AKADEMIA GÓRNICZO-HUTNICZA

WEAIiIB, Katedra Automatyki i Robotyki
Laboratorium Biocybernetyki

Przedmiot:	**Akceleracja Algorytmów Wizyjnych w GPU i OpenCL**			
			pr19aaw01	
Temat projektu:				
	Segmentacja obiektów za pomocą głębokich sieci neuronowych z zastosowaniem GPU			
Wykonali:		Daniel Gacek Robert Radzik Aleksander Orlikowski		
	Automatyka i Robotyka		studia:	magisterskie
Rok akademicki:	2019/2020	Semestr:	II	
Prowadzący:		dr inż. Mirosław Jabłoński		
wersja 0.7 Kraków, 13.01.2019 r.				

1. Wstęp

Cel projektu

Celem projektu opisanego w niniejszym sprawozdaniu, realizowanego w ramach przedmiotu Akceleracja algorytmów wizyjnych w GPU i OpenCL, było zaimplementowanie aplikacji opartej na głębokiej konwolucyjnej sieci neuronowej. Sieć miała wykorzystywać zasoby procesora graficznego (ang. *GPU - Graphics Processing Unit*) w problemie segmentacji obiektów na obrazie. Docelowymi obrazami miały być zdjęcia przekrojów żołędzi pozbawionych drewniejącej miseczki (kupuli). Efektem procesu segmentacji miało być oddzielenie obszarów zawierających wnętrze orzecha - nasiono, od pozostałej części obrazu - tła oraz łupiny. Program opracowany miał zostać w języku Python z wykorzystaniem bibliotek TensorFlow oraz Keras, dedykowanych tworzeniu sieci neuronowych.

Proponowane rozwiązanie

Rozwiązanie rozważane w ramach realizowanego projektu oparte jest na przykładowej implementacji sieci neuronowej do potrzeb segmentacji obrazu, zamieszczonej jako repozytorium Git [1]. Zawiera ono modele kilku rodzajów konwolucyjnych sieci, napisane z wykorzystaniem bibliotek Keras oraz TensorFlow. Implementacja wybrana do potrzeb niniejszego projektu bazuje na idei sieci U-Net, po raz pierwszy opisanej w [2]. Jest to sieć składająca się wyłącznie z warstw konwolucyjnych (ang. *fully convolutional network*). Można ją podzielić na dwie części - enkoder odpowiedzialny za wyekstrahowanie z obrazu wejściowego najistotniejszych jego cech oraz dekoder, którego zadaniem jest odtworzenie na podstawie skompresowanych informacji posegmentowanego obrazu wyjściowego. Zastosowano inną niż w oryginalnej sieci U-Net architekturę (układ warstw). W roli enkodera wykorzystano wstępnie wytrenowaną sieć VGG-16 - wersję sieci VGG z szesnastoma warstwami zawierającymi uczone wagi - zaproponowaną w [3]. Cechuje się ona wysoką skutecznością w problemie klasyfikacji obrazów. Najważniejszymi elementami części dekodującej sieci miały natomiast być warstwy nadpróbkujące (ang. *upsamplers*), które pozwalają na zwiększanie wymiarów macierzy zawierających przetworzony obraz. Zaproponowane rozwiązanie wynika z faktu, że dostępny zbiór uczący jest bardzo nieliczny i zdecydowanie niewystarczający do poprawnego wytrenowania rozbudowanej sieci konwolucyjnej. W związku z tym, postanowiono zastosować jako ekstraktor cech obrazu sieć z wagami uprzednio nauczonymi na obrazach z bazy ImageNet. Dodatkowo, problem niewielkiej liczby obrazów uczących postanowiono zmniejszyć poprzez rozszerzenie zbioru uczącego (ang. *training data augmentation*) za pomocą prostych operacji, głównie przekształceń afinicznych.

Zawartość sprawozdania

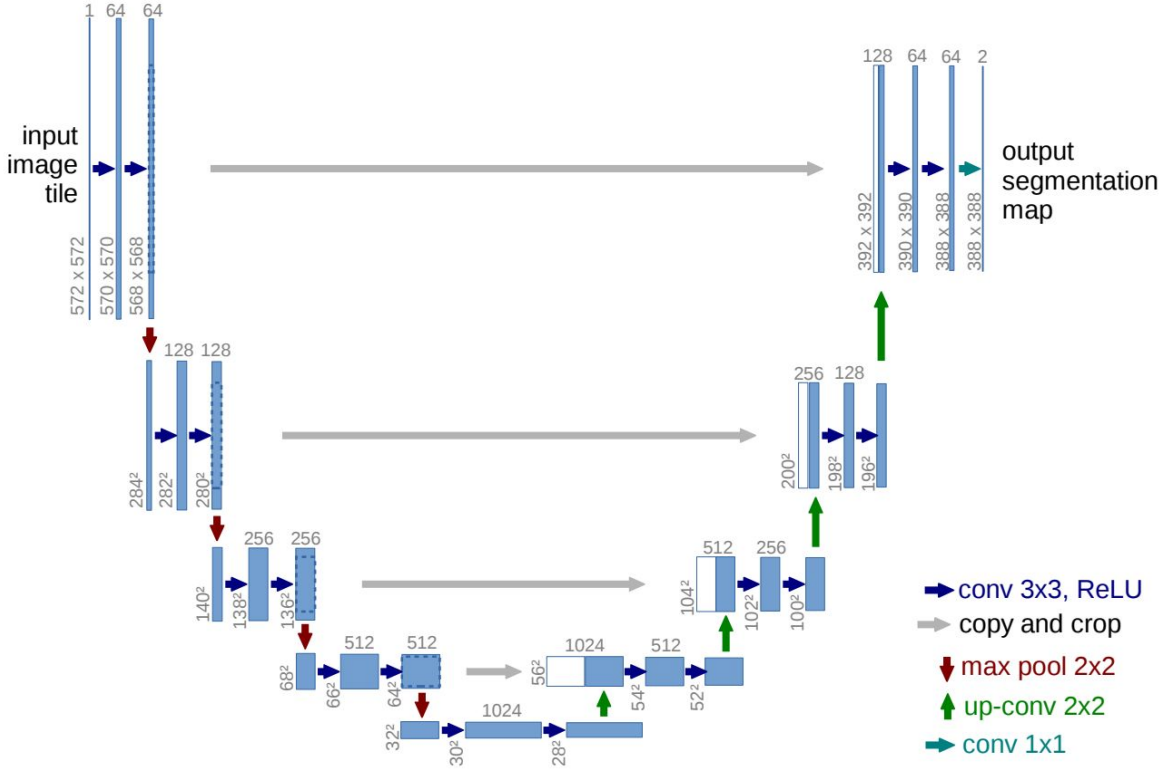
Niniejsze sprawozdanie zostało podzielone na 4 główne rozdziały. Pierwszy z nich zawiera wprowadzenie - przedstawia cele i założenia projektu oraz zarysowuje ideę rozwiązania postawionego problemu. Przedstawiono w nim ponadto dostępne publikacje oraz materiały internetowe związane z badanym zagadnieniem. W rozdziale drugim natomiast szerzej omówiono

rozwiązanie zaproponowane w ramach opisywanego w tym sprawozdaniu projektu. Rozdział trzeci z kolei stanowi opis wstępnych prac związanych z implementacją aplikacji oraz uzyskanych wyników. W czwartym krótko przedstawiono dane związane z zaimplementowanym modelem sieci.

