Wydział Matematyki
i Nauk Informacyjnych
POLITECHNIKA WARSZAWSKA

Metody głębokiego uczenia

Projekt 2

Sieci konwolucyjne

Michał Rdzany, Piotr Sowiński

Spis treści

Wstęp

Opis problemu

Głównym zadaniem w projekcie jest implementacja modelu klasyfikującego obrazy ze zbioru CIFAR-10. Każdy z obrazów w tym zbiorze przedstawia obiekt należący do jednej z 10 klas.

Cel badań

Głównym celem projektu jest poznanie metod tworzenia sieci konwolucyjnych w praktyce.

Opis danych

Wykorzystywane zbiory zostały pobrane ze strony https://www.kaggle.com/c/cifar-10. Obrazy są kolorowe i mają rozmiary 32x32. Zbiór treningowy liczy 50000 obrazów, a testowy 10000. W zbiorze testowym znajduje się też 290000 nie branych pod uwagę obrazów, aby ograniczyć oszukiwanie w konkursie.

Sposób weryfikacji rezultatów

Do lokalnego sprawdzania skuteczności modelu, ze zbioru treningowego został wydzielony zbiór walidacyjny, liczący 10000 obrazów. Weryfikacja klasyfikacji na zbiorze testowym następuje poprzez zarejestrowanie wyników na platformie kaggle.

Opis zaimplementowanej sieci

Instrukcja użycia aplikacji

Aby wytrenować model wystarczy uruchomić kilka pierwszych bloków z pliku ipynb. Plik ten zawiera także kod generujący z pobranych danych wymaganą przez model strukturę folderów i plików.

Architektura sieci

Wykorzystywana sieć ma strukturę bazującą na VGG[1] z pewnymi modyfikacjami. Na schemacie obok pominięto warstwy aktywacji, batch normalization i dropout.

Obraz RGB 32 x 32
Conv3-32 x 2
Maxpool
Conv3-64 x 2
Maxpool
Conv3-128 x 3
Maxpool
Conv3-256 x 3
Maxpool
Conv3-512 x 4
Maxpool
Flatten
Dense 512
Dense 10 - softmax

Większe architektury (głębsze lub z większymi warstwami) działają lepiej, ale dłużej się trenują. Rozmiar architektury dobrano tak, aby było możliwe przetestowanie wielu parametrów na posiadanym GPU.

Wykorzystanie GPU do treningu sprawia, że nawet przy ustawieniu jednakowego ziarna, wyniki mogą być różne. Trenowanie modelu na jednym wątku CPU, aby zapewnić odtwarzalność wyników, nie było możliwe z uwagi na długi czas treningu. Przy dużej (500+) liczbie iteracji skuteczność modelu na zbiorze treningowym charakteryzuje się odchyleniem standardowym na poziomie około 0.002. Na zbiorze walidacyjnym odchylenie standardowe jest wyższe (około 0.005), co może być spowodowane jego mniejszym rozmiarem.

Wykorzystane techniki

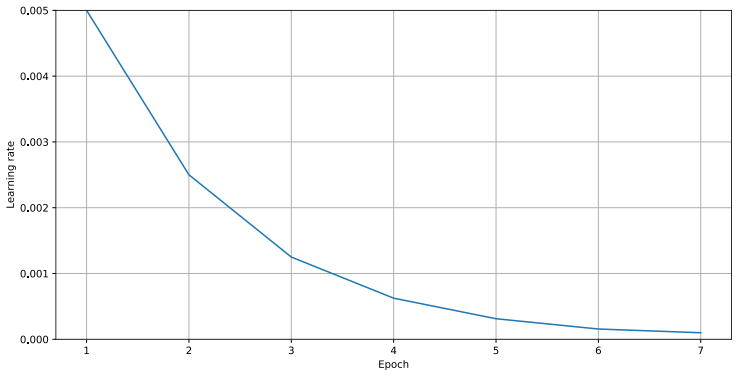
Funkcje aktywacji i straty

W sieci wykorzystywana jest funkcja PReLU[2]. Przetestowano także funkcje ReLU i ELU[3], jednak dawały one gorsze rezultaty. Minusem wykorzystania PReLU jest dłuższy (o około 20% w porównaniu do ReLU) czas iteracji.

