PROJEKT Z PRZEDMIOTU

Techniki Obrazowania Medycznego

Inżynieria Biomedyczna 2019/2020

Michał Komala
Jakub Kowalski
Sophia Krupnik

Cel projektu

Celem projektu jest zapoznanie się z problemem analizy obrazów medycznych dotyczącej zautomatyzowanej segmentacji obrazów CT. Na podstawie "Challenge" z konferencji naukowej MICCAI 2019 i załączonych danych, podjęto się próby doboru i implementacji funkcji, która umożliwi automatyczną segmentację zdjęć CT nerek i nowotworów nerek oraz oceny uzyskanych wyników. Zdecydowano się na wykorzystanie techniki splotowych sieci neuronowych (convolutional neural networks) U-Net obrazów 2D.

Wstęp

Co roku wiele osób zapada na nowotwory nerek. Leczenie polega na operacyjnym usunięciu zmiany rakowej, a przy tym ważna jest znajomość jej dokładnego położenia. Kwalifikacja do zabiegu nefrektomii odbywa się na podstawie zdjęć tomografii komputerowej lub rezonansu magnetycznego. Badanie wykrywające umiejscowienie i rozległość zmiany nowotworowej jest pracochłonne i wymaga ręcznych pomiarów radiograficznych. Z tego względu coraz więcej badaczy pracuje nad rozwojem zautomatyzowanego semantycznego systemu segmentacji, który znacznie ułatwi proces diagnostyczny i późniejsze leczenie. W odpowiedzi na wyzwanie powstał program “KiTS19 Challenge”. Zawiera zbiór 300 obrazów CT, kontekst kliniczny oraz wyniki leczenia każdego z pacjentów, u których wykonano nefrektomię częściową lub radykalną [1], [2].

W pracach badawczych często do automatycznej segmentacji wykorzystywane są konwolucyjne sieci neuronowe o architekturze U-Net. Jest to model warstwowy, w którym występuje ścieżka kontraktacji i ekspansji, co nadaje mu kształt litery U. Podczas kontrakcji obraz jest zmniejszany co pozwala na rozpoznanie obiektu, a podczas ekspansji obraz wraca do swojego pierwotnego rozmiaru i umożliwia odzyskanie lokalizacji tego obiektu. W ten sposób, analizując obraz piksel po pikselu, sieć jest w stanie przewidywać, gdzie znajduje się poszukiwany obiekt [3].

W poniższej pracy podjęliśmy próbę utworzenia dwóch sieci neuronowych U-Net: pierwsza sieć - do wykrywania nerek na obrazie CT, druga - do wykrywania nowotworów nerek. Czytelnik zostanie zapoznany tokiem myślenia autorów podczas tworzenia sieci, a także z kolejno podejmowanymi krokami, na które składają się: przygotowanie środowiska Colab, przygotowanie danych do nauki sieci, implementacja architektury sieci, także nauka

obu sieci U-Net. Na sam koniec przetestujemy wyuczoną sieć oraz przedstawimy otrzymane wyniki.

Metody i materiały

1. Przygotowanie środowiska Colab

   Z racji tego, że obliczenia podczas nauki sieci mogą być czasochłonne dla komputera, który posiada niewielką pamięć RAM, postanowiliśmy wykorzystać środowisko Google Colab. Początkowo planowaliśmy wykorzystać procesor TPU, nie potrafiliśmy jednak w pełni wykorzystać jego potencjału. Dlatego zdecydowaliśmy się na użycie zwykłego CPU gwarantującego niecałe 13 GB pamięci RAM. W środowisku importowaliśmy podstawowe biblioteki używane przy tworzeniu sieci takie jak: tensorflow, numpy, os czy matplotlib.

