Akademia Górniczo-Hutnicza im Stanisława Staszica
Wydział Automatyki, Elektroniki Informatyki i Inżynierii Biomedycznej

Automatyczna segmentacja nerek i nowotworów nerek w obrazach tomografii komputerowej

PROJEKT REALIZOWANY W RAMACH PRZEDMIOTU
TECHNIKI OBRAZOWANIA MEDYCZNEGO

Małgorzata Sosin
Krzysztof Kwaśniak
Piotr Sumara

Kraków, 22 czerwiec 2020

Spis treści

Słowa kluczowe:
kidney tumor, segmentation, neural networks, U-Net, data preprocessing

1. Wstęp

Każdego roku pojawia się ponad 400 000 nowych przypadków raka nerki , a operacja jest jego najczęstszym leczeniem. Ze względu na dużą różnorodność morfologii nerki i guza nerki istnieje obecnie duże zainteresowanie tym, w jaki sposób morfologia guza odnosi się do wyników chirurgicznych, a także w rozwijaniu zaawansowanych technik planowania chirurgicznego. Automatyczna segmentacja jest obiecującym narzędziem do wykrywania nowotworów nerki, jak i innych narządów.

Tematem projektu jest jeden z problemów zaprezentowanych na konferencji naukowej MICCAI 2019 - tj. automatyczna segmentacja nerek i nowotworów nerek w obrazach tomografii komputerowej. Na stronie wyzwania udostępniono dane 300 unikalnych pacjentów z rakiem nerki, którzy przeszli częściową lub radykalną nefrektomię. 210 przypadków zostało przygotowane do szkolenia i walidacji modelu, a pozostałe 90 do obiektywnej oceny modelu.

Ze względu na ograniczone możliwości pamięci urządzeń członków zespołu projekt realizowano na platformie Google Colaboratory.

