




Zadanie 04




Zalewamy wrzółkiem chińską zupkę, czyli tym razem nasza sieć przygotowana jest z użyciem gotowej bazy

W poprzednim zadaniu korzystaliśmy z nieźle przygotowanego (oraz przede wszystkim bardzo dużego) zbioru danych CIFAR10. W praktyce zwykle nie dysponujemy takimi zasobami. Zasymulujemy tę sytuację pobierając zdjęcia z popularnej wyszukiwarki. Będzie ich niewiele (kilkaset) i będą bardzo nierównej jakości. W związku z tym trening oparty o tak mały zbiorek nie będzie zbyt efektywny - będziemy się więc posiłkować pre-treningiem z użyciem innych danych (a nawet "pożyczeniem" wstępnie wytrenowanej sieci). A tak w rzeczywistości jest to (znów!) pretekst by pobawić się kolejnymi mechanizmami związanymi z przetwarzaniem obrazów (zapisywanie i wczytywanie modeli, *fine-tuning*, obsługa zewnętrznych zbiorów danych, etc.).

- 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*).

Kaczko, kaczko, idź!

Pierwszym krokiem będzie przygotowanie nowego, zindywidualizowanego zbioru danych. W tym celu skorzystamy z małej biblioteczki umożliwiającej szybkie pobieranie takich danych z wyszukiwarki DuckDuckGo: `jmd_imagescraper`.

1. Pobierz kilkaset (maksimum to aktualnie 1000 - staramy się dążyć do tej liczby) obrazów należących do 3 różnych klas.
 1. Postaraj się by klasy były ciekawe/nieoczywiste (prowadzący korzystał z zestawu [zielone curry/spaghetti carbonara/ciasto marchewkowe]).
 2. Upewnij się, że żadna z tych klas nie występuje w zbiorze CIFAR, ani w zbiorze ImageNet (<https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a>) - w przeciwnym wypadku wyniki będą wypaczone.
 3. Przeglądaj pobrane obrazy, usuń najbardziej rażące przykłady niewłaściwej zawartości (nie spędź na tym zbyt dużo czasu! ;)).
2. Wczytaj obrazy z dysku tworząc nowy, prywatny zbiór danych.
 1. Twoja ulubiona biblioteka do operacji tensorowych i uczenia maszynowego prawdopodobnie oferuje gotowe klasy i metody pomocnicze! 
 2. Zapewnij wstępne przeskalowanie obrazów do rozmiaru 32x32 (to bardzo agresywne i trochę sztuczne obniżenie rozdzielczości, ale ułatwi nam wykonanie początkowych etapów zadania).
 3. Podziel zbiór na część treningową i walidacyjną w proporcjach 80%-20%. 
 4. Wyplotuj kilka elementów ze zbioru treningowego wraz z ich etykietami by upewnić się, że wszystko działa zgodnie z intencjami. 

Sieci konwolucyjne na bazie gotowej kostki rosołowej

3. Zaczniemy klasycznie. Skorzystajmy z najlepszej architektury spośród tych używanych w Zadaniu 03 i wytrenujmy jej parametry (od zera) na naszym świeżo przygotowanym zbiorze.

Jaki efekt udaje się uzyskać? ☒

1. Pamiętaj by zmienić liczbę wyjść z ostatniej warstwy tak, by pasowała do nowej liczby klas!

4. Teraz zrobimy pierwsze podejście do tzw. *transfer learning*. Wiele zbiorów danych składa się z elementów podobnego typu (np. realistycznych fotografii). To z kolei oznacza, że filtry konwolucyjne znalezione na skutek dopasowania parametrów do jednego zbioru danych (zwykle dużego i dobrej jakości, jak wcześniej używany CIFAR) często świetnie nadają się (z ewentualnie drobnymi korektami) do przetwarzania innego zbioru (np. tego zebranego na DuckDuckGo).

1. Wczytaj zapisany wcześniej model, który został dopasowany do problemu z Zadania 03 (skutecznie klasyfikuje elementy zbioru CIFAR).

1. Jeżeli zapomnieliśmy go zapisać, to niestety konieczne jest powtórzenie treningu (oczywiście tylko finalnej, najbardziej efektywnej aranżacji sieci).