2. Pobranie i zapoznanie się z danymi

   Początkowo sklonowaliśmy dane udostępnione w githubie challange'a (*https://github.com/neheller/kits19.git*) do stworzonego środowiska. Następnie zapoznaliśmy się z funkcjami dostarczonymi wraz z danymi, które ułatwiały ich używanie i odczyt. Pobrany zestaw danych zawierał 300 zbiorów obrazów. W pierwszych 210 zestawach zawarte było od kilkudziesięciu do kilkuset obrazów przekrojów oraz odpowiadająca każdemu zdjęciu maska z oznaczeniem nerki oraz nowotworu. W masce wartość 0 jest przypisana dla tła, 1 dla nerki, a 2 dla nowotworu. Do załadowania obrazów przekrojów i masek użyto dołączonej w repozytorium funkcji *load_case*. Zapoznaliśmy się z rozmiarami danych, a także wartościami pikseli poszczególnych obrazów w celu opracowania metody przygotowania danych do nauki.

3. Przygotowanie danych

   Początkowo planowaliśmy odczytywać po kolei każdy obraz z dostępnych zbiorów w postaci macierzy, a następnie dodawać tę macierz do większej macierzy, w której przechowywane byłyby wszystkie wartości obrazów do nauki sieci. Podobnie postępowano by z maskami nerki oraz nowotworu. Następnie obie otrzymane macierze posłużyłyby do nauki jednej sieci. W rezultacie otrzymywaliśmy macierze o kolosalnych rozmiarach i nie spodziewaliśmy, że będą zajmować one aż tyle pamięci podręcznej - po załadowaniu danych 3 pacjentów, RAM sesji był całkowicie zapełniony (zajęły go referencje do każdej z komórki macierzy). Próbowaliśmy również zapisywać dane na dysku w postaci macierzy npy. Niestety pliki npy zajmowały również mnóstwo pamięci w chmurze - dane dwóch pacjentów zajęły ponad 20 z dostępnych 40 GB. Z tego powodu postanowiliśmy zmienić podejście i zapisywać każdy z przekrojów oraz każdą z masek jako obraz png, prowadząc je do odpowiednio utworzonych folderów. Oprócz tego zdecydowaliśmy się na utworzenie dwóch oddzielnych masek dla nowotworów i nerek oraz naukę dwóch sieci.

Pomijając fakt, że zapis przekrojów do formatu png nie jest najlepszą praktyką w przypadku obrazów medycznych, sposób ten pozwolił nam na przechowywanie ponad 120 zbiorów przy niewielkim zużyciu pamięci dostępnej w chmurze. Oprócz tego zapis do formatu png dawał dodatkowe udogodnienia, które zostaną przedstawione w późniejszej części.

Stworzono trzy zestawy danych, które posłużyły do trenowania (100 zbiorów), walidacji (10 zbiorów) oraz testowania (10 zbiorów) sieci neuronowej. W każdym zestawie wyróżniono odpowiednio po trzy zbiory: maski nowotworów oraz nerek, a także pełne obrazy CT. Obrazy były odczytywane przy użyciu funkcji get_fdata(), a następnie kierowane do odpowiednich nowo utworzonych folderów.

Kolejnym krokiem było odpowiednie przygotowanie danych. W tym celu użyto funkcji ImageDataGenerator z biblioteki keras, która jest jedną z wcześniej wspomnianych zalet konwersji obrazów do formatu png. Generatory pozwalają na podłączenie do właściwej ścieżki, gdzie przechowywane są obrazy i generowanie pożądanej liczby obrazów podczas nauki sieci. Dodatkowo dają możliwość poddania obróbce wczytanego obrazu zanim finalnie trafi on do sieci. W naszym przypadku obrazy przeskalowano do wymiarów 256x256, poddano normalizacji do wartości [0, 1] oraz przedstawiono w skali szarości. Zmniejszejszenie rozmiarów obrazu zmniejszyło obciążenie pamięci RAM, a normalizacja zwiększyła czułość sieci (przy uczeniu na obrazach zalecane jest stosowanie wartości w podanym przez nas przedziale). Ciekawą możliwością oferowaną przez generatory jest również losowe wczytywanie obrazów, co również jest bardzo istotną właściwością przy naszej sieci.