2. Opis proponowanego rozwiązania

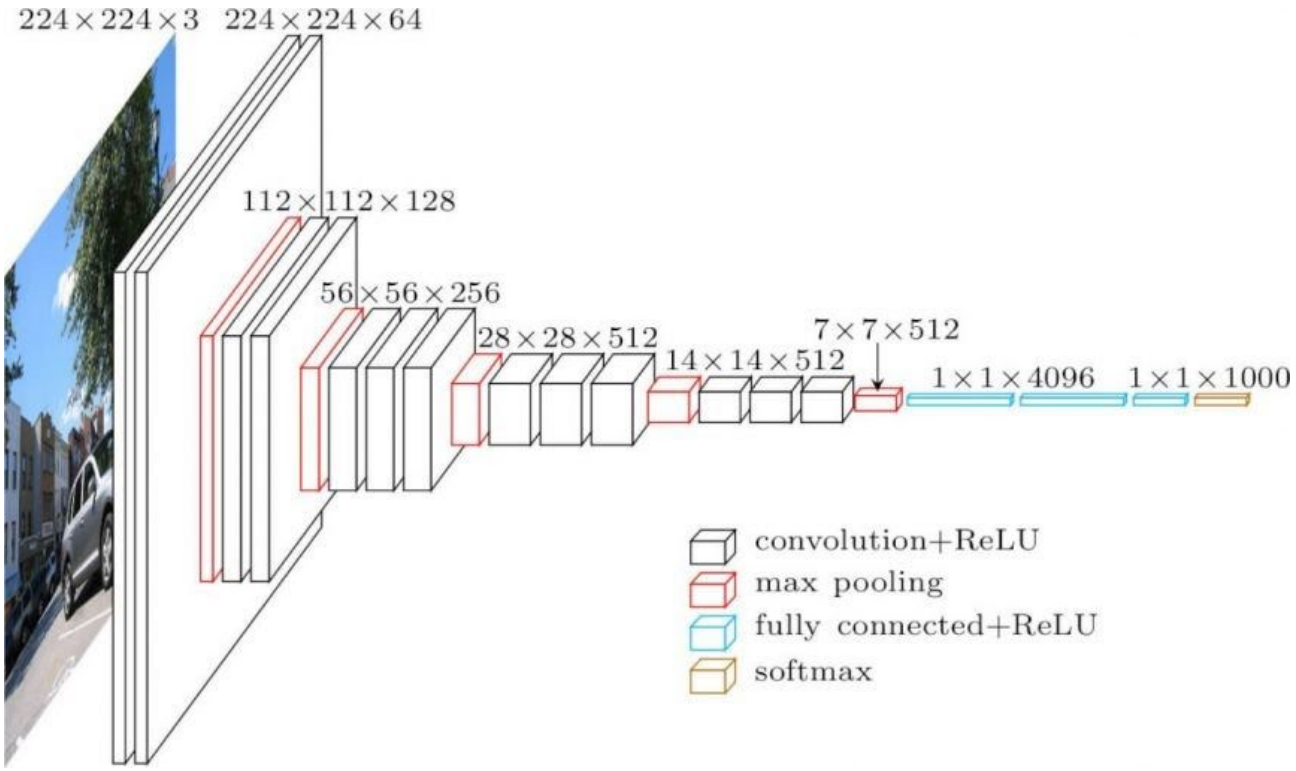
Architektura sieci

Architektura sieci neuronowej wykorzystanej do rozwiązania zadanego problemu segmentacji obrazów żołędzi oparta jest na pomyśle zawartym w oryginalnym opisie sieci U-Net [2]. Stosowanie wyłącznie warstw konwolucyjnych pozwala na zachowanie kontekstu (informacji o otoczeniu danego elementu obrazu) przez cały proces przetwarzania danych wejściowych. Enkoder wraz ze wzrostem liczby kanałów przy jednoczesnym zmniejszaniu wymiaru macierzy umożliwia wydobycie z obrazu wejściowego najistotniejszych jego cech, niezbędnych podczas konstruowania przez dekoder obrazu posegmentowanego. Na rysunku 1. przedstawiono schemat architektury oryginalnej sieci U-Net. Składa się ona z warstw konwolucyjnych, służących przetwarzaniu danych w celu uzyskania na ich podstawie informacji o obrazie, warstw *max-pool* umożliwiających zmniejszenie wymiaru problemu i zachowanie jedynie najważniejszych cech oraz, po stronie dekodera, warstw nadpróbkujących, pozwalających na odtworzenie obrazu w wyższej rozdzielczości. Bardzo istotną cechą sieci U-Net są bezpośrednie połączenia pomiędzy odpowiednimi warstwami enkodera i dekodera. Zdaniem autorów, znacznie poprawiają jakość uzyskiwanych wyników segmentacji. Pozwalają bowiem na przekazanie do warstw dekodera lokalnych cech obrazu w wysokiej rozdzielczości, które w innym przypadku mogłyby zostać utracone. Dzięki nim sieć jest w stanie bardziej precyzyjnie odwzorować obraz wejściowy.



Rys. 1. Architektura oryginalnej sieci U-Net. Źródło: [2]

Wybrano implementację idei sieci U-Net, która jako enkoder wykorzystuje sieć VGG-16. Składa się ona z bloków warstw konwolucyjnych o rozmiarze filtra 3x3, przedzielonych warstwami *max-pool*. Na końcu sieci umieszczono klasyfikator, złożony z trzech warstw w pełni połączonych (ang. *fully connected*) oraz funkcji aktywacji *softmax*. W sumie daje to szesnaście warstw, które podlegają procesowi uczenia (trzynaście konwolucyjnych oraz trzy *fully connected*). Na rysunku 2. przedstawiono architekturę tej sieci. Ponieważ rozważany w ramach niniejszego projektu problem nie dotyczy klasyfikacji, wykorzystano w nim sieć VGG-16 pozbawioną trzech ostatnich warstw (klasyfikatora). Uznano, że bardzo wysoka skuteczność tej sieci w zadaniu klasyfikacji obrazów oznacza, że potrafi ona efektywnie ekstrahować najważniejsze cechy obrazu wejściowego i dobrze sprawdzi się w problemie segmentacji. Zaletą sieci VGG jest stosowanie filtrów konwolucyjnych o niewielkich rozmiarach (3x3), co przekłada się na szybkość wykonywania obliczeń. Jej wadą jest bardzo trudny proces uczenia, wymagający ogromnej ilości danych i dużej mocy obliczeniowej. W niniejszym rozwiązaniu zastosowano więc *transfer learning* - wykorzystano sieć z wagami już wcześniej wstępnie wytrenowanymi na zbiorze obrazów ImageNet, a następnie jedynie douczono ją posiadanym zbiorem uczącym. Niemożliwe byłoby bowiem nauczenie sieci od podstaw.



