Raport końcowy

Projekt realizowany w ramach przedmiotu Techniki Obrazowania Medycznego 2019/2020

Stanisław Kaczmarski
Angelika Kiełbasa
Maja Kałwak

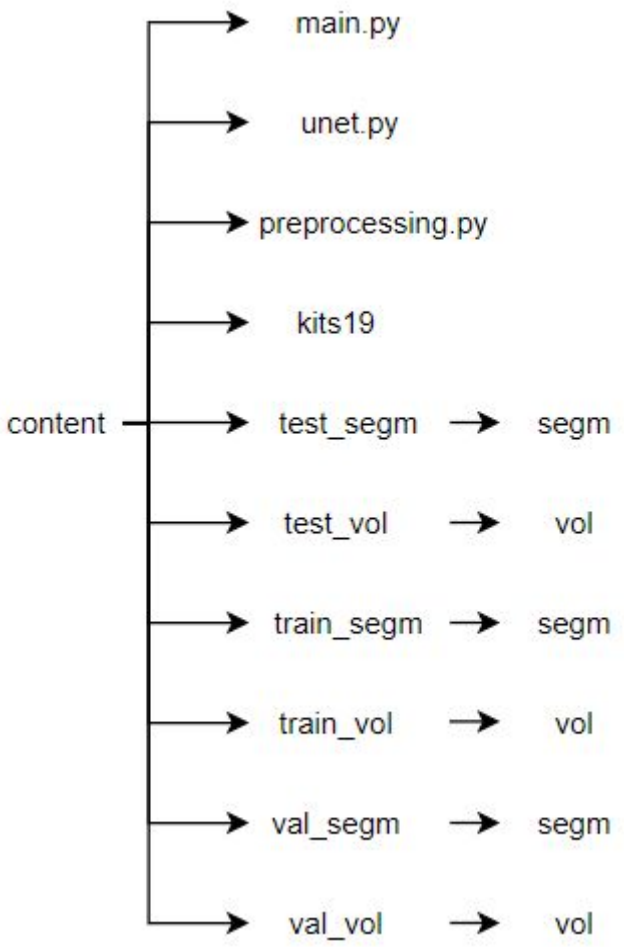
1. Przygotowanie danych

1.1 Otwieranie i pobieranie danych

Dane pacjentów pobrano na colab google wykorzystując kod podany na githubie challengu. W każdym folderze (case) znajdowały się imaging.nii.gz oraz segmentation.nii.gz. Zauważono, że foldery od 210 do 299 nie mają pliku segmentation.nii.gz. Po otwarciu wybranych zdjęć klatki piersiowej, uwidocznione były nerki (kolor niebieski) oraz guz (kolor czerwony).

1.2 Reorganizacja struktury danych

Opracowano nową strukturę danych, dzieląc je na zbiór testowy (test_segm, test_vol), treningowy (train_segm, train_vol), i walidacyjny (val_segm, val_vol). Do każdego z podzbiorów przypisano volume (vol), czyli zdjęcia uzyskane z tomografii komputerowej i segmentation (segm), czyli maski, na których kolorem czarnym (0) oznaczono wartość tła, kolorem szarym (1) wartość nerki, natomiast na biało (2) oznaczono guz. Każdemu zdjęciu np. z vol z podzbioru train_vol odpowiada zdjęcie z segm z pozdbioru train_segm. Analogicznie dla innych podzbiorów. W tym celu zostały stworzone funkcje generate_volume i generate_segm.



Rysunek 1. Przeorganizowana struktura danych.

1.3 Preprocessing

Preprocessing obejmuje przeskalowanie obrazów do rozmiaru 256×256, normalizację obrazów volume oraz odpowiednie przygotowanie generatora danych - funkcja ImageDataGenerator. Zastosowano augmentację, która powoduje rotację losowo wybranych zdjęć o 5°, oraz przesuwa je o 0.15 względem osi x i y.

