Digit Recognizer using Convolutional Neural Network (CNN) technique

Patryk Adler

1. Cel projektu

Opisany poniżej projekt ma za zadanie doprowadzić komputer/maszynę do odczytania poprawnie cyfr z różnych zdjęć. Komputer na podstawie danego zdjęcia przeprowadza odpowiednie obliczenia tj. porównuje swoje dane, które wcześniej otrzymał do nauki i z ich pomocą stwierdza, jaka cyfra jest widoczna na otrzymanym zdjęciu.

Rozpoznawanie cyfr z obrazów posiada aktualnie wiele zastosowań takie jak numeracja budynków w nawigacjach/mapach (np. google maps), w instytucjach takich jak np. poczta, zajmujących się obróbką dużej ilości formularzy/danych np. odczyt kodów pocztowych jak również w aplikacjach na telefony/tablety, które mają za zadanie odczytywać tekst ze zrobionych przez użytkownika zdjęć (np. Office Lens).

2. Wykorzystane narzędzia i technologie

W projekcie wykorzystano narzędzia:

-R w wersji 3.4.2 64 bit z paczkami tensorflow, magrittr

-Python w wersji 3.6.4 64 bit z biblioteką NumPy

R oraz Python'a należy zainstalować samodzielnie na dostępnej maszynie a następnie uruchomić załączony kod dostępny w pliku „digit_recognizer_cnn.txt". Biblioteki tensorflow oraz magittr powinny samoczynnie zostać ściągnięte i załadowane. Domyślnie zostanie wykorzystany przesłany model nauczonej bazy w folderze „model", jeżeli nie chcemy korzystać z tego modelu i nauczyć sieć od nowa możemy zmienić nazwę folderu albo skomentować za pomocą znaku *„#"* poniższą linijkę

```
[234] saver$restore(sess, paste(currentPath,"model/model.ckpt", sep = ""))
```

Zastosowana technika nauczania:

-Convolutional Neural Network

3. Dane do projektu

Dane treningowe oraz testowe dostępne są do pobrania po zalogowaniu na platformie Kaggle.com (https://www.kaggle.com/c/digit-recognizer/data)

Zapisane zostały w dwóch plikach .csv i zawierają w sobie przede wszystkim liczbowy zapis zdjęć o rozmiarach 28 na 28 pikseli.

Plik train.csv służący do treningu posiada 785 kolumn, pierwsza odpowiada cyfrze, jaka została przedstawiona na zdjęciu, następnie pozostałe to zapis każdego pojedynczego piksela dla zdjęcia. Pojedynczy piksel przyjmuje wartość od 0 do 255 co ma odpowiadać jego jasności (0 znaczy jasny, 255 znaczy ciemny).

Plik test.csv zawiera 28000 obrazków w podobnym zapisie co train.csv lecz bez pierwszej kolumny zawierającej opis cyfry widocznej na obrazku.

Dane na potrzeby projektu należało najpierw wczytać następnie odpowiednio podzielić i obrobić. Na potrzeby trenowania bazy, podzieliliśmy, więc dane treningowe w proporcjach 70-30. Następnie pomieszaliśmy wiersze, aby mieć pewność, że dane wejściowe będą odpowiednio losowe a nie np. narastające w podobny sposób. Mogłoby to wpłynąć niekorzystnie na rozwój sieci neuronowej. Dane zostały przekonwertowane z listy wektorów o różnych typach (Data Frame) do listy o jednakowych danych (Matrix). Mamy do czynienia ze znaczną ilością danych, dlatego korzystamy z list typu Matrix, gdyż zajmuje ona mniej miejsca w pamięci i spełnia nasze wymagania.

4. Teoria i eksperymenty

W celu odgadywania cyfr wykorzystujemy konwolucyjną sieć neuronową. Sieć jest podobna do tradycyjnej sieci neuronowej jednak wykorzystuje ona dodatkowy tj. 3 wymiar ‘depth’ oraz obsługuje różne warstwy, na których możemy zastosować zróżnicowane filtry. Dla naszej sieci konwolucyjnej wykorzystujemy 7 warstw w następującej kolejności:

-Pierwsza warstwa Konwolucyjna (Convolution layer 1)

-Pierwsza warstwa Max-Pooling (Max pooling layer 1)

-Druga warstwa Konwolucyjna (Convolution layer 2)

-Druga warstwa Max-Pooling (Max pooling layer 2)

-Warstwa spłaszczania (Flatten Layer)

-Pierwsza warstwa w pełni połączona (Fully-connected layer 1)

-Druga warstwa w pełni połączona (Fully-connected layer 2)

Poszczególne warstwy działają, jako filtry, które mogą służyć np. wydobyciu krawędzi z obrazów, rozmyciu, wyostrzeniu bądź wykryciu charakterystycznych układów linii i kształtów.

