



Zadanie 02



Wyciskanie soku

Celem ćwiczenia jest zapoznanie się z technikami klasyfikacji obrazów sieciami konwolucyjnymi oraz poszukiwania najlepszych ich konfiguracji. Zaimplementujemy klasyczną, głęboką, sieć konwolucyjną, a następnie dobierzemy jej hiperparametry przy użyciu biblioteki Optuna. Będziemy pracować na zbiorze CIFAR-10.

- W przypadku punktów oznaczonych ikoną  poinformuj w jaki sposób je zrealizowano - wspomnij kluczowe klasy/metody/funkcje lub załącz powiązany fragment kodu źródłowego.
- W przypadku punktów oznaczonych ikoną  załącz w raporcie obraz przedstawiający stan kanału będącego wynikiem danej operacji.

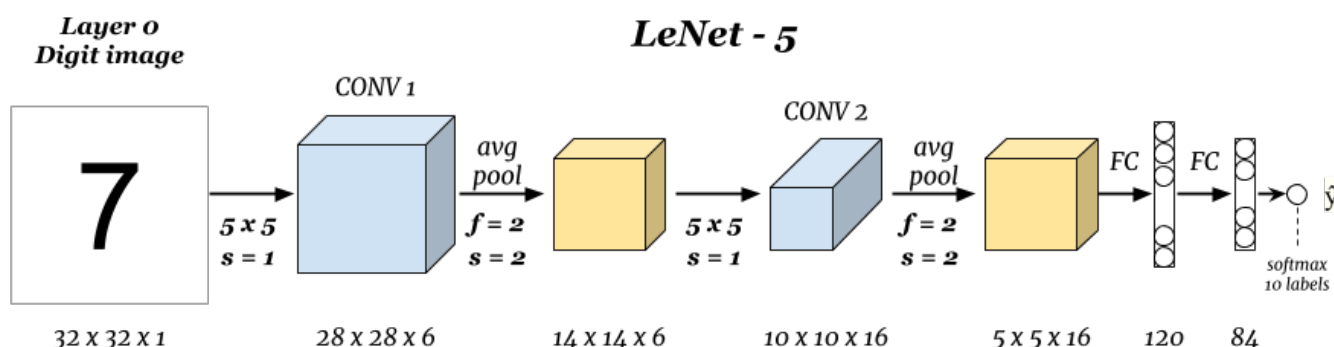
Wszytko się zaczyna i kończy na danych

Rozpoznawanie zawartości obrazka, jak każdy problem uczenia maszynowego zaczynamy od zorganizowania zbioru danych treningowych i testowych. Zbiór, który będziemy klasyfikować to CIFAR-10. Większość frameworków zawiera metody ułatwiające pobranie go jedną linią kodu.



- Sprawdź w jaki sposób w Twoim frameworku można załadować zbiór CIFAR-10. Pamiętaj o podziale na zbiór treningowy i testowy. Zwykle CIFAR-10 jest już podzielony na 50 000 przykładów treningowych i 10 000 testowych. 
- Wyświetl kilka obrazów z zestawu wraz z tekstową etykietą. 

Projektujemy architekturę sieci neuronowej

Kolejnym etapem będzie zdefiniowanie architektury sieci neuronowej do klasyfikacji obrazów. Jako podstawę przyjmijmy sobie klasyczny LeNet, ale możesz użyć wszelkich znanych ci trików i ulepszeń tej architektury, tak aby uzyskać lepsze wyniki. Pozostańmy jednak w świecie sieci konwolucyjnych (CNN) i gęsto połączonych (MLP) (nie-Transformery).




- Zbuduj swoją sieć korzystając z "bloczków" dostępnych w twoim frameworku. Rozwiązujemy problem klasyfikacji, a więc na wyjściu sieci chcemy mieć znormalizowane wartości pewności (ała prawdopodobieństwa.) z przedziału 0 - 1, sumujące się do 1. Zatem, na wyjściu z sieci powinno być 10 neuronów, których odpowiedź przepuszczamy przez funkcję softmax.

- Większość framework-ów udostępnia funkcje do narysowania grafu obliczeniowego modelu. Wyświetl teraz ten graf. Czasem jest to tak proste jak `print(model)`. Wypisz także ilość jego parametrów. 
- Zainicjalizuj sieć wartościami losowymi właściwymi dla użytych funkcji aktywacji. W przypadku ReLU będzie to *He initialization*. Niektóre frameworki domyślnie inicjalizują wagi sieci właściwą metodą. Sprawdź to i napisz jak jest w Twoim framework-u. 