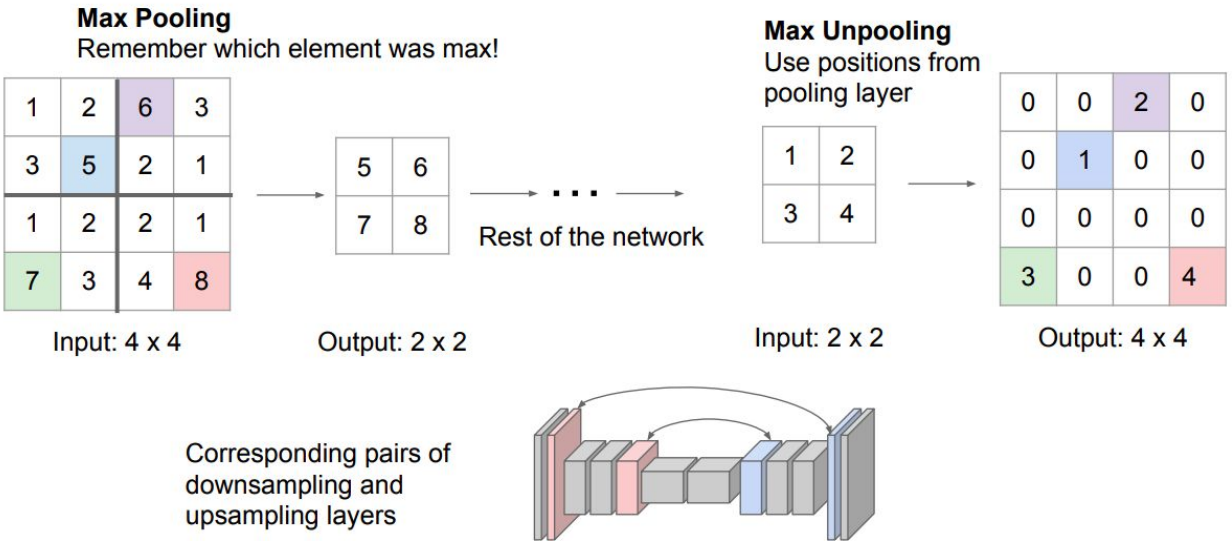
Rys. 2. Architektura sieci VGG-16. Źródło: [4]

Dekoder, którego schemat przedstawiono na rys. 3., służący odtworzeniu na wyjściu posegmentowanego obrazu, można podzielić na cztery bloki, złożone z kilku warstw:

- warstwy nadpróbkującej, która zwiększa wymiar macierzy,
- warstwy konkatenacji, łączącej wejście z poprzedniej warstwy z wyjściem z odpowiedniej warstwy enkodera,
- warstwy *ZeroPadding*, dodającej wokół macierzy ramkę o wartości zero; dzięki temu następna warstwa konwolucyjna nie zmniejsza wymiaru macierzy,

- warstwy konwolucyjnej z filtrem o rozmiarze 3x3,
- warstwy *BatchNormalization*, służącej normalizacji wejść na danym etapie, co poprawia proces uczenia.

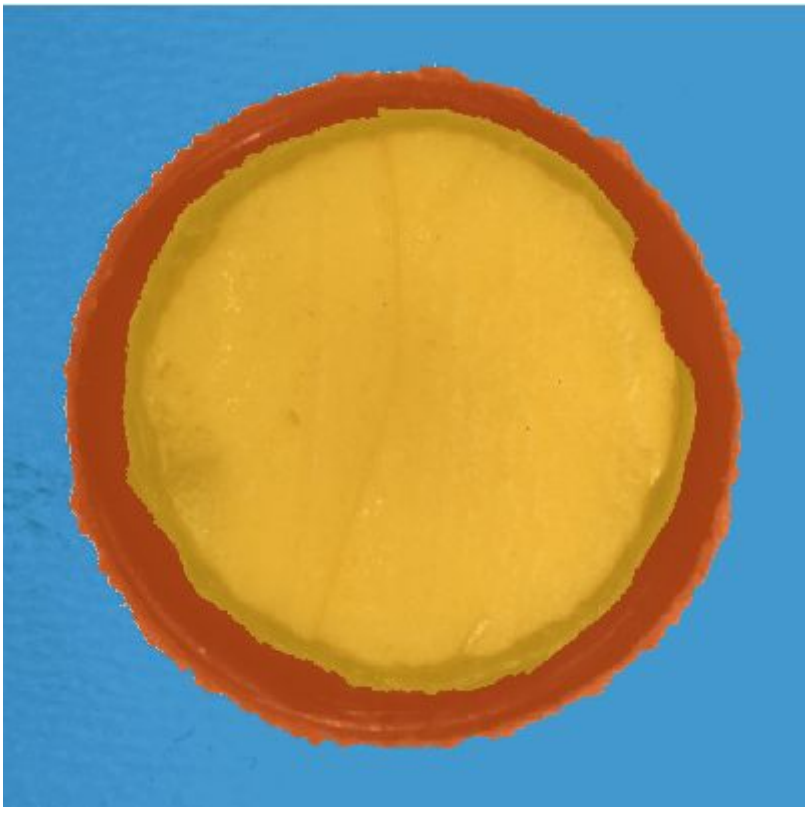
Ponadto, pierwszy blok nie zawiera warstwy *upsamplingu* oraz konkatenacji - jego wejściem jest wyjście z enkodera VGG-16. Konkatenacja ta, podobnie jak w rozwiązaniu zaproponowanym w oryginalnej sieci U-Net, pozwala na przekazanie lokalnych cech obrazu do dalszych warstw sieci. Na rys. 3 pokazano zasadę działania i zalety takiego rozwiązania. Dzięki zapamiętywaniu informacji, gdzie znajdowały się wartości pikseli przekazujące najwięcej informacji, można później w procesie zwiększania wymiaru obrazów wynikowy w kolejnych warstwach wykorzystać te właściwości. Na końcu dekodera umieszczono pojedyncza warstwę konwolucyjną z funkcją aktywacji *softmax*, która zwraca prawdopodobieństwo przynależności każdego piksela obrazu do poszczególnych klas.



Rys. 3. Grafika pokazująca zalety połączeń między warstwami enkodera i dekodera. Źródło: [5]

Obrazy referencyjne

Ze względu na to, że zastosowano uczenie nadzorowane konieczne było przygotowanie obrazów referencyjnych, zawierających informacje o klasach obiektów na obrazie. W pierwszej kolejności wykorzystano narzędzie *Image Labeler* z pakietu Image Processing and Computer Vision programu MATLAB[7].