2. Architektura sieci

Do wykonania projektu, wykorzystano algorytm U-Net, który łączy warstwy enkodera z mapami obiektów dekodera tworząc strukturę przypominającą literę U.

Enkoder zbudowano z 5 bloków. Jeden blok zawiera 2 warstwy konwolucyjne (Conv2D), warstwę Dropout i MaxPooling2D. Zastosowano okna splotu o wymiarach 3×3. Wraz z głębokością sieci, ilość filtrów wzrasta od 16 do 256. Warstwa Dropout odrzuca losowo zadaną ilość węzłów. W naszym przypadku wartość ta dla bloku 1 - 2 wynosi 0,1, dla bloku 3 - 4 wynosi 0,2, a dla bloku 5 równa jest 0.3. Dzięki temu sieć staje się mniej

wrażliwa na określone masy neuronów. To z kolei sprawia, że jest ona w stanie lepiej uogólniać i minimalizować nadmiernie dopasowanie, przez to zapobiega powstawaniu overfittingu.

Dekoder natomiast, składa się z 5 bloków. Zawiera warstwę Conv2DTranspose, warstwę concatenate i nie posiada MaxPooling2D. Wykorzystano warstwę dekonwolucją, ponieważ daje możliwość zastosowania transformacji zmierzającej w przeciwnym kierunku niż regularny splot.

Do optymalizacji modelu wykorzystano optimizer *adam.*

3. Napotkane problemy i ich rozwiązania

3.1. Niska wartość parametrów treningowych

W przypadku implementacji prototypu rozwiązania w kodzie *unet.py* przez nieuwagę nie dodano odpowiedniej liczby warstw uczących - dekoder składał się jedynie z 4 bloków. Ograniczyło to znacznie ilość parametrów treningowych z 4 mln do ok. 1 mln przez to pogarszając umiejętności predykcyjne sieci.

3.2. Zapis danych

Funkcja *plt.imsave()* pomimo ustawienia argumentu *cmap='gray'* po odczytaniu tak zapisanego zdjęcia, jego rozmiar wynosi (Height x Width x 4). Stąd też próbowano zapisu z wykorzystaniem *imageio.imwrite()*, które zapisywało rzeczywiście obrazy w skali szarości. Jednakże później z tego powodu pojawiły się problemy z wykorzystaniem *Conv2D* dla tak zapisanych obrazów, bowiem według dokumentacji powinna ona działać dla obrazów RGB. Jednocześnie był to również problem z zapisem w notacji one-hot. Przez to wrócono do *plt.imsave().* Prawdopodobnym źródłem niekompatybilności (różnych rozmiarów tablic) dla *dice_coef()* było użycie *color_mode='grayscale'* w preprocessingu dla *ImageDataGenerator.flow_from_directory()* uniemożliwiając sumowanie konkretnych wartości warstw. Ostatecznie zmieniono też ilość kanałów IMG_CHANNELS = 3, co znacznie wydłużyło trenowanie sieci (z ok. 5 min./epokę do ponad 1 godz./epokę). Prawdopodobnie ma to również związek z wykorzystaniem *dice_coef_multilabel()* do funkcji straty.

3.3. Notacja one-hot

Miała być realizowana za pomocą ustawienia parametru *class_mode='categorical'* jednakże ponownie pojawił się problem z kompatybilnością rozmiarów zdjęć. O ile zapewnia dużą wygodę w sumowaniu i interpretacji, to rozmiar macierzy takiej notacji uniemożliwiał jej zastosowanie. Różniły się bowiem *y_pred* oraz *y_true.* Niestety nie udało się ustalić dokładnej przyczyny i zmieniono koncepcję wykorzystując funkcję *multilabel_dice_loss* i zmieniając wyjściową funkcję aktywacji na *'softmax'.* Z perspektywy czasu prawdopodobnie lepszym rozwiązaniem byłby zapis maski w notacji one-hot w postaci pliku oraz

wykorzystanie notacji nerek oraz guza w oddzielnych sieciach wykorzystując *dice_coef* binarnie. Nie tylko przyspieszyłoby to obliczenia, ale też ułatwiło interpretację.