Zapętlamy się w treningu

Mamy sieć, teraz czas na zdefiniowanie sposobu treningu sieci, tj. pętli uczącej.

- Trenujemy sieć za pomocą spadku po gradiencie (w pewnym ulepszonym wariancie). Kluczowe będzie określenie funkcji kosztu, z której ten gradient będziemy otrzymywać. Dla problemu klasyfikacji właściwą funkcją będzie entropia krzyżowa (Categorical Cross-Entropy Loss). Skonfiguruj taką funkcję.
- Będziemy monitorować proces treningu na podstawie *training loss* oraz *validation loss*. A dopiero po całym treningu raportujemy *test loss*. Zatem zadaj teraz o wydzielenie ze zbioru treningowego kawałka (10%) na potrzeby walidacji. Dane walidacyjne powinny zostać wylosowane z całej puli danych treningowych.
- Trening przeprowadzamy w pętli, małymi krokami iterując po zbiorze danych w mini-batch-ach. Pamiętaj o wymieszaniu danych na samym początku. Do wykonywania kroków uczenia wykorzystaj optymalizator Adam (lub jego wariant). Zaimplementuj taką pętlę, opisz jak działa.
- Sprawdź czy wszystko działa (uruchom trening). Twoja pętla ucząca powinna raportować co pewien interwał aktualny loss dla danych treningowych oraz loss dla danych walidacyjnych.
- Na koniec treningu zaraportuj wynik dokładności klasyfikacji dla wytrenowanej sieci na całym testowym datasetcie (10 000). `accuracy = (number of correct predictions) / (number of all predictions)`. W tym momencie model powinno się przełączyć w tryb inferencji (niezbędne gdy używamy np. Dropout). Podsumuj działanie całej pętli i podaj wynik bazowy accuracy. 

Hiperparametryzujemy sieć i jej trening

Na tym etapie powinieneś mieć działającą sieć konwolucyjną oraz całą maszynę do jej treningu. Jesteś w stanie osiągnąć pewien bazowy wynik dokładności klasyfikacji. Zapewne zrobiłeś też szereg założeń co do rozmiaru sieci, stałej uczącej (learning rate), regularyzacji itp. Robi się z tego całkiem nowa przestrzeń możliwych ustawień, przestrzeń hiperparametrów. Przeszukamy ją teraz, aby osiągnąć możliwe dobre wyniki.

- Wyodrębnij swój dotychczasowy kod jako funkcję, która na wejściu przyjmuje wartości hiperparametrów (wszystkich, które znajdziesz, ale koniecznie learning rate i rozmiar sieci definiowany np. jako ilość warstw i ich rozmiar), a zwracać będzie wynik accuracy dla zbioru testowego.
- Korzystając z biblioteki do poszukiwania hiperparametrów, np. Optuna, skonfiguruj przeszukiwanie przestrzeni hiperparametrów. Opisz jak to działa. 
- Teraz możesz dołożyć do swojego kodu wszelkie znane ci triki i ulepszenia, które mogą pozwolić uzyskać możliwie dobre wyniki. Jeżeli techniki, którymi się posługujesz (np. Dropout) mają jakieś hiperparametry to wyciągnij je do interfejsu funkcji trenującej. Opisz jakie ulepszenia stosujesz, jeżeli nie znasz żadnych to spróbuj skonfigurować klasyczną regularyzację L2. 

14. Puść tyle iteracji poszukiwania hiperparametrów na ile pozwala ci czas i dostępne zasoby obliczeniowe. Pamiętaj, aby zapisywać najlepsze konfiguracje hiperparametrów i wyniki dla nich. **Podaj najlepszą konfigurację i jej wynik.** Opisz swoje wnioski z tego doświadczenia. [🇺🇸]
15. Na zakończenie użyj swojej najlepszej sieci do klasyfikacji kilku ręcznie wybranych obrazków ze zbioru testowego. Wyświetl obrazy, ich prawdziwą etykietę oraz etykietę przewidzianą przez sieć. Wyświetl kategorie tekstowo, nie jako liczby. [🖼️]