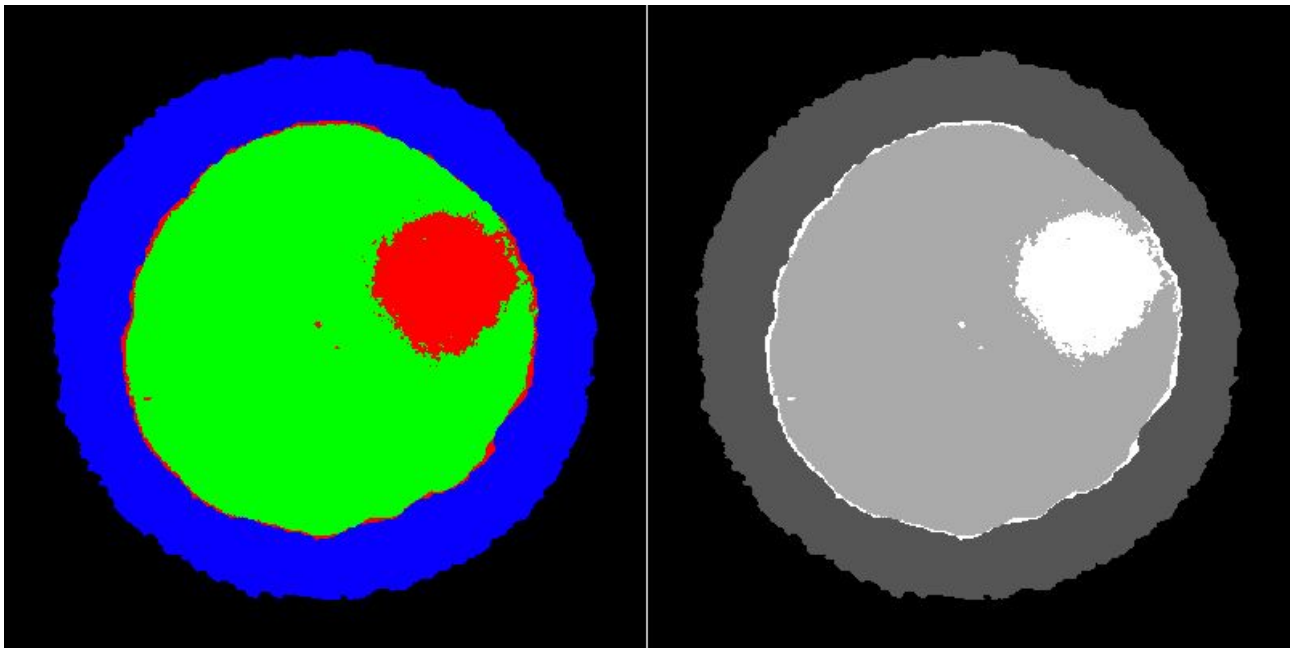


Rys. 4. Przykładowa etykieta otrzymana z wykorzystaniem Image Labeler.

Ze względu na nieregularny kształt podjęto próby etykietowania na podstawie koloru. Metoda nie sprawdziła się dla żołędzi o dużym procencie chorej tkanki, szczególnie, gdy występowała ona w pobliżu łupiny. Chcąc uzyskać jednolity format danych referencyjnych zdecydowano się na ręczne oznaczenie etykiet w programie graficznym Gimp. Założono, że sieć będzie rozróżniać cztery klasy pikseli:

- tło,
- łupinę,
- tkankę zdrową,
- tkankę chorą oraz szczeliny.

W pierwszej kolejności, korzystając z narzędzia tworzenia ścieżki, wydzielono obszar tła, łupiny oraz miąższu. Dzięki tej operacji rozwiązano problem przydzielania tej samej etykiety tkance chorej oraz łupinie. W następnej kolejności podzielono miąższ na tkankę chorą i szczeliny oraz tkankę zdrową. Dokonano tego konwertując wnętrze żołędzia do odcieni szarości oraz stosując binaryzację z progiem 110/255 [7]. Wynikiem powyższej operacji jest obraz RGB. Przykładową etykietę zaprezentowano poniżej na rysunku 5.1. Na rysunku 5.2 zaprezentowano obraz po konwersji do wartości 0-3.



Rys. 5.1 Wynik po etykietowaniu w programie gimp. (czarny - tło, niebieski - łupina, zielony - tkanka zdrowa, czerwony - tkanka chora i szczeliny).
Rys 5.2 Prezentacja etykiety w modelu programowym.

Zbiór danych

Zbiór obrazów, z wykorzystaniem którego miał zostać zrealizowany projekt, liczył 50 zdjęć żołędzi. Na rysunku 4. przedstawiono przykładowy obraz. Posiadane dane podzielono w sposób losowy na zbiory uczący, walidacyjny oraz testowy następująco:

- zbiór uczący - 45 zdjęć (w tym zbiór walidacyjny - 5 zdjęć),
- zbiór testowy - 5 zdjęć.

Rys. 6. Przykładowy obraz żołędzia z otrzymanego zbioru danych

Ponieważ zaimplementowana konwolucyjna sieć neuronowa miała zostać wytrenowana metodą uczenia nadzorowanego, konieczne było przygotowanie zbioru referencyjnych rozwiązań (ang. *ground truth*) problemu segmentacji dla wszystkich posiadanych obrazów. Zostało to przeprowadzone ręcznie z wykorzystaniem aplikacji GIMP. Uznano bowiem, że tak spreparowane wyniki segmentacji będą dokładniejsze od rezultatów, które można byłoby uzyskać, stosując konwencjonalne, klasyczne metody przetwarzania obrazu. Na rysunku 5. zaprezentowano jeden z przygotowanych obrazów referencyjnych, które miały zostać użyte w procesie uczenia. Postanowiono podzielić piksele obrazu na cztery klasy.

Ze względu na ograniczone zasoby pamięci posiadanych kart graficznych, na których uruchamiano obliczenia, rozdzielczość obrazów przekazywanych na wejście sieci neuronowej była zmniejszana do 480x480. Obrazy wyjściowe z sieci miały natomiast wymiary 240x240. Ich rozmiar był następnie rozszerzany do rozdzielczości oryginalnego obrazu - 869x869 - z wykorzystaniem funkcji *imresize* biblioteki OpenCV. Z uwagi na to, że sieć nie jest w stanie wyprodukować wyniku segmentacji z dokładnością pikselową, a ponadto dokładność taka nie jest w problemie segmentacji wymagana, uznano, że zwiększenie rozmiarów obrazów wynikowych z zastosowaniem interpolacji nie wpłynie znacząco na uzyskiwane rezultaty. W celu porównania czasów wykonania obliczeń testy przeprowadzono również na obrazach wejściowych o rozmiarach 256x256 oraz 384x384.

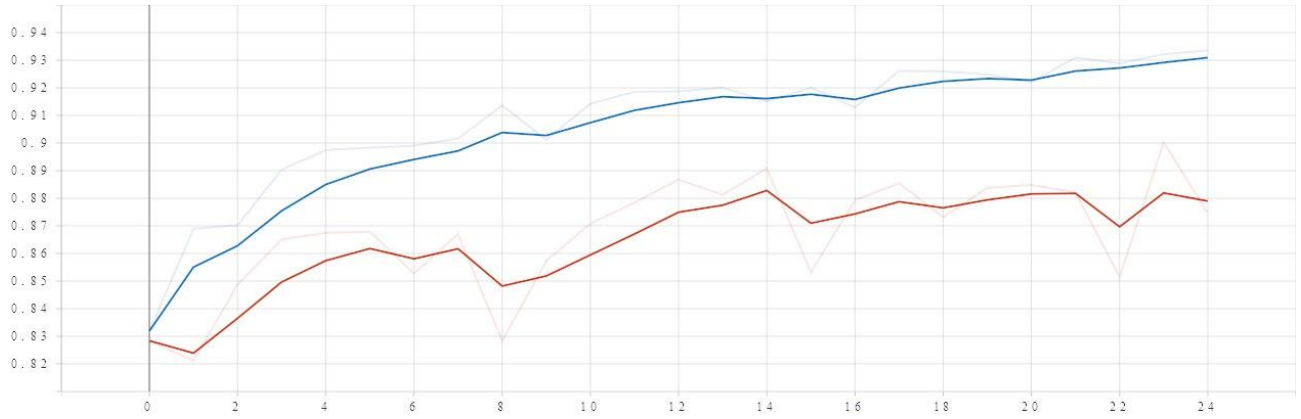
Aby zwiększyć efektywność uczenia sieci, zastosowano metodę rozszerzenia zbioru danych. Zestaw obrazów (ang. *batch*), wykorzystywanych w danym kroku epoki uczenia, był w losowy sposób przekształcany. Obrazy mogły zostać poddane operacjom takim jak:

- przerzucenia w pionie i poziomie,
- przycięcia,
- operacje afiniczne:
    - skalowanie,
    - translacja,
    - rotacja,
    - pochylanie (ang. *shear*).