4. Przygotowanie architektury sieci

Architekturę sieci U-Net stworzono z pomocą z pomocą biblioteki keras. Przy pomocy podrzędnej jej biblioteki layers zdefiniowaliśmy kolejne warstwy sieci neuronowej. Na jej strukturę składały się:

- wejście sieci

Przedstawienie wymiarów obrazu wejściowego przy pomocy funkcji tf.keras.layers.Input. W naszym przypadku: rozmiar 256x256 w skali ‘grayscale’: (256,256,1).

- warstwa skurczu

Tworzyliśmy kolejne piętra warstwy skurczowej przy użyciu funkcji:

- tf.keras.layers.Conv2D - zdefiniowanie liczby filtrów wykonujących splot przestrzenny na obrazach otrzymanych na wejściu. Zdecydowaliśmy się na wykonywanie splotów z użyciem maski o wielkości 3x3, która początkowo przypisywane ma wartości rozkładu normalnego.
- tf.keras.layers.Dropout - jest to funkcja pozwalająca na zwiększenie czułości sieci i zmniejszenie prawdopodobieństwa wystąpienia overfittingu poprzez zwiększenie “niezależności” neuronów. Ustawiliśmy jej wartość na 0,1 co oznacza, że przy nauce co krok odrzucane jest losowo 10% neuronów i sieć uczy się bez ich udziału.

- tf.keras.layers.MaxPooling2D - wybór maski o rozmiarze 2x2 spowodował dwukrotne zmniejszenie obrazu przy końcu każdego z poziomów skurczu.

- warstwa rozkurczu

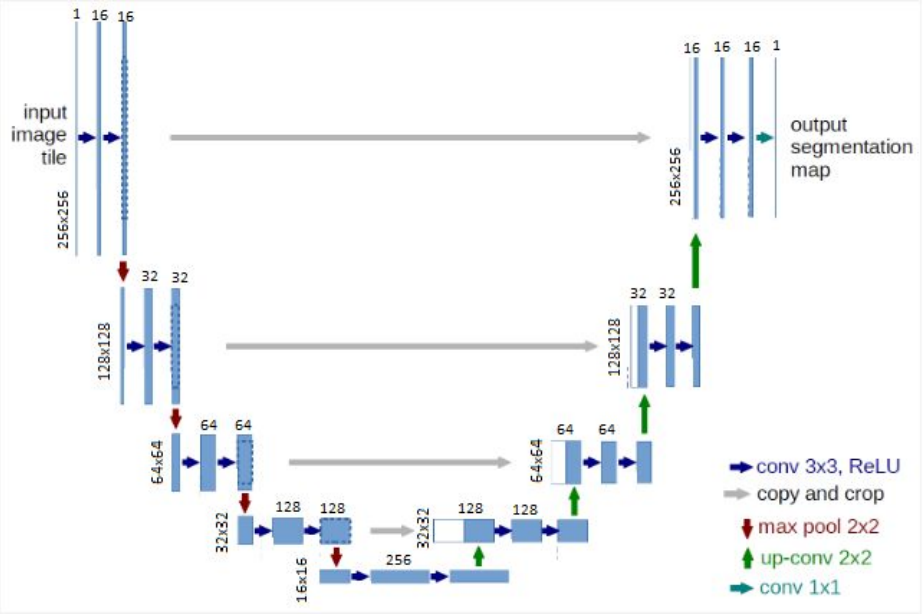
Tworzyliśmy kolejne piętra warstwy rozkurczowej przy użyciu funkcji:

- tf.keras.layers.Conv2DTranspose - transformacja w kierunku przeciwnym niż normalny splot. Poprzez dobór rozmiaru maski i wielkości kroku do tego samego co w funkcji MaxPooling2D następuje dwukrotne powiększenie macierzy.
- tf.keras.layers.concatenate - połączenie odpowiadających sobie poziomów warstwy skurczowej i rozkurczowej.
- tf.keras.layers.Conv2D oraz tf.keras.layers.Dropout

- wyjście sieci

Zdefiniowanie pojedynczego filtra z użyciem funkcji tf.keras.layers.Conv2D.

