





Zadanie 03

Pierwszy i prawdopodobnie ostatni raz w życiu (trenujemy od podstaw konwolucyjną sieć neuronową)


Wracamy do problemu klasyfikacji obrazów, którym zajmowaliśmy się już w Zadaniu 01. Tym razem podejmiemy do niego jednak nieco poważniej. Po pierwsze: weźmiemy pod uwagę nie dwie, a dziesięć klas. Po drugie: skorzystamy z kolorowych fotografii. Po trzecie: jako klasyfikator wykorzystamy konwolucyjną sieć neuronową.

Uwaga! To zadanie może być obciążające obliczeniowo, a na finalne wyniki trzeba trochę poczekać. Jeżeli masz taką możliwość, skorzystaj ze sprzętowej akceleracji. Jeżeli obliczenia trwają bardzo długo - zmniejsz rozmiar wykorzystywanej sieci (np. wykorzystując dwukrotnie czy czterokrotnie mniejszą liczbę kanałów). Testowe obliczenia wykonuj tylko na małym podzbiorze całych danych. Z ich użyciem upewnij się, że wszelkie wykresy i obrazy są prawidłowo tworzone i zapisywane. Końcowe obliczenia zostaw na noc / na czas innej aktywności. Przede wszystkim zaś - zabierz się za zadanie nieco wcześniej niż kilka godzin przed ostatecznym terminem. Taka natura pracy z dużymi modelami..!


- W przypadku punktów oznaczonych ikoną  poinformuj w jaki sposób je zrealizowałeś - wspomnij kluczowe klasy/metody/funkcje lub załącz powiązany fragment kodu źródłowego.
- W przypadku punktów oznaczonych ikoną  załącz w raporcie obraz przedstawiający efekt danej operacji.
- W przypadku punktów oznaczonych ikoną  załącz w raporcie wykres przedstawiający trendy optymalizacji (uczenia) dla danych treningowych i testowych. W tym przypadku - jak zmieniała się skuteczność klasyfikacji (*accuracy*) i funkcja straty (*loss*).
- W przypadku punktów oznaczonych ikoną  umieść w raporcie informację ile zwykle trwało wykonanie jednej epoki treningu takiego modelu

Dane, dane, me królestwo za dane

Tego typu modele są opisywane przez tysiące niezależnych parametrów - a by znalezienie ich rozsądnych wartości było w ogóle możliwe, konieczne są naprawdę obszerne zbiory danych. W tym zadaniu skorzystamy z (również już legendarnego) zbioru CIFAR10 (<https://www.cs.toronto.edu/~kriz/cifar.html>) - zawierającego kilkadziesiąt tysięcy niewielkich kolorowych fotografii.