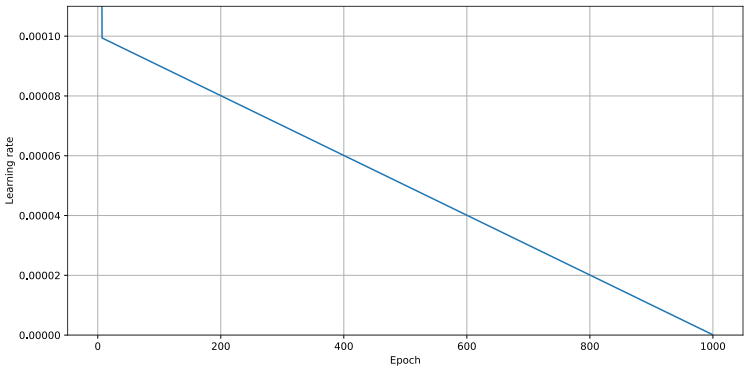
Używana funkcja straty to cross-entropy, w ostatniej warstwie funkcja aktywacji to softmax.

Optymalizacja

Model wykorzystuje algorytm Adam[4]. Przetestowano także algorytm SGD, który działał jednak nieco gorzej. Bardzo istotne okazało się poprawne dobranie współczynnika nauki w kolejnych epokach.



Przez kilka pierwszych epok współczynnik nauki lr jest zmniejszany dwukrotnie po każdej epoce.



W dalszej części procesu uczenia współczynnik nauki maleje liniowo do zera ($lr - ax$).

Przetestowano także funkcje $lr \cdot a^x$, $lr \cdot \frac{1}{1+ax}$ i cykliczne zwiększanie współczynnika nauki[5], jednak techniki te dawały gorsze rezultaty.

Uczenie batchowe

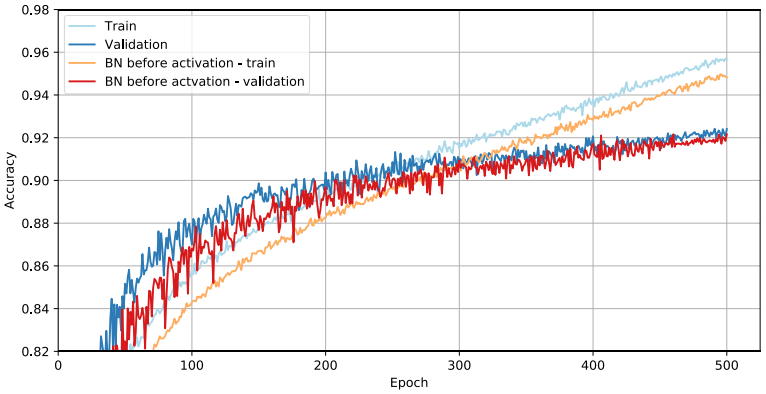
Zastosowano uczenie batchowe z batchami rozmiaru 64. Rozmiar batchy został wybrany na podstawie wydajności GPU używanego do trenowania modelu.

Inicjalizacja

Używana jest inicjalizacja He_uniform, ale zastosowanie Glorot_uniform daje podobne rezultaty.

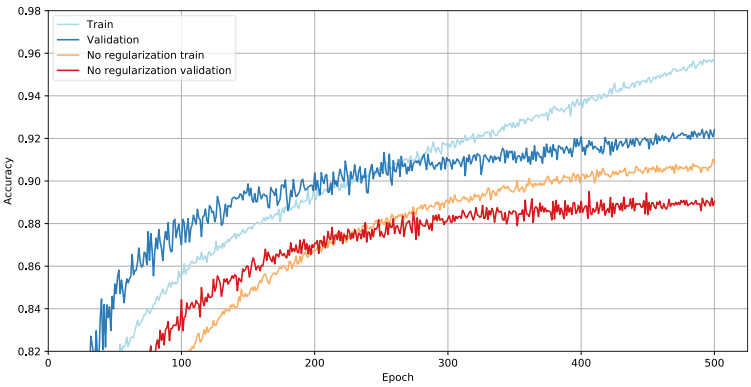
Batch normalization

Zastosowanie tych warstw znacząco polepsza działanie modelu i pozwala na wykorzystanie większych współczynników nauki. W pracy prezentującej tę technikę[6], warstwa ta jest umieszczona przed funkcją aktywacji. Okazuje się jednak, że w naszym modelu umieszczenie jej po funkcji aktywacji daje stale nieco lepsze rezultaty.