Wykorzystywano ponadto filtry rozmywające - Gaussa, uśredniający oraz medianowy, a także operacje dodające na obrazie szum, czy też zmieniające jego kontrast i jasność.

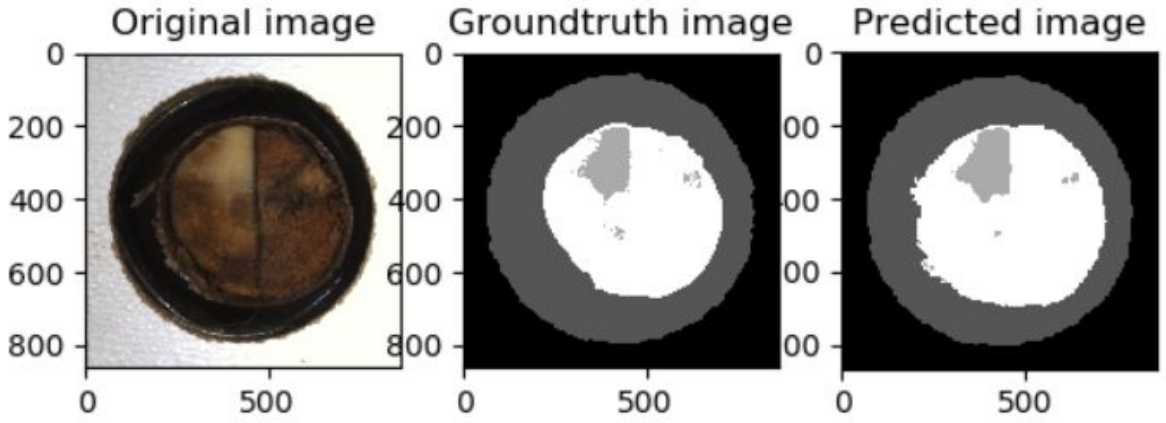
3. Wyniki

W trakcie analizy problemu przeprowadzono szereg eksperymentów polegający na zmianie parametrów sieci, odpowiednim podziale i doborze zbiorów treningowych, testujących oraz walidacyjnych. Opisaną wcześniej sieć uczono wielokrotnie w celu wyboru tej dla której wyniki segmentacji osiągnęły najlepszą dokładność. Podczas procesu uczenia sieci po każdej epoce jej wagi zapisywane były do pliku, aby można było skorzystać z nich w dalszym etapie. W celu śledzenia wyników podczas procesu uczenia korzystano z narzędzia Tensorboard. TensorBoard to narzędzie zapewniające możliwość pomiarów i wizualizacji potrzebnych podczas pracy z uczeniem maszynowym. Umożliwia śledzenie wskaźników eksperymentów, takich jak błąd i dokładność oraz pozwala na wizualizację grafu modelu. Przykładowy wykres można zobaczyć na rys. 6.

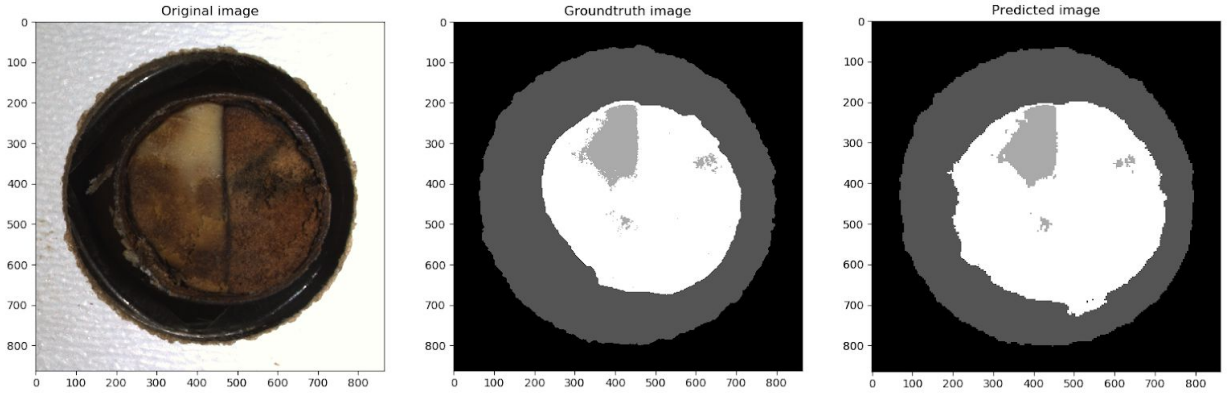


Rys.7. Przykładowy wykres dokładności zb. treningowego i walidacyjnego w zależności od epoki

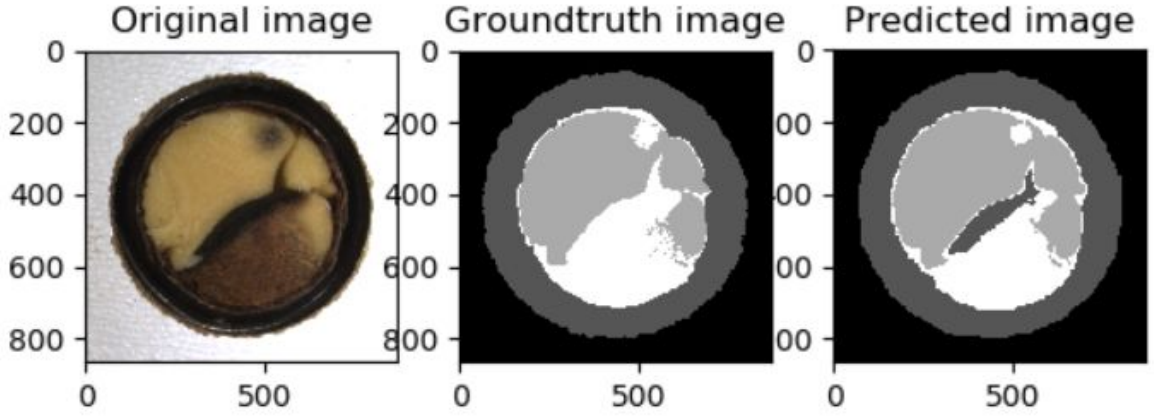
Przeprowadzono dwa główne eksperymenty. Podczas pierwszego rozszerzenie zbioru danych (ang. *training data augmentation*) zawierało w sobie funkcję rozmazania obrazu wykorzystując rozmycie gaussowskie, medianowe oraz filtr uśredniający. Do różnic należała również wartość parametrów *batch_size* oraz *val_batch_size*. W przypadku słabszej konfiguracji sprzętowej (zob. dodatek B) wartości te zostały ustawione na 1, natomiast w przypadku wydajniejszej platformy obliczeniowej na odpowiednio 4 oraz 1. Parametr *steps_per_epoch* ustawiono na domyślną wartość 512 dla starszej karty, a w przypadku nowszej karty graficznej ustalono na 100. Na rys. 7, 8, 9 i 10 zamieszczono porównanie wyników dwóch eksperymentów. Bardzo ciekawe jest porównanie rys. 9 i 10 gdzie dzięki wyszkolonemu enkoderowi w postaci architektury sieci VGG16 udało się uzyskać lepszy obraz wynikowy aniżeli stworzony do tego celu obraz wzorcowy. Znajdujący się na środku obrazu ubytek w miąższu został przyporządkowany jako łupina, a nie miąższ z objawami choroby, gdzie na obrazie wzorcowym te informacje nie zostały zawarte.