Warstwy konwolucyjne przeprowadzają operacje konwolucji na naszych danych.

Max-Pooling to operacja redukcji liczby pikseli w obrazie, w naszym projekcie warstwa ta ma 2x2 pikseli a więc będzie zmniejszać dane wchodzące dwukrotnie.

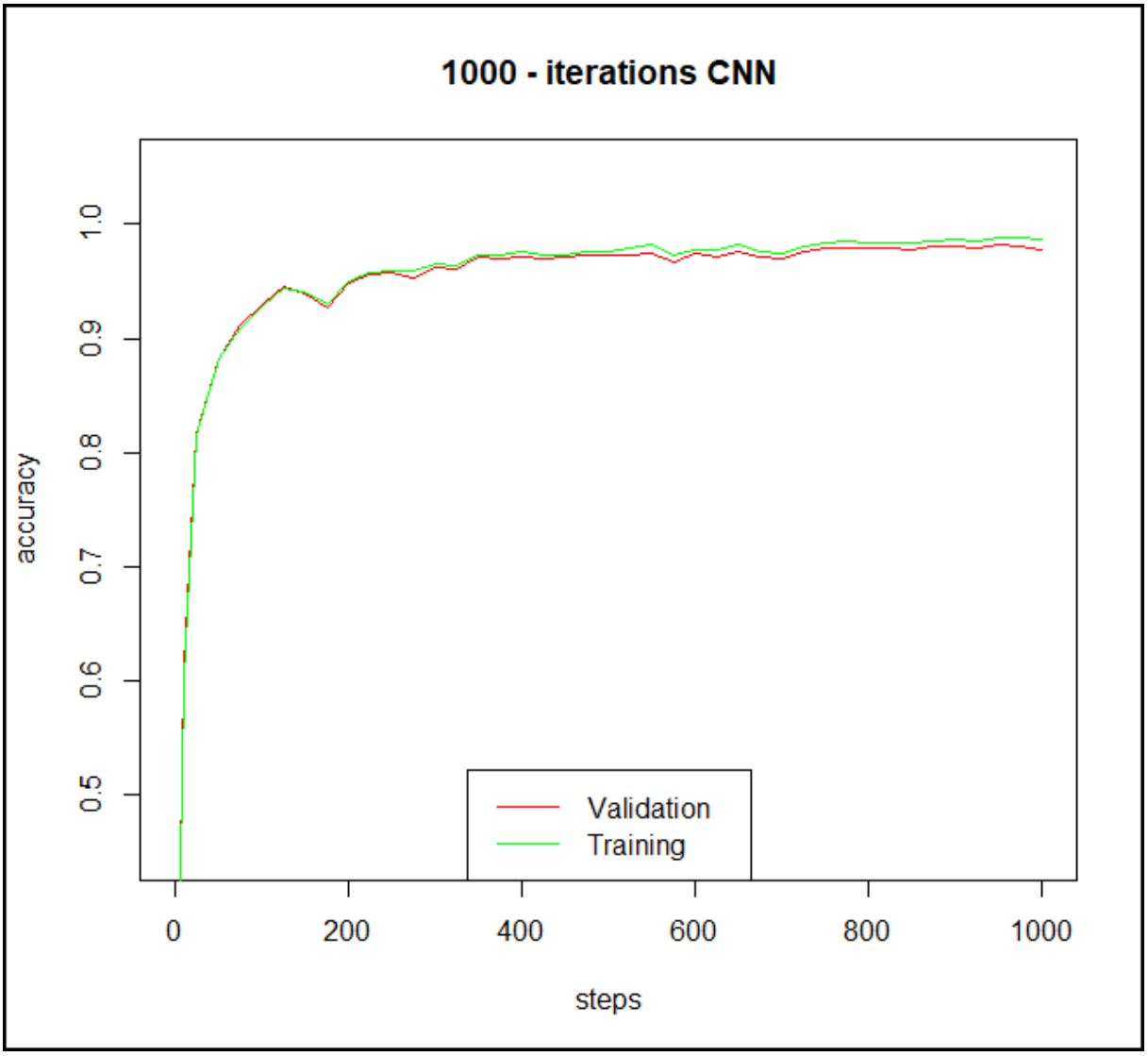
Warstwa spłaszczania (Flatten layer) zamienia dane wejściowe w pojedynczy wektor, aby możliwe było połączenie warstw.

Warstwy w pełni połączone łączą wszystkie neurony z jednej warstwy ze wszystkimi neuronami w kolejnej tak jak ma to miejsce w tradycyjnej sieci neuronowej.