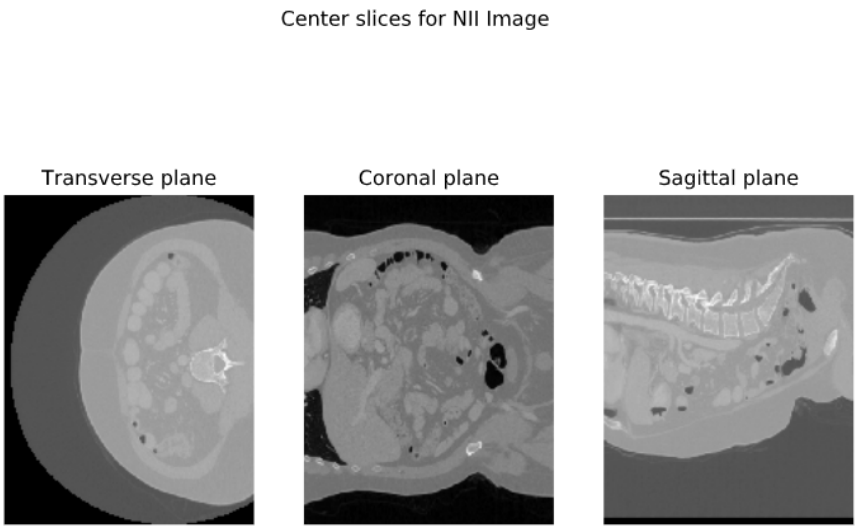
2. Metody

2.1 Wczytanie danych

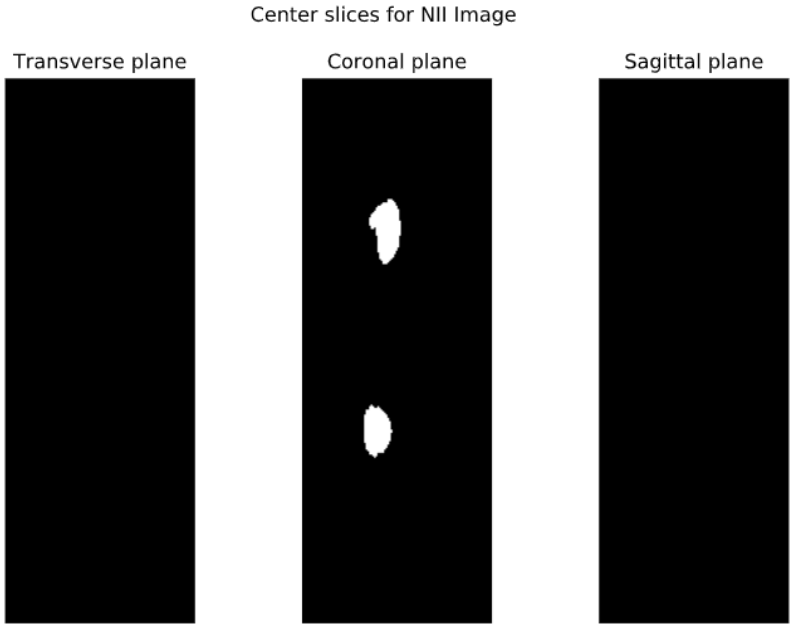
Początkowo sklonowano dane pochodzące z githuba wyzwania do Colaba (notatnik *data_load.ipynb*). Używając dostarczonej z danymi funkcji *get_imaging()*, dane zostały pobrane do odpowiedniego folderu. Następnie utworzono foldery train oraz test, w których utworzono foldery imaging i segmentation. Tak pogrupowane dane umożliwiły w dalszym etapie podział na dane treningowe oraz testowe. Dane pobrane z repozytorium wyzwania uporządkowane były w 300 folderach, a w każdym z nich znajdowały się pliki imaging.nii.gz oraz segmentation.nii.gz. Przy użyciu funkcji *list_files()*, która korzysta z biblioteki os przeniesiono te pliki do odpowiednich folderów. W funkcji można ustawiać jak ilościowo mają wyglądać zbiory testowe i treningowe. Ponieważ platforma Google Colaboratory “przechodziła” po folderach losowo co w pewnych etapach znacznie utrudniało działania, w funkcji dopisano fragment kodu, który “zmusza” funkcję do przechodzenia po przypadkach po kolei. W kolejnym kroku przy użyciu funkcji *kits19_utils.py*, która opisana jest poniżej zapisano przekroje dla każdego przypadku. Foldery zarchiwizowano do formatu zip i przekopiowano na wcześniej utworzony dysk Google, aby nie trzeba było każdorazowo wczytywać danych. Proces ten jest czasochłonny, więc korzystanie z danych zapisanych na dysku znacznie przyspieszyło kolejne etapy pracy.

2.2 Podział na przekroje

Wszystkie funkcje potrzebne do wyświetlania i podziału na przekroje znajdują sie w pliku *kits19_utils.py*. Do wczytania skanu z tomografu wykorzystano bibliotekę nibabel. Przygotowana funkcja *load_image()* przyjmuje jako argument plik z rozszerzeniem *".nii.gz"* i zwraca go w formie macierzy numpy, co umożliwia dalsze operacje wykonywane w kolejnych etapach. Funkcja *generate_axis_view* pozwala zobrazować środkowe przekroje skanów. Poniżej wynik jej działania dla case_123:



Rysunek 1. Środkowe przekroje dla imaging w case_123



Rysunek 2. Środkowe przekroje dla segmentation w case_123

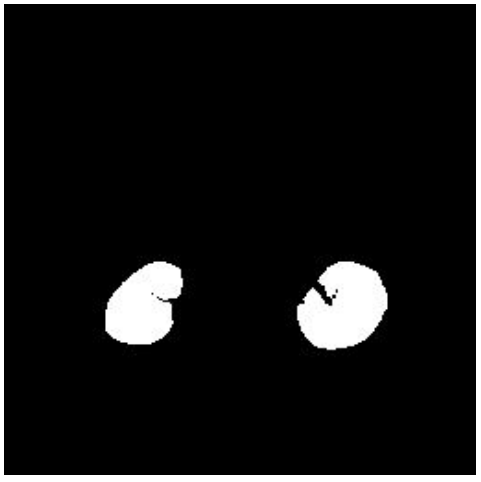
W przedstawionym przekroju segmentacja jest najlepiej widoczna dla płaszczyzny czołowej w środkowych przekrojach, natomiast biorąc pod uwagę widoczność segmentacji we wszystkich przekrojach, najlepiej jest ona widoczna w płaszczyźnie poprzecznej.

W takiej też płaszczyźnie zostały zapisane poszczególne prz`ekroje, posłużyła do tego funkcja *save_slices_all,* która przed zapisaniem przekształcała skan na skalę Hounsfielda. Jest to typowa skala używana w tomografii komputerowej, jednakże obrazy generowane przez tomograf nie są zapisywane przy jej pomocy, dlatego niezbędne jest ręczne przekształcenie. Wyniki zostały zapisane w formacie *".jpeg"*.