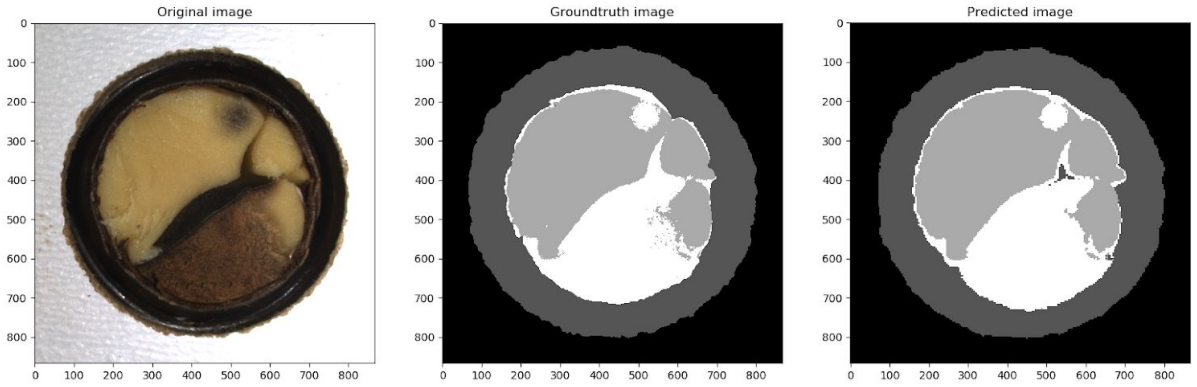
Rys.8. Porównanie oryginalnego obrazu, wzorca oraz wyniku sieci wykorzystującej rozmycia w procesie rozszerzania zbioru treningowego



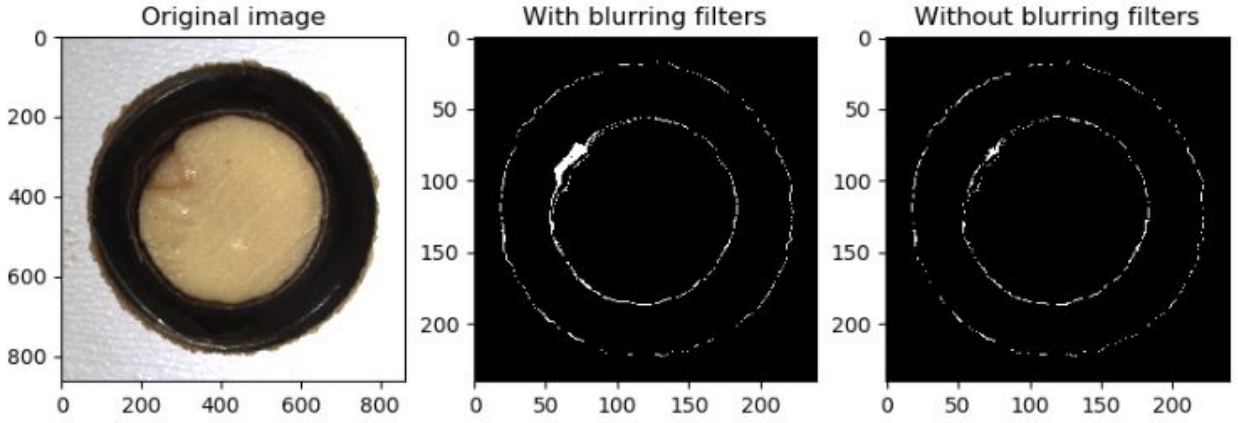
Rys.9. Porównanie oryginalnego obrazu, wzorca oraz wyniku sieci nie wykorzystującej rozmycia w procesie rozszerzania zbioru treningowego



Rys.10. Porównanie oryginalnego obrazu, wzorca oraz wyniku sieci wykorzystującej rozmycia w procesie rozszerzania zbioru treningowego



Rys.11. Porównanie oryginalnego obrazu, wzorca oraz wyniku sieci nie wykorzystującej rozmycia w procesie rozszerzania zbioru treningowego



Rys.12. Porównanie oryginalnego obrazu i masek pokazujących punkty niezgodne obrazów wynikowych sieci z obrazem wzorcowym

W celach porównawczych uczenie oraz testy na wydajniejszej konfiguracji sprzętowej przeprowadzono dla kilku rozmiarów obrazu wejściowego - 256x256, 384x384 oraz 480x480. Dla wyższych rozdzielczości w trakcie uczenia pojawiały się liczne ostrzeżenia o braku dostępnej pamięci operacyjnej na karcie graficznej, na skutek czego zrezygnowano z dalszych testów. Uzyskiwane czasy zarówno uczenia, jak i późniejszych obliczeń na zbiorze testowym byłyby bowiem w oczywisty sposób wyraźnie gorsze. Wybrane parametry modelu ustalono następująco:

- *batch_size* - 4,
- *val_batch_size* - 1,
- *epochs* - 20,
- *steps_per_epoch* - 100.