Finalnie otrzymujemy cyfrę od, 0 do 9 która jest wynikiem działania naszej sieci.

5. Wyniki

Zależnie od liczby przeprowadzonych iteracji otrzymujemy, co raz lepsze wyniki. Poniżej przedstawiony jest graf dla 1000 iteracji dla sieci tworzonej od zera. Na następnej stronie widoczne są natomiast tekstowe statystyki.



Trening powyższej sieci zajął około 7 i pół minuty a więc stosunkowo krótko mimo wszystko otrzymaliśmy wysokie 0.977-0.980 skuteczności dla danych walidacyjnych. Jak widać pierwsze 100 iteracji wykazuje największy przyrost to tutaj maszyna robi największe postępy. Następnie trafność naszych wyników nadal wzrasta, ale już dużo wolniejsza. Występują również wahania/spadki, jak widać na początku są największe z czasem stają się znacznie mniejsze. Dla przypomnienia dane treningowe to 70% danych z pliku train.csv, natomiast dane walidacyjne to pozostałe jego 30%.

Dane tekstowe dla grafu:

"Step = 1 || Training Accuracy = 0.106564626097679"
"Step = 1 || Validation Accuracy = 0.107063494622707"
"==="
"Step = 5 || Training Accuracy = 0.432312935590744"
"Step = 5 || Validation Accuracy = 0.433174610137939"
"==="
"Step = 10 || Training Accuracy = 0.600170075893402"
"Step = 10 || Validation Accuracy = 0.612698435783386"
"==="
"Step = 25 || Training Accuracy = 0.814931988716125"
"Step = 25 || Validation Accuracy = 0.816031754016876"
"==="
"Step = 50 || Training Accuracy = 0.883061230182648"
"Step = 50 || Validation Accuracy = 0.8826984167099"
"==="
"Step = 75 || Training Accuracy = 0.908435344696045"
"Step = 75 || Validation Accuracy = 0.911904752254486"
"==="
"Step = 100 || Training Accuracy = 0.929183661937714"
"Step = 100 || Validation Accuracy = 0.930634915828705"
"==="
"Step = 150 || Training Accuracy = 0.940136075019836"
"Step = 150 || Validation Accuracy = 0.9396031498909"
"==="
"Step = 200 || Training Accuracy = 0.950646281242371"
"Step = 200 || Validation Accuracy = 0.948412716388702"
"==="
"Step = 300 || Training Accuracy = 0.966054439544678"
"Step = 300 || Validation Accuracy = 0.961984097957611"
"==="
"Step = 400 || Training Accuracy = 0.97642856836319"
"Step = 400 || Validation Accuracy = 0.971825420856476"
"==="
"Step = 500 || Training Accuracy = 0.976258516311646"
"Step = 500 || Validation Accuracy = 0.972857117652893"
"==="
"Step = 600 || Training Accuracy = 0.978163242340088"
"Step = 600 || Validation Accuracy = 0.975000023841858"
"==="
"Step = 700 || Training Accuracy = 0.974625825881958"
"Step = 700 || Validation Accuracy = 0.969682514667511"
"==="
"Step = 800 || Training Accuracy = 0.984081625938416"
"Step = 800 || Validation Accuracy = 0.978888869285583"
"==="
"Step = 900 || Training Accuracy = 0.98724490404129"
"Step = 900 || Validation Accuracy = 0.98071426153183"
"==="
"Step = 1000 || Training Accuracy = 0.986360549926758"
"Step = 1000 || Validation Accuracy = 0.978015899658203"
"==="

6. Interpretacja wyników

Dla powyższych danych, czyli sieci zbudowanej z 1000 iteracji baza przybiera wagę około 280mb, dla każdej kolejnej iteracji znacząco rośnie i dla około 5 tysięcy ma już wartość 1,5gb. W pewnym momencie dochodzimy jednak do znikomego przyrostu skuteczności teoretycznie opierając się o trafność danych walidacyjnych.

W zadaniu kaggle.com otrzymaliśmy jednak również konkursową paczkę test.csv, która to zawiera wyłącznie dane 28 tysięcy obrazków bez ich poprawnej interpretacji. Ostatnim krokiem przy użyciu naszej nauczonej sieci jest podstawienie tych danych i ich oszacowanie. Finalnie otrzymujemy plik „format_answer.csv", który zawiera odpowiedzi naszej sieci na obrazki testowe. Plik przesyłamy w formularzu dostępnym na stronie. Dla sieci zbudowanej z 5 tysięcy iteracji najlepszy finalny wynik wynosił 0.98628, a więc 1,372% obrazków było błędnie interpretowanych przez nasz algorytm. Myślę, że jest to wynik zadowalający.