2.3 Preprocessing danych

W celu przygotowania danych do wstawienia do sieci przygotowano funkcje *imaging_preprocessing()* i *segmentation_preprocessing()*. Korzystając z przekrojów stworzono generatory, dzięki którym ograniczono pamięć potrzebną do trenowania sieci. Oprócz tego funkcje realizowały również zmianę rozmiarów do 256x256x1 oraz zmianę ze skali RGB na skalę szarości. W celu przeskalowania wartości w funkcji *imaging_preprocessing()* obrazy podzielono przez 255, dzięki czemu uzyskano wartości w zakresie 0-1. Dla funkcji *segmentation_preprocessing()* zastosowano dzielenie przez 127, dzięki czemu uzyskano wartość 0 dla tła, 1 dla nerki oraz 2 dla wysegmentowanego nowotworu.. Dzięki tym przekształceniom udało się skrócić czas trenowania sieci i ograniczyć pamięć potrzebną na wykonanie tej operacji. Użyto standardowej wartości *batch_size = 32*. Poniżej obrazy (dla różnych case'ów) po zastosowaniu funkcji do preprocessingu:

Rysunek 3. Obraz po zastosowaniu funkcji *imaging_preprocessing()*

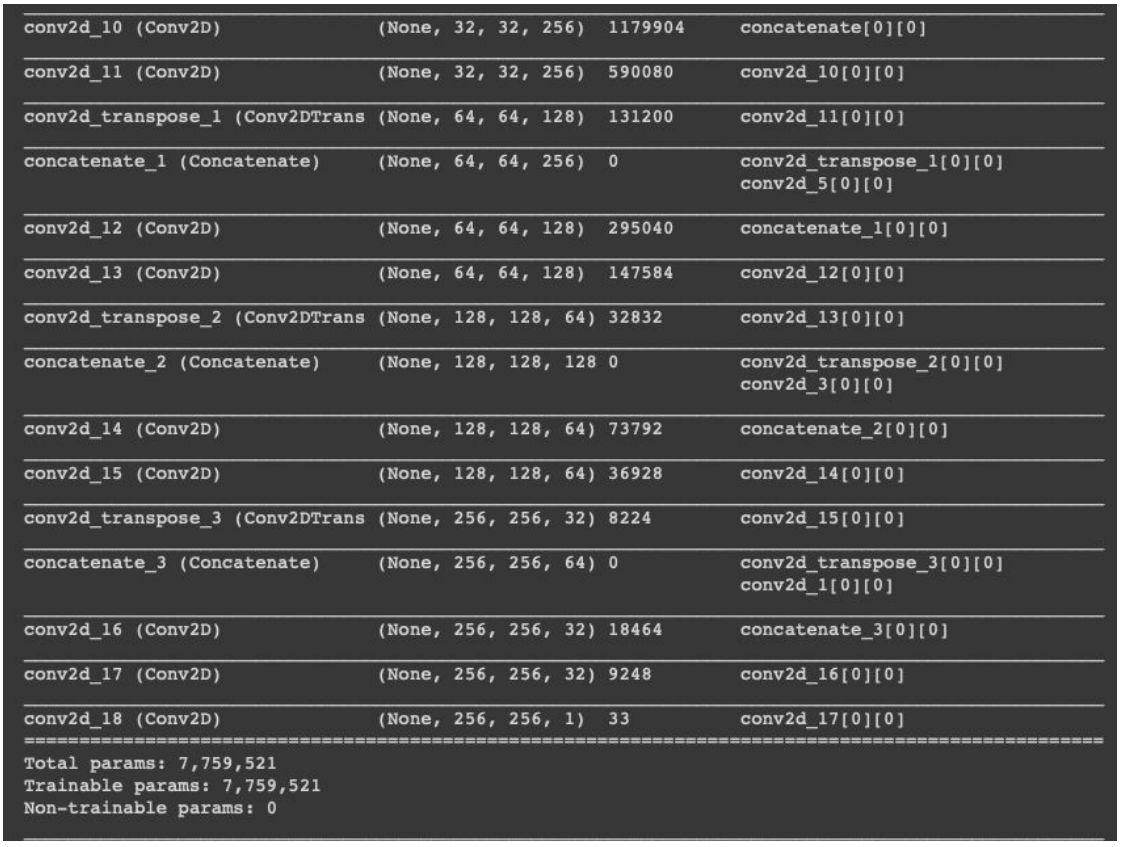


Rysunek 4. Obraz po zastosowaniu funkcji *segmentation_preprocessing()*

2.4 Sieć U-Net

Ze względu na szerokie wykorzystywanie sieci U-Net w realizacji segmentacji biomedycznych, znalazła ona zastosowanie w projekcie. Dzieli się ona na dwie ścieżki: enkoder i dekoder. Enkoder zbudowany został z czterech bloków składających się z dwóch identycznych warstw *Conv2D* i jednej warstwy *Maxpooling2D,* po których następują dwie warstwy *Conv2D*. Warstwy w poszczególnych blokach