1. Twój ulubiony framework prawie na pewno powinien pozwalać na łatwe wczytanie tego zbioru. Wyświetl kilka jego losowych elementów by upewnić się, że ładowanie danych skończyło się sukcesem. Do jakich klas należą? Czy są różnorodne? [
2. Zamień etykiety przyporządkowane obrazom na wektory prawdopodobieństw (w takim formacie informacje zwracać będzie nasza sieć). Użyj do tego celu metody *one hot encoding*. Przykładowo:

```
3 -> [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

(czyli - jeżeli zdjęcie jest klasy o indeksie 3, to jest 100% prawdopodobieństwa że należy do czwartej z klas i 0% prawdopodobieństwa, że do którejś z pozostałych). [

Minimalna architektura

Nasza startowa mikro-architektura będzie się składała z jednego niewielkiego bloku konwolucyjnego oraz dodatkowej końcowej warstwy klasyfikującej. Powinna odpowiadać następującemu potokowi przetwarzania:

- przyjmij na wejście trójkolorowe obrazy 32x32;
 - przeskaluj wartości pikseli tak by zawierały się w zakresie [0,1] (zamiast [0.255]);
 - przetwórz obraz z użyciem 5 filtrów konwolucyjnych 3x3;
 - z paddingiem zachowującym rozmiar obrazu;
 - z aktywacją sigmoidą;
 - wynikiem powinien być obraz składający się z 5 nowych kanałów;
 - ponownie przetwórz obraz z użyciem 5 filtrów konwolucyjnych 3x3 (ten sam padding i aktywacja);
 - wejściem powinny być kanały wytworzone przez poprzedni segment filtrów;
 - zmniejsz rozdzielczość obrazu używając *max pooling* z okienkiem 8x8;
 - "spłaszcz" wynikowy tensor, zamieniając go na wektor wartości;
 - na koniec przetwórz ten wektor z użyciem 10 gęsto (*dense*) połączonych neuronów;
 - tym razem skorzystaj z aktywacji *softmax*;
 - każdy z tych neuronów reprezentuje prawdopodobieństwo, że wejściowy obraz przedstawiał klasę o danym indeksie.
3. Zaimplementuj taką architekturę w wybranej bibliotece (im bardziej wysokopoziomowo i obiektowo tym lepiej). [📖]
4. Ile wag/parametrów (które trzeba dostroić) składa się na taki model?
5. Podaj na wejście modelu kilka zdjęć. Jakie prawdopodobieństwa poszczególnych klas zwrócił na wyjściu?
6. Zoptymalizuj model tak, by był dobrym klasyfikatorem. [📖]
1. Skorzystaj z optymalizatora typu *stochastic gradient descent*.
 1. Przyjmij *learning rate* równe 0.001, a *momentum* równe 0.9 (małe kroki toczenia, duża "masa" wirtualnej kulki).
 2. Jako funkcję straty (*loss*) wykorzystaj entropię krzyżową (*categorical crossentropy*) - doskonale nadaje się do oceny jakości klasyfikacji.
 3. Optymalizuj wykorzystując tzw. *batche* (porcje) składające się z 64 zdjęć.
 4. Wykonaj 150 epok optymalizacji (epoka - jedna iteracja po całym zbiorze treningowym).
 2. Przygotuj wykres prezentujący jak zmieniała się zarówno wartość *loss* (straty) jak i *accuracy* (skuteczności) z epoki na epokę. [📈] [🕒]
 1. Pamiętaj, by wykres zawierał trend zarówno dla danych treningowych, jak i testowych (których nie używamy do optymalizacji). Te trendy będą się często różnić!
 2. Jeżeli wykres mocno "drga", to warto go "wygładzić" zastępując każdy wynik średnią z kilku ostatnich wyników (np. dziesięciu).
 3. W celu łatwego porównywania wyników postaraj się by **wszystkie** wykresy w raporcie miały taką samą skalę na osiach X (epoki) i Y (*loss/accuracy*).
 4. Warto najpierw zapisać gdzieś dane liczbowe - a dopiero potem "polerować" wygląd wykresu. Dzięki temu nie trzeba będzie powtarzać długich obliczeń!
 7. Jak teraz wyglądają zwracane prawdopodobieństwa?

What? NETWORK is evolving!

8. Zwiększ liczbę filtrów wykorzystywanych w obu warstwach konwolucyjnych z 5 do 20.
 1. Jak zmieniła się liczba parametrów modelu?
 2. Jak zmienił się czas trenowania?
 3. Jak zmieniły się rezultaty treningu? ☒ [🕒]