Dużym problemem okazał się dobór ilości filtrów na każdym z poziomów projektowanej sieci. Początkowo ich ilość była zbyt duża znacznie spowalniając naukę. Z racji tego, że pojedyncza sesja Colaba resetuje się po okresie około 12 godzin zmuszeni byliśmy zmniejszyć liczby filtrów tak, aby zmieścić się w tym czasie. Początkowa ilość parametrów do wyuczenia wynosiła ponad 34 mln. Czterokrotnie zmniejszono liczbę filtrów na każdym poziomie przy każdym splocie i ostateczna liczbą parametrów do wyuczenia przez sieć wyniosła 2,158,417. Poniżej przedstawiona jest graficznie architektura zdefiniowanej sieci demonstrująca ilość użytych poziomów oraz filtrów.



Rysunek 1. Architektura zdefiniowanej sieci.

Załadowaliśmy finalną wersję modelu z użyciem funkcji compile. Do optymalizacji użyliśmy polecanego i wielokrotnie powtarzającego się w publikacjach

naukowych ‘adam optimizer’. Ze względu na to, że nasza sieć będzie przewidywać przynależność jedynie do jednej klasy, jako funkcję straty obraliśmy ‘binary cross entropy’. Dzięki temu uzyskamy prawdopodobieństwo dla każdego z pikseli w obrazie do przynależności do uczonej klasy. Przedstawioną architekturę użyto zarówno dla sieci poszukującej nerek jak i nowotworów.

5. Uczenie sieci

Po przygotowaniu generatorów obrazów i architektury sieci mogliśmy przystąpić do nauki. Cały proces objaśniamy na przykładzie sieci do wykrywania nerki z racji tego, że w przypadku sieci do wykrywania nowotworów postąpiono w analogiczny sposób. Ponieważ do uczenia wykorzystywane są generatory, skorzystaliśmy z funkcji *fit_generator*. Do nauki użyto cztery generatory: 2 do obrazów CT oraz 2 do masek nerki (po jednym generatorze z każdego rodzaju do walidacji oraz po jednym do treningu). Eksperymentalnie wyznaczyliśmy ilość obrazów, która może być jednocześnie podawana do sieci (batch_size) bez przekroczenia dostępnego RAMu. Na zasadzie prób i błędów wyznaczono batch_size = 128, co znacznie przyspieszyło naukę sieci. Ze względu na to, że obrazów treningowych było ponad 22 tysiące, podczas treningu sieć musiała wykonać prawie 170 kroków przy każdym powtórzeniu (epochu). Z racji wcześniej wspomnianego mankamentu związanego z resetowaniem Colaba, w czasie 12 godzin byliśmy w stanie wykonać jedynie 2 powtórzenia. Z tego powodu naukę sieci przeprowadzono w dwóch częściach. Colaba, w którym prowadzono naukę połączono z dyskiem Google. Początkowo wykonano dwa pierwsze powtórzenia, a otrzymane po nich wagi sieci zapisano w formacie hdf5 na dysku. Po resecie sesji załadowano wcześniejsze wagi z użyciem funkcji *load_weights* i wznowiono naukę od 2 epocha (parametr initial_epoch), po czym dokonano kolejnych dwóch powtórzeń. Przy nauce dodatkowo zdefiniowano zabezpieczenie przed overfittingiem w postaci funkcji EarlyStopping. Gdyby funkcja straty ulegała wzrostowi nauka zostałaby przerwana.