2. Zamroź wszystkie jego wagi, tak by nie zmieniały się w trakcie uczenia.

1. Powinna to umożliwiać używana biblioteka. ☒

3. Upewnij się, że wykorzystywane warstwy typu *batch normalisation* pracują w trybie inferencji - czyli po prostu nie będą zmieniać ustalonych już oszacowań średniej i wariancji poszczególnych cech (w przeciwnym wypadku mogłyby zupełnie zepsuć efekty pre-treningu).

1. To też wspiera biblioteka, realizacja jest często powiązana z poprzednim punktem. ☒

4. Usuń z modelu ostatnią, gęsto połączoną warstwę (tą służącą do klasyfikacji). ☒

5. Zamiast usuniętej warstwy dodaj nową, jeszcze niewytrenowaną.

1. Jej wagi nie powinny być zamrożone - powinny móc się zmieniać! ☒

6. Wytrenuj na nowym zbiorze danych taką "pożyczoną sieć" (większość parametrów została już wstępnie ustalona dla poprzedniego zbioru, jedynie ostatnia gęsta warstwa jest trenowana "od zera"). Jak teraz wypadają wyniki? ☒

5. Dla naszego nowego zbioru danych to może być jednak za mało (w końcu bazowa sieć radziła sobie z CIFARem tylko przyzwoicie - ale daleko jej było do rekordów)! Powtórzmy więc ten manewr, ale wykorzystując jako bazę sieć, która była trenowana na znacznie większym i bardziej różnorodnym zbiorze (np. na legendarnym ImageNet - <https://www.image-net.org/index.php> - być może najważniejszym zestawie danych treningowych w całej historii współczesnego rozpoznawania obrazów). A najlepiej taką sieć, która ma sprawdzoną architekturę i osiągała już wcześniej sukcesy.

1. Zwiększmy wejściowy rozmiar obrazów z 32x32 do 256x256 - nowa sieć bazowa pozwoli nam na efektywne wykorzystywanie tak dużych obserwacji.

2. Na potrzeby tego zadania skorzystamy z sieci Xception, zaprezentowanej pierwszy raz na CVPR2017.

1. Ciekawi szczegółów mogą poczytać o niej więcej tutaj: <https://arxiv.org/abs/1610.02357> (była to głęboka sieć z połączeniami residualnymi i sprytną realizacją konwolucji, która dzieli ich obliczanie na dwa niezależne etapy).





2. Trzeba zdobyć (pobrać) gotowy model, a w szczególności gotowe wartości parametrów.

1. Jeżeli używamy TensorFlow, to można go dorwać choćby tu: <https://keras.io/api/applications/xception/> .
2. Użytkownicy PyTorch'a znajdują odpowiedni model np. tutaj: <https://github.com/Cadene/pretrained-models.pytorch#xception> . Jeżeli to za dużo ambarasu, to można skorzystać z dowolnego zamiennika z <https://pytorch.org/vision/stable/models.html> - byleby używany model kończył się GAPem.
3. Usuńmy z pobranego modelu wszystkie warstwy od GAP wzwyż (czyli sam GAP i wszystko co po nim).
 1. Używany framework zwykle mocno pomaga w takich operacjach.
4. Dopilnujemy, by jego wagi były zamrożone, a wszelkie moduły normalizacyjne nie zmieniały swoich estymat. Upewnijmy się też, że jest gotowy na wejście w rozmiarze 256x256x3.
5. Wykorzystajmy go zamiast sieci z Zadania 03 do stworzenia nowego klasyfikatora.
 1. Potok przetwarzania powinien wyglądać mniej-więcej tak: `wejście -> Xception ["zamrożone" wagi, usunięte końcowe warstwy] -> GAP -> warstwa gęsto połączona [zupełnie nowa, w pełni gotowa do treningu] -> wyjście`. [📄]
3. Wytrenujemy taki klasyfikator (a właściwie jego ostatnią warstwę). Powinno wystarczyć co najwyżej kilkadziesiąt epok. Jakie tym razem uzyskaliśmy efekty? Zapiszmy gdzieś stan sieci na koniec treningu. [📄]
4. Ten wynik da się jeszcze zwykle odrobinę poprawić, stosując tzw. *fine-tuning* - korektę pobranych gotowych wag tak, by pasowały do naszego problemu.
 1. Wczytajmy ponownie sieć z poprzedniego punktu. "Rozmroźmy" wagi wszystkich jej warstw, umożliwiając im teraz ewolucję w czasie uczenia. [📄]
 2. Mimo to, dopilnujemy by ewentualne *batch normalisation* nadal były w trybie inferencji! Zmiany zaszytych wewnątrz nich estymat mogłyby kompletnie zniszczyć wszystkie efekty dotychczasowego uczenia! W efekcie *batch normalisation* będą w ciekawym stanie "pośrednim" - wagi odpowiedzialne za skalowanie wyników będą mogły się zmieniać, ale wykorzystywane wartości średniej i wariancji już nie. [📄]
 3. Zmniejsz prędkość uczenia się (*learning rate* lub podobny parametr) - nawet 10-krotnie! Będziemy przecież robić tylko drobne korekty.
 4. Puść dodatkowe kilkanaście-kilkadziesiąt epok takiego powolnego douczania. Czy udało się wycisnąć z modelu jeszcze trochę efektywności? Zapisz gdzieś wynikowy model (przyda się w ostatniej sekcji). [📄]