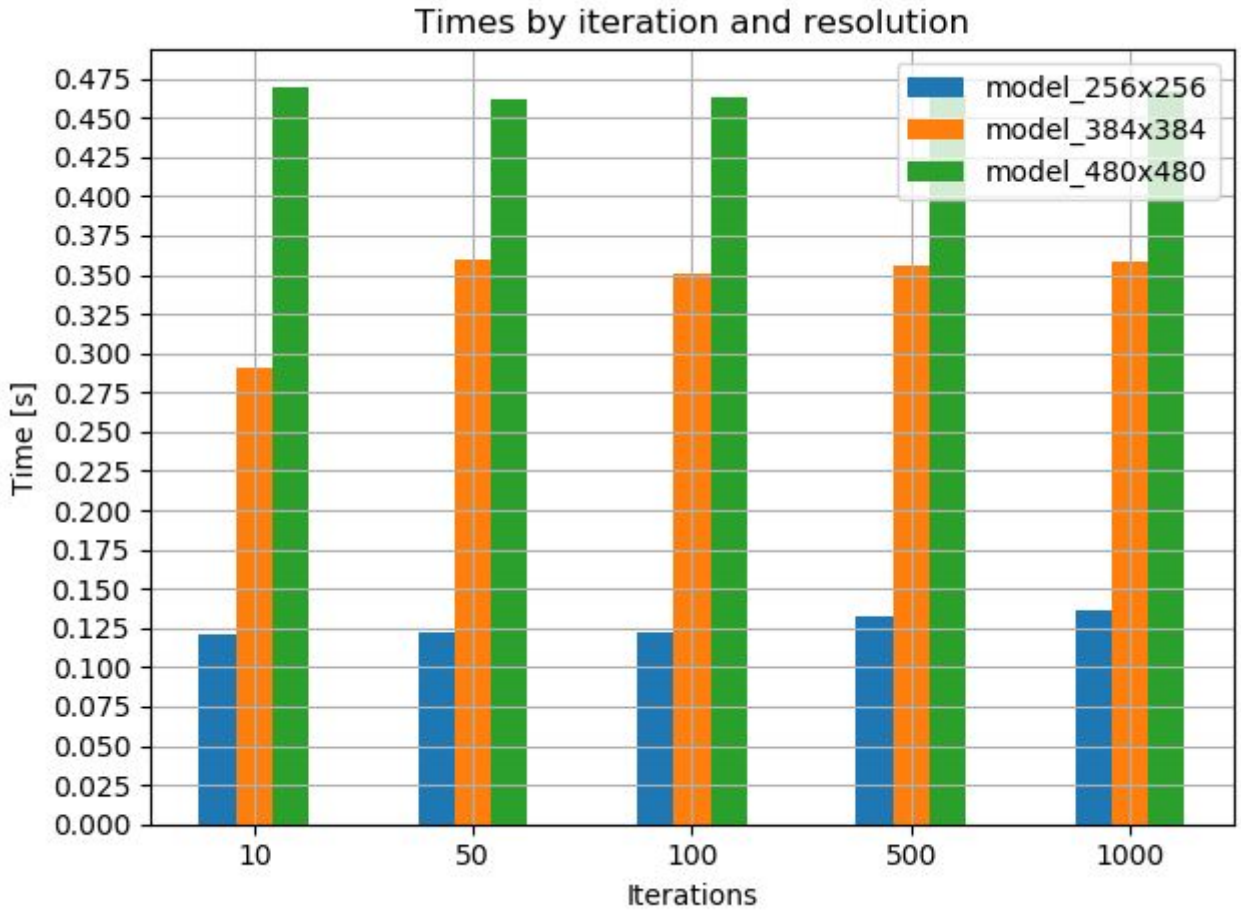
Dla każdej z wersji modelu uzyskano skuteczność zarówno na zbiorze uczącym, jak i walidacyjnym zbliżoną do 98%. Średnie czasy uczenia jednej epoki dla modeli z poszczególnymi rozmiarami obrazów wejściowych przedstawiono poniżej:

- 256x256 - 32 s,

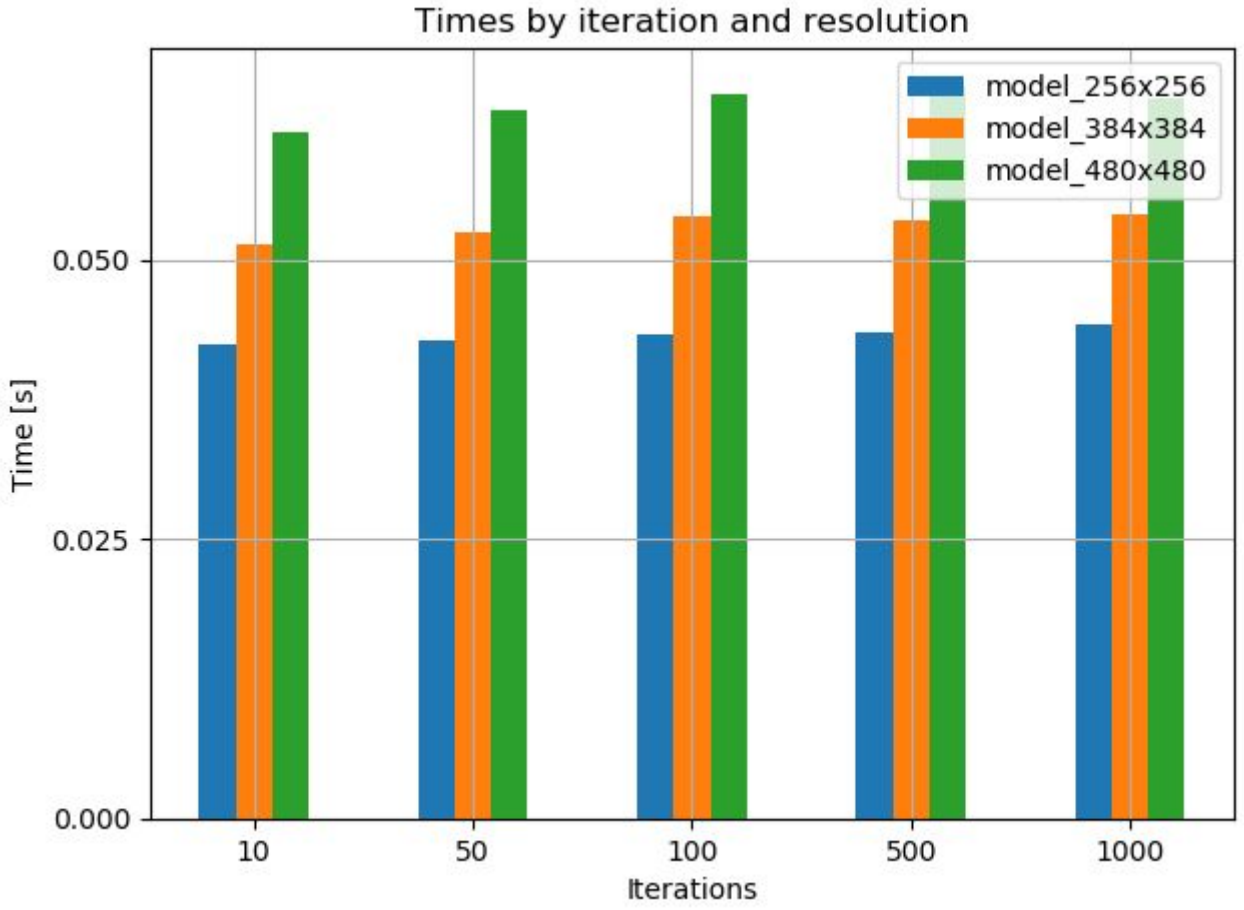
- 384x384 - 37 s,
- 480x480 - 47s.

Tak jak należało się spodziewać, średni czas trwania jednej epoki uczenia rósł wraz ze wzrostem rozdzielczości obrazów wejściowych - obliczenia wewnątrz sieci były wykonywane na większych macierzach.

Sprawdzono również czas wykonywania obliczeń z wykorzystaniem tych trzech modeli. Testy przeprowadzono na obu dostępnych platformach sprzętowych. Wyniki przedstawiono poniżej.