3.4. Niewłaściwa budowa drzewa plików

W prototypie rozwiązania przedstawiona forma drzewa plików była nieodpowiednia w wykorzystaniu *ImageDataGenerator*. Generowała bowiem klasę dla każdego pacjenta. Takie podejście możnaby wykorzystać dla oddzielnych masek nerki i guza jako 2 oddzielnych klas. Ich rozdzielenie można zrealizować wykorzystując notację one-hot.

3.5. Jednostki Hu na grayscale

Pierwotnie odczytane dane były zapisane w jednostkach Hounsfielda, jednak jak się później okazało zapis za pomocą *plt.imsave()* miał przekształcać na 'grayscale' automatycznie i dodatkowo normalizować zakres wartości do 0-255 (również maski).

3.6. Multilabel dice coefficient

Zaproponowane w prototypie rozwiązanie funkcji straty nie uwzględniało większej ilości etykiet maski zwracając nieodpowiednie do interpretacji wyniki. Wykorzystano zatem *dice_coef_multilabel()* dostosowując odpowiednie rozmiary zdjęć. Takie rozwiązanie uwzględnia 3 etykiety - tło, nerkę oraz guz. Specyfiką takiej funkcji straty są ujemne wartości metryki.

3.7. Colab Research Google

Projekt realizowano w środowisku Colab wykorzystując dostępne tam środki. Jednak to środowisko praktycznie uniemożliwia wprowadzanie zmian we wprowadzonych wcześniej modułach. Dodatkowo często przy dłuższym treningu sieci nastąpiło rozłączenie i utrata danych co znacznie wpłynęło na czas rozwiązania. Stąd oraz z problemów z kompatybilnością wynikły niżej opisane wprowadzone uproszczenia. Na przyszłość unikając tego problemu należałoby realizować implementację na własnym sprzęcie lub nie realizować kodu modułowo. Dodatkowo warto wspomnieć o dziwnej zależności czasu obliczeń - często bez jawnego powodu czas obliczeń dla takiego samego kodu był znacznie większy.

3.8. Dodatkowy strumień danych

Wymagający czasowo proces uczenia był niejednokrotnie przerywany przez niezidentyfikowany błąd dodatkowego strumienia danych. Wykorzystanie:

```
7 from PIL import Image, ImageFile
8 ImageFile.LOAD_TRUNCATED_IMAGES = True
```

wyeliminowało ten problem.

3.9. Ograniczenie czasowe

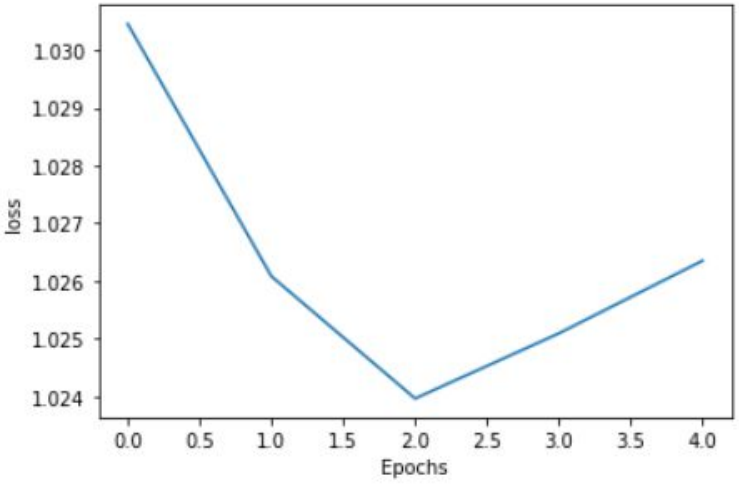
Wyżej wymienione problemy i ich rozwiązania znacznie wpłynęły na dostępny czas rzeczywistej realizacji celu projektu. Głównie ze względu na nagłe zwiększenie potrzebnego czasu trenowania sieci oraz rozłączenia z Colab'em wprowadzono wiele uproszczeń mających zapewnić jedynie przykładową realizację projektu:

- liczbę case'ów treningowych zmniejszono do 30, walidacyjnych do 15, a testowych do 10
- zwiększono batch_size do 128
- zmniejszono liczbę epok do 5
- zmniejszono rozmiary zdjęć do 128 x 128 x 3
- usunięto po 1 warstwie enkodera i dekodera
- dokonano predykcji jedynie dla 1118 zdjęć
- nie dokonano predykcji dla osobnych przypadków pacjentów

4. Ewaluacja

Niestety w procesie nie uzyskano wyników, które umożliwiają wykonanie ewaluacji. Segmentacja nie udała się, przez co wizualizacja danych nie jest możliwa - nie otrzymano oczekiwanych rezultatów. Poniżej opisano kroki, które zostałyby podjęte w przypadku uzyskania odpowiednich rezultatów.

Dla każdego pacjenta należałoby stworzyć osobny folder, który byłby traktowany jako oddzielny zbiór i dla tego zbioru powinno wyznaczyć się parametry takie jak: średnia, odchylenie standardowe, wartość maksymalna i minimalna (policzona oddzielnie dla nerki i nowotworu). Dodatkowo zwizualizowano by uzyskane rezultaty.



Rysunek 2. Wykres epok od funkcji straty.

Na wykresie widać, że funkcja początkowo maleje, a następnie wzrasta. Co może być objawem nieprawidłowości sieci - algorytm powinien dążyć do minimalizowania funkcji straty.

Poniższe zdjęcia prezentują proces uczenia się danej wielkości zbioru - od najszybszego do najwolniejszego.

```
WARNING:tensorflow:From <ipython-input-7-6356d5a1379b>:56: Model.fit_generator (from tensorflow.python.keras.engine.traini
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/5
53/52 [==============================] - 1251s 24s/step - loss: 1.0297 - dice_coef_multilabel: -0.0296 - val_loss: 1.0389
Epoch 2/5
53/52 [==============================] - 1242s 23s/step - loss: 1.0249 - dice_coef_multilabel: -0.0249 - val_loss: 1.0390
Epoch 3/5
53/52 [==============================] - 1233s 23s/step - loss: 1.0249 - dice_coef_multilabel: -0.0249 - val_loss: 1.0388
Epoch 4/5
53/52 [==============================] - 1240s 23s/step - loss: 1.0249 - dice_coef_multilabel: -0.0249 - val_loss: 1.0382
Epoch 5/5
53/52 [==============================] - 1238s 23s/step - loss: 1.0249 - dice_coef_multilabel: -0.0249 - val_loss: 1.0394
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/resource_variable_ops.py:1817: callir
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
Epoch 1/5
32/31 [==============================] - 1634s 51s/step - loss: 1.0305 - dice_coef_multilabel: -0.0305 - val_loss: 1.0330 - val_dice_coef_multilabel
Epoch 2/5
32/31 [==============================] - 1637s 51s/step - loss: 1.0261 - dice_coef_multilabel: -0.0261 - val_loss: 1.0296 - val_dice_coef_multilabel
Epoch 3/5
32/31 [==============================] - 1613s 50s/step - loss: 1.0240 - dice_coef_multilabel: -0.0238 - val_loss: 1.0285 - val_dice_coef_multilabel
Epoch 4/5
32/31 [==============================] - 1627s 51s/step - loss: 1.0251 - dice_coef_multilabel: -0.0251 - val_loss: 1.0278 - val_dice_coef_multilabel
Epoch 5/5
32/31 [==============================] - 1632s 51s/step - loss: 1.0264 - dice_coef_multilabel: -0.0264 - val_loss: 1.0267 - val_dice_coef_multilabel
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/resource_variable_ops.py:1817: calling BaseResourceVariable.__i
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Non-trainable params: 0
```

```
WARNING:tensorflow:From <ipython-input-6-a5ff0ae3ff94>:56: Model.fit_generator (from tensorflow.python.keras.engine.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/10
166/165 [==============================] - 4257s 26s/step - loss: 1.0298 - dice_coef_multilabel: -0.0298 - val_loss:
Epoch 2/10
166/165 [==============================] - 4254s 26s/step - loss: 1.0289 - dice_coef_multilabel: -0.0289 - val_loss:
Epoch 3/10
155/165 [===========================>..] - ETA: 3:50 - loss: 1.0289 - dice_coef_multilabel: -0.0289
```