9. Sekwencja konwolucja->konwolucja->pooling tworzy tzw. blok konwolucyjny. Wielokrotna aplikacja takiego bloku stopniowo przetwarza obraz (dane mocno zależne od lokalizacji i struktury) w wektor cech (niezależnych od siebie).
 1. Zmień rozmiar okna *poolingu* z 8x8 na 2x2.
 2. Przygotuj funkcję, która dla zadanych parametrów zwraca cały taki blok - wykorzystamy ją by wielokrotnie umieszczać w potoku przetwarzania sekwencję konwolucja->konwolucja->pooling (wyjście z jednego bloku stanowi wejście do kolejnego). [🖥️]
 1. Niech parametrem bloku będzie liczba filtrów wykorzystywanych w warstwach konwolucyjnych.
 3. Zmodyfikuj model tak, by korzystał z dwóch bloków - w pierwszym jest po 20 filtrów w warstwie, w drugim po 40. [🖥️]
 1. Uruchom kilka-kilkanaście epok treningu takiej sieci. Czy efekty wyglądają obiecująco?
 2. Zamień aktywacje we wszystkich warstwach (poza ostatnią) na aktywację *ReLU*. [🖥️]
 1. Powinno to przyspieszyć trening - pochodna *ReLU* jest niezerowa w znacznie szerszym zakresie argumentów niż pochodna sigmoidy.
 2. To szczególnie istotne przy głębszych sieciach (a taką będziemy trenować).
 3. Wytrenuj dwublokowy model - jak teraz wyglądają rezultaty? ☒ [🕒]
 4. Dodaj jeszcze dwa bloki - teraz będzie ich łącznie cztery. Liczby filtrów to kolejno 20, 40, 80, 160.
 1. A teraz? Jak przebiega trening, jakie sieć osiąga wyniki? ☒ [🕒]
 10. Dodamy teraz kilka usprawnień. Części z nich nie omawialiśmy jeszcze na zajęciach - będą poruszone za tydzień. Na szczęście nie przeszkadza to w korzystaniu z ich gotowych implementacji.
 2. Po każdej warstwie konwolucyjnej (a przed następną warstwą) umieść warstwę *batch normalisation*. [🖥️]
 1. Stabilizuje ona wynikowy rozkład wartości, ułatwiając uczenie się kolejnych warstw.
 3. Jak teraz przebiegało uczenie? ☒ [🕒]
 11. Na koniec trzeba zadbać o to, by nasza sieć nie skupiała się na dopasowaniu do danych treningowych - a zamiast tego próbowała być nieco bardziej uniwersalna.
 1. Po każdej warstwie *poolingowej* (a przed wyjściem z bloku) umieść warstwę *dropout*. [🖥️]
 2. Warstwa ta w czasie treningu "ukrywa" losowo wybrane wartości, uniemożliwiając korzystanie z nich dalszym warstwom. Dzięki temu nie mogą one "nauczyć się" korzystania tylko z małej i specyficznej części danych - będą musiały być bardziej "elastyczne".
 3. Procent ukrywanych wartości jest parametrem takiej warstwy - a więc i całego naszego bloku. Zastosuj następującą sekwencję *dropout rates*: 0.1, 0.2, 0.3, 0.4 (na początku niskie, by nie tracić za wiele informacji, potem stopniowo coraz większe).
 4. Kolejny raz zbadaj proces uczenia. ☒ [🕒]

Mind the GAP !

(to już ostatnia sekcja - ale na tym etapie masz prawo odczuwać zmęczenie tematem! - przejdź się na spacer, zdrzemnij, albo chociaż napij herbaty)

12. Zostaje jeszcze sprawa obsługi fotografii o nietypowych wymiarach. W tym celu wykorzystamy operację *Global Average Pooling* (GAP).
 1. Zmodyfikuj definicję wejścia do sieci tak, by przyjmowała obrazy o dowolnym rozmiarze. [🖨]
 2. Ostatnią z użytych warstw typu *max pooling* zamień na warstwę typu GAP. [🖨]
 3. Ostatni już raz - zweryfikuj przebieg procesu uczenia. **Uwaga!** Postaraj się zapisać znalezione na koniec parametry modelu (większość bibliotek pozwala na jego bezpośrednie zapisywanie) - przydadzą się za chwilę. [☑] [🕒]
13. Przygotuj nowy obraz zawierający obiekt należący do jednej z 10 rozpoznawanych klas - ale taki, który nie jest kwadratowy. Przeskaluj go do podobnego rozmiaru jak obrazy używane do treningu - ale z zachowaniem jego własnych proporcji (czyli np. 28x32 albo 36x30 - zgadza się rząd wielkości, ale nie jest to 32x32). Czy w tym przypadku sieć również poprawnie rozpozna zawartość zdjęcia?
14. Przejrzyj wszystkie uzyskane wyniki (jest ich sporo na tym etapie). Co o nich myślisz? Jakie możesz wyciągnąć wnioski na temat stosowanych architektur?