Rys.13. Wykres słupkowy pokazujący średni czas przetwarzania obrazu w zależności od modelu i liczby iteracji dla konfiguracji sprzętowej nr 1



Rys.14. Wykres słupkowy pokazujący średni czas przetwarzania obrazu w zależności od modelu i liczby iteracji dla konfiguracji sprzętowej nr 2

4. Analiza modelu

Zastosowany w ramach niniejszego projektu model głębokiej sieci neuronowej opartej na architekturze autoenkodera zaimplementowany z wykorzystaniem biblioteki TensorFlow charakteryzuje się następującymi parametrami:

- Łączna liczba parametrów: 12 324 100
- Liczba parametrów trenowalnych: 12 322 180
- Liczba parametrów nietrenowalnych: 1920
- Łączna liczba warstw: 36
- Warstwa wejściowa przyjmująca obraz o rozdzielczości: 256x256, 384x384, 480x480 (w zależności od wersji)
- Liczba warstw konwolucyjnych: (VGG) 13, (Dekoder) 5
- Liczba warstw MaxPooling: (VGG) 5
- Liczba warstw BatchNormalization: (Dekoder) 4
- Liczba warstw ZeroPadding2D: (Dekoder) 4
- Liczba warstw Concatenate: (Dekoder) 3
- Waga modelu to około 145 MB

5. Bibliografia

[1] https://github.com/divamgupta/image-segmentation-keras [ostatni dostęp: 08.12.2019 r.]
[2] Ronneberger, O. et al. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Medical Image Computing and Computer-Assisted Intervention -- MICCAI 2015. Cham, Switzerland: Springer, 2015, pp. 234-241. URL: arXiv:1505.04597 [ostatni dostęp: 08.12.2019 r.]
[3] Simonyan, K., Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. URL: arXiv:1409.1556 [ostatni dostęp: 08.12.2019 r.]
[4] https://neurohive.io/en/popular-networks/vgg16/ [ostatni dostęp: 08.12.2019 r.]
[5] https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html [ostatni dostęp: 08.12.2019 r.]
[6] https://docs.gimp.org/2.10/en/gimp-tool-threshold.html [ostatni dostęp: 16.12.2019 r.]
[7] https://www.mathworks.com/help/vision/ref/imagelabeler-app.html [ostatni dostęp: 08.01.2019 r.]

6. DODATEK A: Konfiguracje sprzętowe

Do obliczeń podczas eksperymentów przeprowadzonych w ramach projektu użyte zostały dwie platformy obliczeniowe. Poniżej znaleźć można dokładne specyfikacje ich konfiguracji sprzętowych.

1. Konfiguracja nr 1
    - CPU: Intel i7-4700HQ
        - Liczba rdzeni: 4
        - Liczba wątków: 8
        - Maks. częstotliwość taktowania: 3.40 GHz
        - Technologia: 22 nm
    - GPU: NVIDIA GeForce GTX 850M
        - Liczba rdzeni CUDA: 640
        - Częstotliwość taktowania rdzenia: 876 MHz
        - Pamięć: 2 GB
        - OpenGL: 4.5
        - OpenCL: 1.1
        - Technologia: 28 nm
    - RAM:
        - Częstotliwość taktowania: 1600 MHz
        - Pamięć: 12 GB

2. Konfiguracja 2
    - CPU: AMD Ryzen 7 3700X
        - Liczba rdzeni: 8
        - Liczba wątków: 16

    - Maks. częstotliwość taktowania: 4.40 GHz
    - Technologia: 7 nm
- GPU: NVIDIA GeForce RTX 2070 SUPER
    - Liczba rdzeni CUDA: 2560
    - Częstotliwość taktowania rdzenia: 1605 MHz (1770 MHz w trybie boost)
    - Pamięć: 8 GB GDDR6
    - OpenGL: 4.5
- RAM:
    - Częstotliwość taktowania: 3200 MHz
    - Pamięć: 16 GB

7. DODATEK B: Dokumentacja techniczna

Podczas pracy nad projektem wykorzystywane było środowisko Pycharm IDE, które usprawnia pisanie kodu i zarządzanie projektami wykorzystującymi język skryptowy wysokiego poziomu Python. Poniżej zamieszczono listę narzędzi i bibliotek, które zostały wykorzystane w projekcie:

- Interpreter Python w wersji 3.7.0.
- The NVIDIA CUDA Toolkit - wersja 10.0
- Sterowniki NVIDIA® GPU - wersja >= 410.0
- NVIDIA cuDNN - wersja >= 7.4.1
- Tensorflow-gpu - wersja 2.0.0
- Keras - wersja 2.3.1
- OpenCV-python - wersja 4.1.0
- Numpy - wersja 1.16.4
- Matplotlib - wersja 3.1.0
- Gimp - wersja 2.10.12

8. DODATEK C: Zawartość płyty DVD

- Kod źródłowy projektu
- Wyuczone modele dla różnych rozmiarów obrazów wejściowych
- Zbiór obrazów uczących i testowych
- Sprawozdanie w formacie pdf

Drzewo plików projektu zbudowane jest następująco:

- /SRC
    - /keras_segmentation - zawiera pliki z kodem źródłowym modelu sieci ([1] z modyfikacjami)
    - /modele_256_384_480 - zawiera zapisane wyuczone modele dla różnych rozmiarów obrazów wejściowych
    - /PRZEKROJE - zawiera zbiór obrazów uczących
    - /PRZEKROJE_TEST - zawiera zbiór obrazów testowych

    - /PRZEKROJE_VAL - zawiera zbiór walidacyjny
    - /PRZEKROJE_ANNOTATIONS - zawiera ręcznie opracowane wyniki referencyjne dla zbioru uczącego
    - /PRZEKROJE_ANNOTATIONS_TEST - zawiera ręcznie opracowane wyniki referencyjne dla zbioru testowego
    - /PRZEKROJE_ANNOTATIONS_VAL - zawiera ręcznie opracowane wyniki referencyjne dla zbioru walidacyjnego
    - /train_new.py - skrypt wywołujący funkcję uczącą model
    - /test_new.py - skrypt pozwalający na przetestowanie wyuczonego modelu na wybranych obrazach
    - /test_time_final.py - skrypt pozwalający na zmierzenie czasu wykonywania obliczeń
    - /show_time_final.py - skrypt rysujący wykres zmierzonych czasów trwania obliczeń
- /DOC
    - /sprawozdanie_pr19aaw01.pdf - Sprawozdanie w formacie pdf

9. DODATEK D: Historia zmian

Tabela 1 Historia zmian.

Autor	Data	Opis zmian	Wersja
A. O.	03.12.2019	Utworzono szkielet dokumentu Dodano wstęp Dodano bibliografię	0.0
A. O.	08.12.2019	Poprawiono wstęp Dodano opis rozwiązania	0.1
D. G.	08.12.2019	Opis wstępnych wyników Opis ulepszonego upsamplingu	0.2
D. G.	08.12.2019	Dokumentacja techniczna	0.3
D. G.	04.01.2020	Uzupełnienie dokumentacji Porównanie czasowe modeli	0.4
A. O.	05.01.2020	Drobne poprawki Dodanie konfiguracji nr 2 Uzupełnienie dokumentacji Wyniki uczenia	0.5
R. R.	10.01.2020	Uzupełnienie dokumentacji Opis pozyskania obrazów referencyjnych Poprawienie semestru na stronie tytułowej Przeniesienie informacji o etykietach ze zbiorów danych Zmiana numeracji rysunków od 4 na większy o 1 Zmiana obrazu etykiety w modelu programowym	0.6
D. G.	12.01.2020	Porównanie czasowe działania modeli	0.7