



# Zadanie 06

## Konwolucje jednak niepotrzebne...


Konwolucyjne sieci neuronowe działają bardzo dobrze w zadaniu rozpoznawania obrazów i przez długi czas królowały w benchmark-ach. W ostatnim czasie jednak, zrodziła się pewna rewolucja w podejściu do przetwarzania obrazów zainspirowana osiągnięciami z dziedziny przetwarzania języka naturalnego. Chodzi o tzw. Transformatory, czyli sieci neuronowe z mechanizmem uwagi - Self-Attention. Wykazano, że mechanizm ten może, całkowicie zastąpić konwolucje w zadaniach rozpoznawania obrazów.

Dziś zaimplementujemy sobie takiego Transformera i zbadamy jego działanie.

- 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.

## Przygotowanie danych


Do treningu użyjemy zbioru CIFAR-10 urozmaicając go za pomocą augmentacji. Augmentacja ma na celu zmniejszenie prawdopodobieństwa overfittingu. Modyfikujemy obrazy wycinając ich fragmenty, biorąc odbicie lustrzane i tym podobne operacje, aby urozmaicić zbiór i wprowadzić do niego nieregularności.

1. Przygotuj następującą sekwencję transformacji danych: RandomHorizontalFlip -> RandomResizedCrop -> Normalize. Opisz czego używasz. 

RandomHorizontalFlip - z prawdopodobieństwem 0.5 odwraca obraz w osi y, tj. robi odbicie lustrzane (a nie odwraca do góry nogami).

RandomResizedCrop - wycina z obrazu losowy fragment i skaluje go spowrotem do wielkości oryginalnego obrazu. Rozmiar wycinka określ na przedział  $<0.8 - 1.0>$  wielkości oryginału, a aspect ratio między 0.9, a 1.1.

Normalize - zastosuj normalizację do średniej 0 i wariancji 1 (dla każdego kanału osobno). Do tej operacji wyciągnij wartości średniej i odchylenia standardowego ze zbioru treningowego.

2. Tak przygotowaną procedurę augmentacji przypnij w miejsce odpowiednie dla Twojego framework-a. Może to być Data Loader albo część modelu. Opisz jak to się robi w twoim framework-u. 
3. Pobierz, załaduj i przygotuj zbiór danych CIFAR-10 do treningu. (1 pkt.)

## Cięcie obrazu na fragmenty

Sieci typu transformer operują na zbiorach danych, np. na zdaniach złożonych z sekwencji słów, tworzących jedną zrozumiałą całość. Dzięki temu sieć podczas inferencji może kojarzyć ze sobą poszczególne elementy zbioru, tak aby wyciągnąć z nich sensowną wiedzę.

Aby użyć transformerów do przetwarzania obrazów musimy każdy obraz zamienić jakoś na sekwencję danych. W tym celu będziemy je kroić na kawałki (patch-e) i z tych kawałków robić sekwencję. Coś w stylu zamiany obrazu na zdanie.

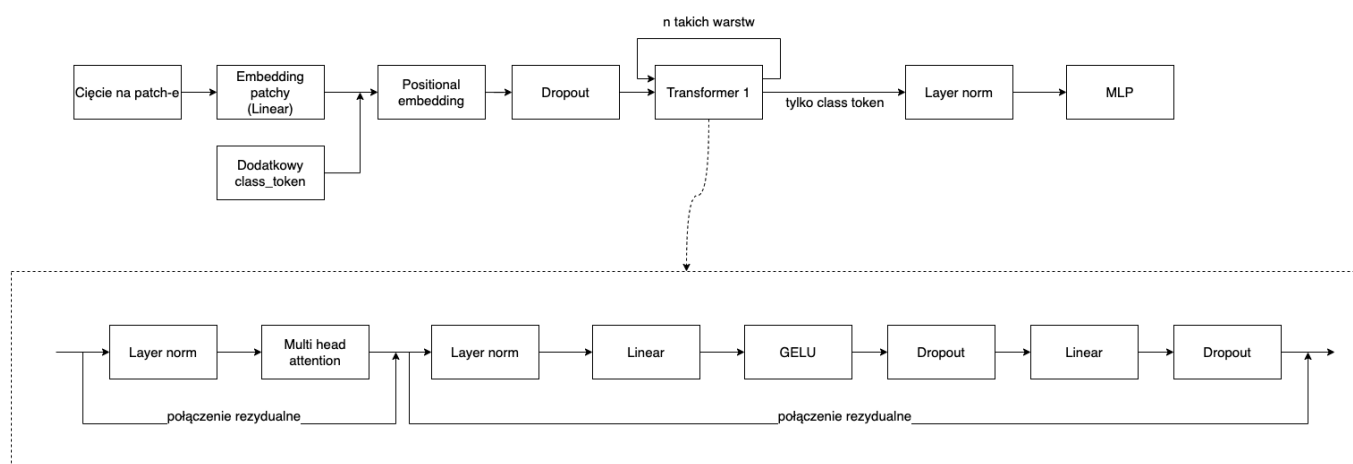
4. Przygotuj procedurę zamieniającą batch obrazów na batch sekwencji patch-y. Opisz jej działanie. 