5. Podsumowanie

Wymiary obrazu wejściowego pierwotnie wynosić miały $256 \times 256 \times 3$. Wybrano takie wymiary, ponieważ większy rozmiar mógłby zajmować zbyt dużo pamięci, natomiast zbyt mały mógłby spowodować utratę informacji. Jednakże podczas pracy, napotkano liczne problemy opisane powyżej i zmieniono koncepcję. Ostatecznie wymiary zostały zmniejszone do $128 \times 128 \times 3$ w celu przyspieszenia obliczeń.

W naszej opinii znacznie lepszy wynik dałoby zastosowanie dwóch sieci. Oddzielnej dla nerki i dla guza. Prawdopodobnie czas pracy uległby skróceniu. Dodatkowo takie rozwiązanie, zniwelowałoby problem nieproporcjonalności zbioru tła do zbioru zbioru masek nerek i guza. Umożliwiłoby zastosowanie tradycyjnej prostej formy funkcji straty (dice_coef), bez zastosowania multilabel. Takie ponowne podejście obejmowałoby:

- zapis danych volume w odpowiednim folderze w formacie (512×512) - funkcja *imageio.imwrite()*
- odczytanie masek segmentacji, standaryzację HU, zapis w notacji one-hot i oddzielne zapisanie masek nerki i guza w odpowiednich folderach

- wykorzystanie *ImageDataGenerator* do augmentacji danych, wyróżniając odpowiednie maski z folderów zgodnie z implementacją tej klasy
- implementacja architektury sieci dla podanego rozmiaru danych
- implementacja binarnej wersji *dice_coef()* i funkcji straty
- ilościowa analiza wyników poprzez obliczenie *dice_coef()* dla oddzielnych pacjentów, obliczenie średniej, odchylenia standardowego oraz wskazanie najlepszej, średniej i najgorszej segmentacji
- scalenie wyników segmentacji

Tak naprawdę, do końca nie wiadomo co jest faktyczną przyczyną niepowodzenia. Podczas pracy nad projektem podejmowano decyzje, które według nas, w tamtym momencie, były najlepsze. Gdybyśmy mieli podejść do problemu drugi raz, zastosowano by kroki opisane powyżej. Niemniej jednak projekt ten pozwolił nam zagłębić się w sieci neuronowe oraz zagadnienia związane z segmentacji obrazów. Pozwolił na zrozumienie algorytmu U-Net. Dodatkowo poznano różne funkcję biblioteki Keras m.in ImageDataGenerator.

6. Bibliografia

[1] https://kits.lib.umn.edu/
[2] https://grand-challenge.org/
[3] https://github.com/neheller/kits19
[4] https://github.com/zhixuhao/unet
[5] https://keras.io/
[6] https://github.com/keras-team/keras/issues/9395
[7] Chen, Joseph, and Benson Jin. "A 2D U-Net for Automated Kidney and Renal Tumor Segmentation." (2019).
[8] Paolucci, Iwan. "Kidney tumor segmentation using a 2D U-Net followed by a statistical post-processing filter." (2019).

7. Rozkład pracy

Angelika Kiełbasa - opis metod - raport, preprocessing, pomysł na ewaluację wyników, raport

Maja Kałwak - opis danych, research preprocessingu, preprocessing, opis metod - raport, raport

Stanisław Kaczmarski - implementacja architektury sieci, implementacja main, nadzór merge-request, opis metod - raport, raport