różnią się między sobą ilością filtrów. Następne warstwy to dekoder, który ponownie zbudowany jest z czterech bloków składających się z takich samych warstw *Conv2D*. Dodatkowo każdy z bloków jest rozpoczynany przez warstwę transponowaną. Architektura sieci została przedstawiona poniżej:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 1)	0	
conv2d (Conv2D)	(None, 256, 256, 32)	320	input_1[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 32)	9248	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 64)	18496	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36928	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 128)	147584	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 256)	590080	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 256)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 512)	1180160	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 512)	2359808	conv2d_8[0][0]
conv2d_transpose (Conv2DTranspo	(None, 32, 32, 256)	524544	conv2d_9[0][0]
concatenate (Concatenate)	(None, 32, 32, 512)	0	conv2d_transpose[0][0] conv2d_7[0][0]



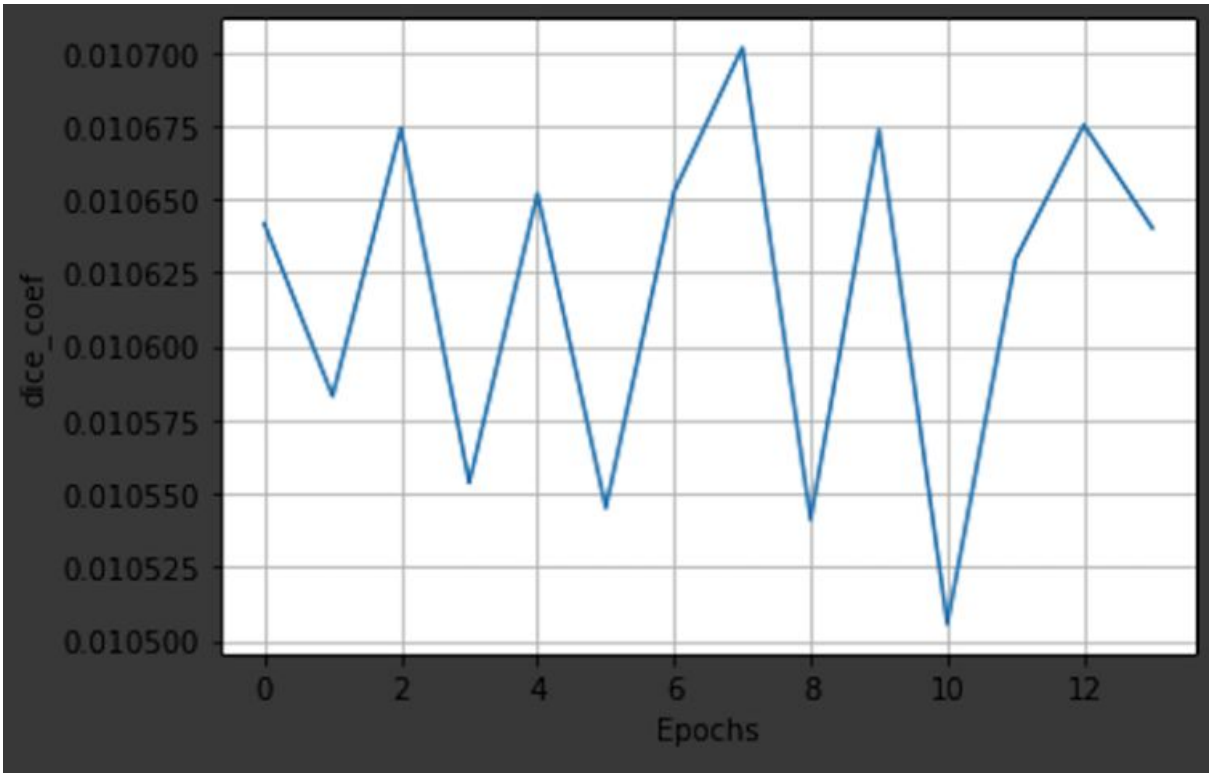
Rysunek 5. Architektura sieci U-Net

Ze względu na problemy z Google Colaboratory, nie było możliwości wytrenowania sieci na wszystkich dostępnych case'ach, ponieważ połączenie było przerywane po upływie określonego czasu. Z tego powodu zdecydowano na wytrenowanie modelu na zbiorze treningowym składającym się z 35 case'ów oraz walidacyjnym składającym się z 15 case'ów. Ograniczono również liczbę epok do 15.