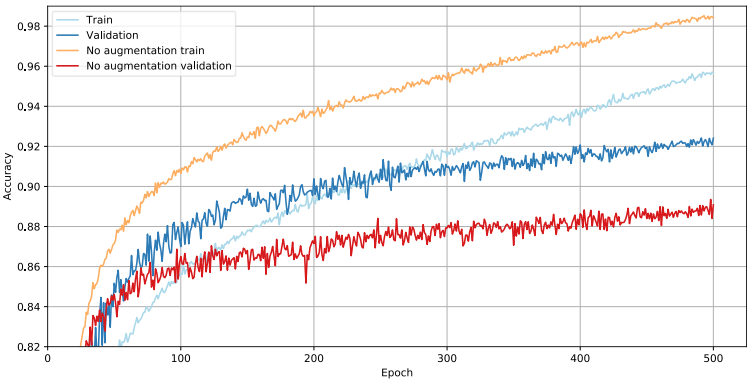
Regularyzacja

Regularyzacja zwiększa skuteczność modelu zarówno na danych uczących jak i walidacyjnych, poprzez utrzymywanie niskich wag połączeń. Wykorzystano regularyzację l2. Wyznaczona optymalna wartość kary (0.002) jest większa niż stosowana w innych podobnych projektach[7,8], co może być spowodowane mniejszym rozmiarem naszego modelu.



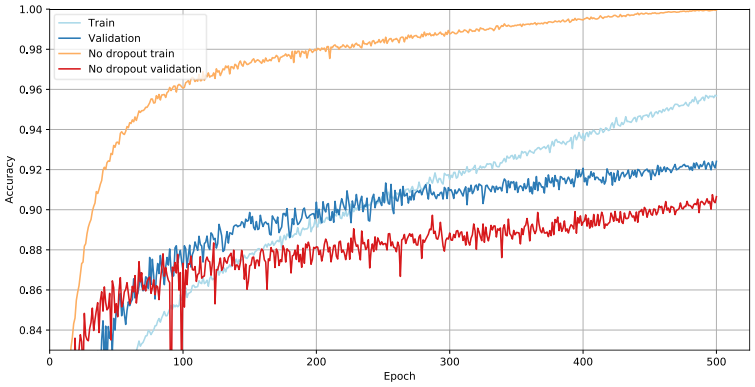
Augmentacja danych

Wykorzystano odbicia poziome i przesunięcia o maksymalnie trzy piksele w dowolnym kierunku w pionie i w poziomie. Zastosowanie tych przekształceń znacząco poprawia zdolność modelu do generalizacji.



Dropout

Jest to kolejny element walczący z problemem przeuczenia. Nie daje może tak spektakularnych rezultatów jak augmentacja (powoduje mniejszy wzrost skuteczności na zbiorze walidacyjnym i znacznie większy spadek na zbiorze treningowym), ale mimo to pozwala wyraźnie poprawić skuteczność modelu na zbiorze walidacyjnym. Najskuteczniejsze okazało się umieszczenie dropoutów w całej sieci i zwiększanie ich wraz z głębokością.



Inne techniki wypróbowane przy tworzeniu modelu

Uczenie transferowe

Wykorzystując modele wytrenowane na bazie ImageNet, dostępne w pakiecie Keras, nie udało się osiągnąć zadowalających rezultatów poprzez wytrenowanie tylko ostatnich warstw. Może to być spowodowane zbyt małym rozmiarem obrazów w porównaniu do ImageNet.

Architektura „All-CNN”

Wykorzystanie tylko warstw konwolucyjnych[9] dawało gorsze rezultaty niż zastosowanie architektury z poolingiem o podobnym stopniu skomplikowania.

Wyniki

Skuteczność modelu

Do ostatecznego treningu modelu wykorzystano wszystkie dostępne dane (połączono zbiór treningowy i walidacyjny). Model był trenowany przez 1000 epok. W konkursie na platformie Kaggle model osiągnął skuteczność 92,89%. Jest to wynik nieco gorszy od innych podobnych projektów[7,8]. Nasz model jest jednak wyraźnie mniejszy, co było konieczne do przetestowania wielu parametrów w rozsądnym czasie z użyciem posiadanego sprzętu.

Cytowane prace

1. **Karen Simonyan and Andrew Zisserman.** *Very Deep Convolutional Networks for Large-Scale Image Recognition.* 2014.

2. **Kaiming He, et al.** Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2015.

3. **Djork-Arné Clevert, Thomas Unterthiner and Sepp Hochreiter.** *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).* 2016.

4. **Diederik P. Kingma and Jimmy Ba.** *Adam: A Method for Stochastic Optimization.* 2014.

5. **Leslie N. Smith.** *Cyclical Learning Rates for Training Neural Networks.* 2015.

6. **Sergey Ioffe and Christian Szegedy.** *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015.

7. **Wei Li.** *cifar-10-cnn: Play deep learning with CIFAR datasets.* https://github.com/BIGBALLON/cifar-10-cnn, 2017.

8. **Yonatan Geifman.** *https://github.com/geifmany/cifar-vgg,* 2017.

9. **Jost Tobias Springenberg, et al.** *Striving for Simplicity: The All Convolutional Net.* 2015.