Prezentacja wyników

Zbiór testowy zawierał obrazy przekrojów dla 10 przypadków (2271 obrazów). Załadowaliśmy modele, a następnie przy użyciu funkcji *predict_generator()* dla każdego piksela z testowych przekrojów wyznaczyliśmy prawdopodobieństwo przynależności do klas: ‘nerka’ albo ‘nowotwór’. Uznaliśmy, że prawdopodobieństwo powyżej 50% będzie dobrą granicą, dla której piksel traktowany będzie nie jako tło, a jako nerka bądź jej nowotwór (w zależności od testowanej sieci). W ten sposób zamieniliśmy otrzymane macierze prawdopodobieństw w maski binarne obrazujące wykrytą nerkę lub nowotwór (wartość 1) oraz tło (wartość 0). W celu odpowiedniego przedstawienia wyników dokonano wizualizacji, która pokazuje porównanie maski z predykcji, maski przygotowanej ręcznie oraz odpowiadającego przekroju.

Rysunek 2. Przykład złego dopasowania maski nerki.

Rysunek 3. Przykład złego dopasowania maski nowotworu.

Rysunek 4. Przykład średniego dopasowania maski nerki.

Rysunek 5. Przykład średniego dopasowania maski nowotworu.

Rysunek 6. Przykład dobrze dopasowanej maski nerki.

Rysunek 7. Przykład dobrze dopasowanej maski nowotworu.

Pokazane przykłady wyraźnie wskazują na niedokładność naszego rozwiązania. Bardzo często nie występuje dopasowanie żadnej maski nerki, dopasowanie tylko do jednej nerki lub segmentowa jest inna struktura (np. rdzeń kręgowy). Inną częstą sytuacją jest dopasowywanie maski nerki do przekrojów na których nie da się wyodrębnić nerek. Masek, które moglibyśmy ocenić jako bardzo dobrze dopasowane jest bardzo niewiele.

Kolejnym krokiem była analiza statystyczna polegająca na porównaniu podobieństwa odpowiadających sobie masek otrzymanych przy pomocy wyuczonej sieci oraz tych wyznaczonych ręcznie. Napisano funkcję, która początkowo porównuje zgodność położeń pikseli o wartości 1 (nerka/nowotwór) pomiędzy maską wyznaczoną przez sieć oraz odpowiadającej jej masce ze zbioru testowego. Następnie określono procentowy udział liczby zgodnych pikseli względem liczby pikseli zaznaczonej na masce testowej.

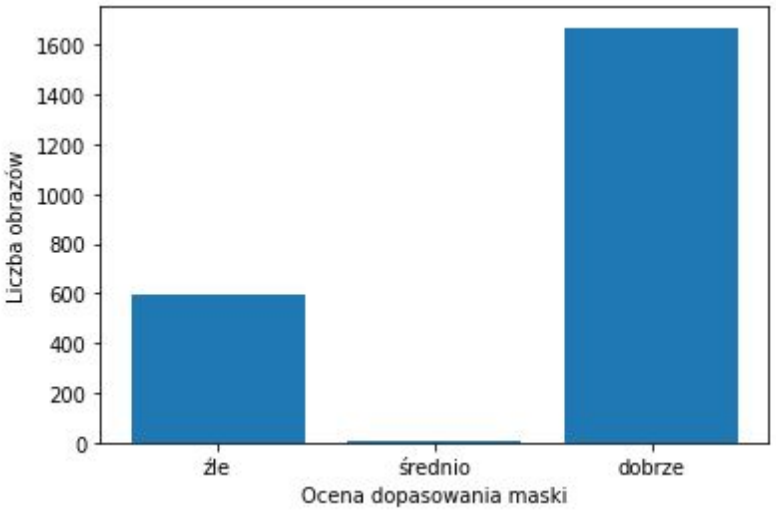
W przypadku masek nowotworu dokonano podziału na trzy zbiory:

a) maska źle dopasowana (<10%) - sieć nie odnalazła istniejącego nowotworu, bądź wskazała nieistniejący nowotwór (oba przypadki mogłyby być tragiczne w skutkach bez interwencji lekarza).