Użyto optymalizatora *Adam* oraz zastosowano funkcję straty *dice_coef_loss()*. Ze względu na uzyskane wyniki nie podjęto próby optymalizacji modelu przy pomocy dopasowywania parametrów optymalizatora.

3. Wyniki

Zależność współczynnika *dice_coef* od aktualnej epoki została przedstawiona na wykresie poniżej:



Rysunek 6. Wykres zależności *dice_coef* od epoki

Analizując wykres można stwierdzić, że wyniki w żaden sposób nie są satysfakcjonujące. *Dice_coef* na poziomie ok. 1% nie jest wystarczający, aby przeprowadzić dalszą analizę i liczyć na jej pozytywne skutki. Przyczyną takiego stanu rzeczy prawdopodobnie jest błąd popełniony na którymś z poprzednich etapów pracy. W celu jego wykrycia i rozwiązania należałoby jeszcze raz prześledzić wszystkie kroki realizacji projektu, w szczególności uwzględniając zmianę formatu plików na *.png* oraz *preprocessing*. Przyczyną może być również złe dopasowanie architektury sieci do danych.

4. Dyskusja i podsumowanie

Dalsze kroki, które należałoby podjąć w przypadku uzyskania bardziej wiarygodnych wyników współczynnika *dice_coef*:

1. Wizualizacja wyników oddzielnie dla nerki i dla nowotworu.
2. Obliczenie współczynnika dla obu przypadków wraz z maksymalnymi i minimalnymi jego wartościami oraz odchyleniem standardowym.
3. Optymalizacja wyników poprzez niewielkie poprawki w sieci, optymalizatorze i dopasowywanie funkcji *dice_coef()* do naszych danych.

Napotkane problemy w trakcie realizacji projektu:

Jednym z pierwszych problemów, na które napotkaliśmy podczas realizacji projektu, było użycie złej funkcji podczas zapisywaniu przekrojów. Funkcja *cv2.imwrite()* z biblioteki *opencv* zapisywała obrazy przekształcone do wartości 0-1, co powodowało utratę danych w zapisanych plikach (zapisane przekroje były w całości czarne). Problem został rozwiązany poprzez użycie funkcji *plt.imsave()*. Problem stanowił również rozmiar danych, który uniemożliwiał przechowywanie ich na własnych urządzeniach. Zamiast każdorazowego wczytywania plików na nowo do Google Colaboratory, zdecydowano na zapisanie ich w archiwum i przechowywanie na dysku. Dzięki temu każde następne wczytanie trwało kilka sekund. Niestety rozwiązanie, jakim jest Google Colaboratory nie do końca również się sprawdziło - mimo ogromnego ułatwienia w kwestii przechowywania dużych rozmiarów danych, problem rozpoczął się w momencie trenowania sieci. Niejednokrotnie *runtime* był automatycznie restartowany, co uniemożliwiło działanie na dużej ilości epok.

Nakład pracy członków zespołu:

Początkowo zrobiono wstępny podział pracy między członków zespołu:

- Piotr Sumara - wczytywanie danych i przygotowanie ich do podziału na przekroje
- Krzysztof Kwaśniak - podział na przekroje i zamiana na skalę Hounsfielda
- Małgorzata Sosin - preprocessing
- sieć i ewaluacja - działanie wspólne

W praktyce jednak ciężko jest rozdzielić nakład pracy między członków zespołu, ponieważ na każdym etapie realizacji projektu praca była wspólna. Współtwórcy komunikowali się ze sobą przy każdym zadaniu i wspólnie ustalali strategię działania tak, aby każdy w pełni rozumiał przedstawione rozwiązania. W przypadku napotkanych problemów przy implementacji, wspólnie szukali ich rozwiązania, dlatego można uznać, że praca była rozdzielona po równi.

5. Bibliografia

https://www.kaggle.com/gzuidhof/full-preprocessing-tutorial?fbclid=IwAR3ihE_JBMHKkX3UL
\
https://www.freecodecamp.org/news/how-to-transfer-large-files-to-google-colab-and-remote-jupyter-notebooks-26ca252892fa/
https://nipy.org/nibabel/

https://arxiv.org/ftp/arxiv/papers/1908/1908.02625.pdf

https://arxiv.org/ftp/arxiv/papers/1811/1811.04815.pdf