Poszczególne patch-e nie powinny na siebie nachodzić. Procedura powinna móc przygotować patch-e o dowolnym rozmiarze (ale kwadratowe), tj. jej argumentem powinien być rozmiar patch-a.

5. Przetestuj działanie procedury tnącej na patche, zwizualizuj wynik (wyświetl patch-e). 

## Definicja modelu

Stworzymy sobie model Visual Transformera jak w artykule "An Image is Worth 16X16 Words", z jedną modyfikacją. Layer norm zastosujemy przed warstwami liniowymi, a nie po połączeniach rezydualnych. Schemat modelu powinien być taki jak na rysunku poniżej.



6. Zdefiniuj model sieci według diagramu. Opisz w jaki sposób go tworzysz, jeśli się da wyświetl jego computation graph, dla pewności, że wszystko jest połączone jak należy.  

Do prawidłowej implementacji będziesz potrzebować warstwy typu: multihead attention, liniowych, layer norm, dropout. Oprócz tego potrzebne będą połączenia rezydualne, a jako funkcje aktywacji użyjemy GELU. Nie zapomnij także o dodatkowym tokenie na pozycji zerowej pełniącym funkcję class token-a. Na podstawie tego tokenu będzie wykonywana klasyfikacja.

Skorzystaj z gotowych implementacji warstwy multihead attention dostępnych w twoim framework-u.

Pierwszym krokiem w forward pass powinno być zamienianie obrazu na sekwencję patch-y o wielkości 4.

Patche należy zamienić na embeddingi przy pomocy warstwy liniowej. Wielkość tensora wyjściowego z embeddingu powinna wynosić 256. Do embeddingów patch-y dodaj positional embedding (tensor do nauczania).

Każda warstwa ukryta (liniowa) w bloku transformera powinna mieć wielkość 512. Natomiast multihead-attention powinna mieć 8 "głów". Warstwa transformera na wejściu bierze tensor o wielkości 256 i na wyjściu też daje tensor o wielkości 256, czyli operuje na rozmiarze takim jaki ustaliliśmy dla embeddingu. W sumie potrzeba 6 bloków typu transformer, jeden po drugim.

Ostatnim elementem modelu jest pojedyncza warstwa gęsta/liniowa (poprzedzona layer norm), która jest połączona jedynie z class token.

Wszystkim dropout-om ustaw prawdopodobieństwo zerowania połączeń na 0.2.

# Trening


7. Wytrenuj tak przygotowaną sieć korzystając z poniższych ustawień. Puść 160 epok. (Jeżeli nie masz GPU, zredukuj ilość epok)

Aby przyspieszyć trening naszej sieci posłużymy się optimizer-em AdamW, który jest modyfikacją optimizer-a Adam. Użyjemy także learning rate scheduler, tj. procedurę, która w epoce nr 100 i 150 pomnoży learning rate o 0.1.

Jako loss function użyj Cross Entropy (categorical).

Na wyjściu z naszej sieci mamy wektor liczb. Największa z nich wyznacza klasę jaką sieć przypisuje obrazowi. Tzn. aby wyznaczyć indeks klasy obrazu należy wykonać operację argmax na wektorze wynikowym sieci.

*Jeżeli trenujesz na mniejszej liczbie epok zmodyfikuj milestone-y schedulera. Jeżeli twój framework nie udostępnia tego typu LR schedulera to użyj innego, możliwie podobnego.*

8. Przedstaw wyniki wytrenowanej sieci. Oprócz podania wyniku liczbowego sprawdź na kilku przykładowych obrazach. 🖨️ 
9. [Bonus dla zainteresowanych, + 5 pkt] Zwizualizuj uwagę sieci na wybranym obrazku przy pomocy metody Attention Rollout. Opisz jak to działa. 🖨️ 