b) maska średnio dopasowana (>10% i <50%) - daje cenną wskazówkę o położeniu nowotworu, jednak wyznacza jedynie jego niewielką powierzchnię

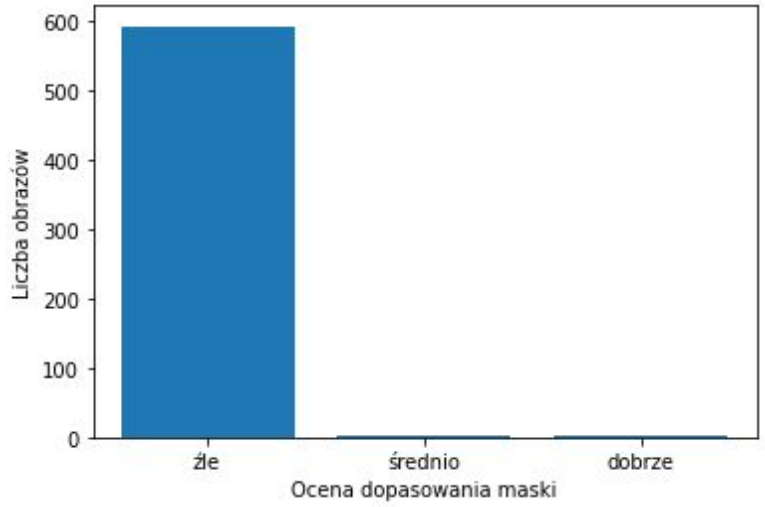
c) maska dobrze dopasowana (>50%) - w większości wykrywa istniejący nowotwór

W rezultacie otrzymano histogram przedstawiony na Wykresie 1. Wśród masek znajdowało się jednak wiele takich, na których nie były oznaczone nowotwory (macierz wypełniona samymi 0), co w pewien sposób zaburza wyniki. Histogram potwierdza jednak, że w wielu przypadkach, gdzie na obrazie nie znajdował się nowotwór, sieć prawidłowo nie oznaczyła żadnego fragmentu obrazu jako potencjalnego guza.



Wykres 1. Zależność liczby obrazów od obiektywnej oceny dopasowania maski nowotworu

Postanowiliśmy, więc w dalszym postępowaniu wykluczyć sytuacje, w których na obrazie nie znajdował się guz i sieć nie oznaczyła go. Ponownie stworzyliśmy histogram (Wykres 2), który obrazuje precyzje sieci w przypadku przekrojów, które zawierały nowotwór.



Wykres 2. Zależność liczby obrazów od obiektywnej oceny dopasowania maski nowotworu z wykluczeniem przekrojów niezawierających guza

Okazało się, że zaledwie niewielki odsetek z ‘dobrego’ dopasowania z Wykresu 1 stanowią prawidłowo wyznaczone maski. W większości źle dopasowane maski stanowiły niesłusznie wskazane nowotwory (wyznaczenie nowotworu pomimo jego braku) bądź brak jakiegokolwiek zaznaczenia w przypadku ich występowania.

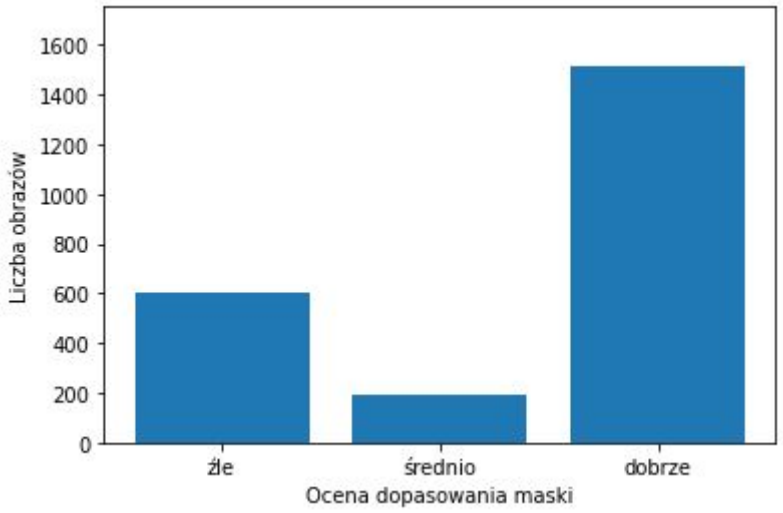
Dla dopasowanych masek nerek stworzono także 3 zbiory, jednak z innymi granicznymi wartościami procentowymi:

a) maska źle dopasowana (<30%) - sieć nie odnalazła nerki, bądź wskazała zaledwie jej niewielki fragment

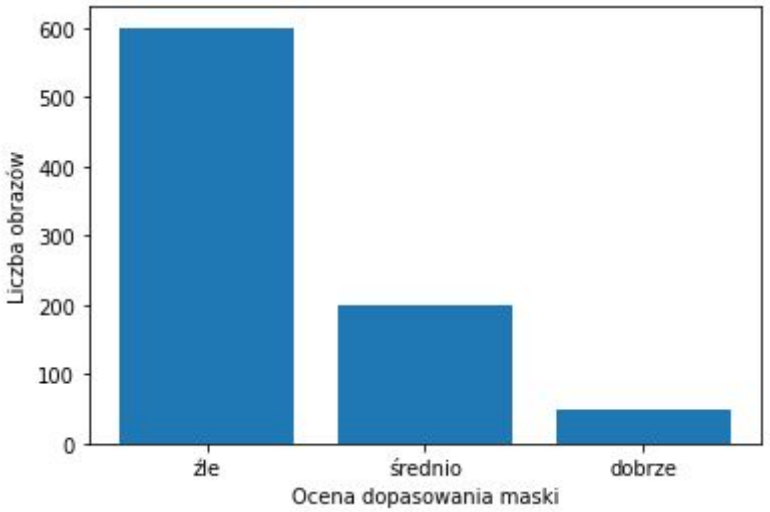
b) maska średnio dopasowana (>30% i <70%)

c) maska dobrze dopasowana (>70%) - wskazanie sporego fragmentu nerki/nerek

Wykonano dwa histogramy na podobnej zasadzie jak w przypadku nowotworów, które zostały przedstawione poniżej (Wykres 3, Wykres 4).



Wykres 3. Zależność liczby obrazów od obiektywnej oceny dopasowania maski nerki



Wykres 4. Zależność liczby obrazów od obiektywnej oceny dopasowania maski nerki z wykluczeniem przekrojów niezawierających nerki

Na podstawie otrzymanych histogramów możemy wyciągnąć podobne wnioski jak w przypadku nowotworów. W przypadku nerek możemy zaobserwować jednak więcej masek zakwalifikowanych do zbiorów średniego i dobrego dopasowania. Prawdopodobnie jest to spowodowane występowaniem nerek w tych samych miejscach oraz powtarzającym się kształtom tego narządu, co ułatwiło sieci uczenie.

Wykresy pokazują, że maski źle dopasowane stanowią znaczną większość. Dobre dopasowanie maski jest marginalnym ułamkiem dla wszystkich masek. W przypadku masek nowotworów ta tendencja jest jeszcze większa. Prawdopodobnie jest to wynikiem tego, że nowotwory są dużo bardziej nieregularne pod względem kształtu, a także rozmiaru.

Dyskusja z podsumowaniem

Dzięki pracy na powszechnych danych osób poddanych nefrektomii wraz z podsumowaniem leczenia, mamy dostęp do wyników analiz ich zdjęć [3]. Umożliwia to nauczanie sieci do automatycznego rozpoznawania zmian nowotworowych w nerce oraz porównanie działania sieci z wynikami otrzymanymi po ręcznej segmentacji. Błędy przy uczeniu może generować fakt, że obrysy ręcznie zaznaczone, a później wykorzystane do uczenia, mogą w różny sposób obejmować zmiany, ze względu na wykonanie ich w odstępach czasu lub przez różnych specjalistów [4].

