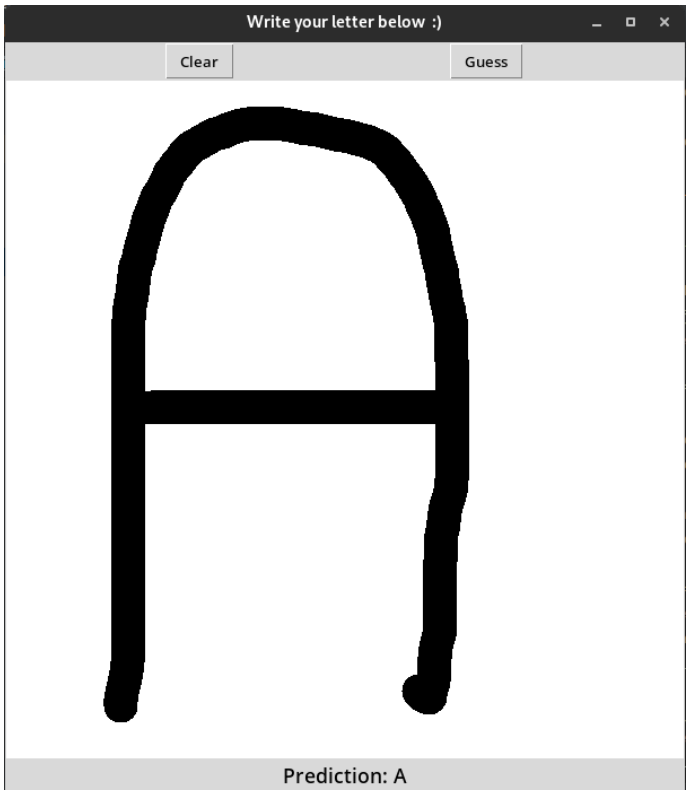
5 Test działania sieci neuronowej

5.1 Test na zbiorze testowym

Na zbiorze testowym sieć ma celność na poziomie 91.5%, co jest bardzo dobrym wynikiem, szczególnie biorąc pod uwagę krótki czas treningu, który zajął około dwóch minut. Sieć jest również szybka, podjęcie decyzji na podstawie danych wejściowych zajmuje około $5ms$.

5.2 Test przez użytkownika

Została przygotowana atrakcyjna aplikacja pozwalająca użytkownikowi narysować literę odręcznie. Po wciśnięciu przycisku *guess* sieć odgadnie narysowaną literę.



Rysunek 2: Zdjęcie stworzonej aplikacji

6 Wnioski

Została utworzona skuteczna i praktyczna sieć neuronowa pozwalająca na zastosowanie w realnych problemach. Przy większym nakładzie pracy prawdopodobnie udałoby się znacznie poprawić skuteczność sieci. Szczególnie pomocne mogłoby być poszerzenie zbioru treningowego i zwiększenie ilości warstw sieci.
Stworzony model jest łatwo wykorzystać w prawdziwej aplikacji, co zostało pokazane w przykładzie aplikacji powyżej.