No dobra, ale w sumie na jakiej podstawie taka sieć podejmuje decyzje?

Kompleksowa odpowiedź na to pytanie wymaga zaznajomienia się z całą dyscypliną naukową, jaką jest badanie interpretowalności modeli decyzyjnych. W tym przypadku zadowolimy się bardziej zgrubnym zrozumieniem tego co się dzieje, w sprytny sposób wykorzystując naturę warstwy GAP.

6. Przygotujmy kilka-kilkanaście fotografii, które nasza sieć klasyfikuje poprawnie. Najlepiej, jeżeli będą mieć różnorodną zawartość.
7. Dla pierwszej z nich wykonajmy poniższe kroki.
 1. Przyjrzyjmy się bezpośrednio temu, co zwraca wykorzystywana sieć bazowa Xception. Jeżeli z wymiarami wszystko poszło zgodnie z planem, to powinny być to tensory zawierające 2048 kanałów, każdy z nich o rozdzielczości 8x8.

1. Podaj na wejście sieci wybraną fotografię i podejrzuj zawartości kilku-kilkunastu z tych 2048 kanałów. Jak wyglądają ich aktywacje? 
2. Kanały te można zsumować wagowo (wagi pochodzą z końcowych neuronów aktywacyjnych). Taka suma tworzy prostą mapkę, która pokazuje jak duży był wpływ danego obszaru zdjęcia na finalną klasyfikację. 
1. Czyli w praktyce: $[\text{zawartość kanału 0 przed GAP}] * [\text{waga związana z kanałem 0 w tym neuronie ostatniej warstwy gęsto połączonej, który zapalił się najmocniej (który odpowiada za tę klasę, do której została przydzielona fotografia)}] + [\text{zawartość kanału 1 przed GAP}] * [\text{waga związana z kanałem 1 w tym neuronie ostatniej warstwy gęsto połączonej, który zapalił się najmocniej}] + \dots$ (i tak dla wszystkich 2048).
2. To samo, ale jeszcze krócej: mnożymy zawartość kanałów przed GAPem przez odpowiadające im współczynniki w wybranym neuronie wyjściowym i sumujemy uzyskane wyniki w jeden kanał.
3. Obejrzyj uzyskaną *heatmapę*. Jak ma się do tego, co znajduje się na fotografii?
8. Wykonaj w ten sam sposób *heatmapy* dla wszystkich wybranych obrazów. Porównaj obszary na których koncentrowała się sieć z ich rzeczywistą zawartością. 
9. Poszukajmy tych obrazów, dla których nasza sieć się myli.
 1. Wyplotujmy kilka-kilkanaście z nich. Czy to typowe elementy danej klasy? Czy człowiek mógłby je prawidłowo rozpoznać? 
 2. Dla tych obrazów również przygotuj oparte o GAP *heatmapy* istotności. Czy pomagają w zrozumieniu, na jakiej podstawie sieć zachowała się w błędny sposób? 