Podczas testowania obrazów, wykrywanie obszaru nerki z pierwszej, zaprojektowanej sieci okazało się niewystarczające, co eliminuje jej realne wykorzystania w medycynie. Bardzo niskie prawdopodobieństwo wykrycia nowotworów całkowicie dyskwalifikuje również drugą sieć od zastosowań diagnostycznych. Gorsze wyniki mogą wynikać z mniejszej ilości zaznaczeń nowotworów w porównaniu do zaznaczeń nerek, jak również z niewyraźnej granicy między nowotworem, a nerką. Kolejną przyczyną jest brak powtarzalności zajmowanego obszaru - zmiany mogą znajdować się niezależnie na lewej lub prawej nerce, w różnej pozycji względem głębokości i przekroju. Ponadto wpływ na ostateczny wynik miały wyżej opisane ograniczenia związane z programem colab, przez co wykorzystano mniej filtrów, a na każdym etapie uczenia tracono dane.

System mógłby zostać ulepszony poprzez zastosowanie większej ilości filtrów czy obrazów do treningu oraz wykorzystanie dodatkowych funkcji przetwarzających dane wejściowe np. poprzez wyczyszczenie zbędnych informacji z obrazów. W celu poprawienia wyników można by również wykorzystać dodatkową funkcję generatorów jaką jest *data augmentation*. Pozwala ona na zwiększenie liczby obrazów do nauki sieci poprzez przybliżanie, oddalanie bądź przesuwanie podstawowego obrazu. Dodatkowo zastosowanie tej funkcji wpływa na odporność sieci na występujący w obrazach szum, a w naszym przypadku mogłoby skutecznie pomóc przy nauce sieci do wykrywania nowotworów, których kształty są często nieregularne. Oprócz tego można by zastosować większy odsetek przy funkcji *dropout* - więcej niż 0.1, co mogłoby zwiększyć czułość sieci. Należałoby również poszukać innych metod skrócenia czasu uczenia w celu wykorzystania środowiska Google Colab. Dodatkową sugestią z naszej strony jest również zmiana podejścia do nauki obu sieci. Otrzymane wyniki uświadomiły nam jak ciężkim zadaniem jest trening sieci pod kątem wykrywania samych nowotworów. Sieć wielokrotnie nie znajdowała ich lub wskazywała powierzchnie innych narządów. Wydaje nam się, że lepszą propozycją przy ewaluacji dwóch sieci byłby podział na naukę z użyciem: 1 - masek z jednoczesnym oznaczeniem nerki oraz nowotworu, 2 - masek z samym nowotworem. Obrazy CT podawane byłyby do pierwszej sieci. Uzyskaną z niej maskę nakładano by na przekrój CT i podawano do drugiej sieci. W rezultacie niemożliwe byłoby wykrycie nowotworu poza nerką oraz oczekiwalibyśmy zwiększonej precyzji ze strony sieci przy wykrywaniu tych struktur. Podejście to wymagałoby bardzo dobrego treningu pierwszej sieci, ale przypuszczamy, że otrzymane wyniki byłyby bardziej obiecujące.

Bibliografia

[1] https://kits19.grand-challenge.org/home/
[2] The KiTS19 Challenge Data: https://arxiv.org/pdf/1904.00445.pdf
[3]https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47
[4] http://www.embio.agh.edu.pl/library/Andrzej_Skalski_PhD.pdf
[5]Kurs Image Segmentation using U-Net na kanale YouTube: https://www.youtube.com/channel/UC34rW-HtPJulxr5wp2Xa04w
[6]Olaf Ronneberger, Philipp Fischer, Thomas Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015
[7]https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
[8]https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca
[9]https://keras.io/api/
[10]https://www.tensorflow.org/api_docs/python/tf
[11]https://towardsdatascience.com/keras-data-generators-and-how-to-use-them-b69129ed779c