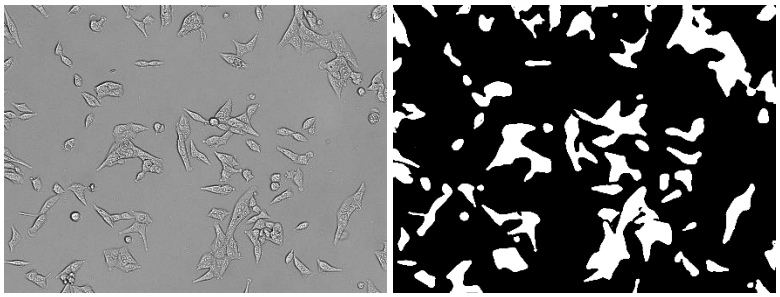
Segmentacja komórek na zdjęciach mikroskopowych w świetle przechodzącym.

Wstęp

Celem pracy było stworzenie modelu, który na podstawie zdjęć w skali szarości, ma za zadanie przewidywać pozycje komórek, gdzie wynikiem końcowym jest macierz reprezentująca maskę binarną pozycji komórek.

Źródło danych

Dane zostały pozyskane z projektu grantowego i reprezentują komórki mysiej linii nowotworowej, PAN_02. Pierwotnie są one w skali szarości 16-bitowe, o wymiarach 1936 x 1456 pixeli [Rys. 1].



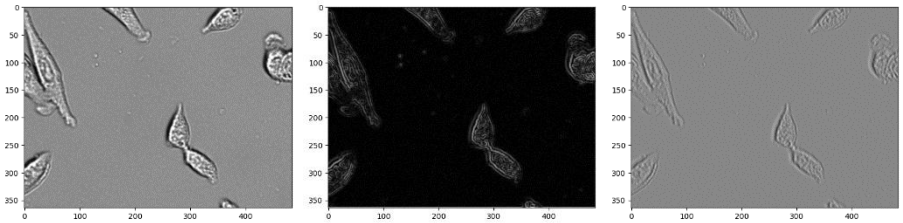
Rysunek 1 Przykładowe zdjęcie komórek w świetle przechodzącym i maski binarnej położeń komórek.

Wzorzec [Rys. 1] dla tych danych został pierwotnie utworzony automatycznie za pomocą operacji na zdjęciach w skali szarości, które następnie były delikatnie popoprawiane/odrzucane ręcznie (648 zdjęć).

Sposób wprowadzania danych do sieci

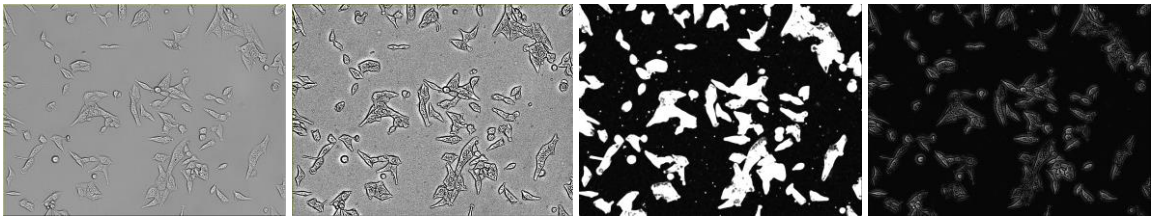
Początkowo próbowałem wprowadzać do sieci tylko surowe zdjęcie wraz z maską, co powodowało, że sieć się nie uczyła – powodem był brak innych kanałów, na podstawie których sieć mogła tworzyć dodatkowe relacje. Dodanie już drugiego kanału, nawet w postaci tego samego zdjęcia spowodowało drastyczną różnicę w uczeniu się sieci.

Z tymi wnioskami podczas wprowadzania danych poddawałem je filtrom Sobela – w celu wyszczególnienia granic obiektów (komórek) – zacząłem wprowadzać do sieci zdjęcia składającego się z 3 kanałów [Rys. 2]– zdjęcia surowe, zdjęcie podane pełnemu filtru Sobela (po wypadkowej x i y), oraz zdjęcie różniczkowanie po płaszczyźnie X. Sieć zasilana tego typu danymi była już w stanie osiągać wysoką precyzję w stosunku do dostarczonego wzorca.



Rysunek 2 Rysunek przedstawiający 3 kanały wprowadzane do sieci. Odpowiednio: surowe zdjęcia, filtr Sobel'a, oraz wynik różniczkowania poziomego.

Jako kolejny etap, starałem się wprowadzić 4 kanały, które w moim odczuciu dawałyby najwięcej możliwych informacji na temat obiektów na zdjęciach [Rys. 4]



Rysunek 3 Rysunek przedstawiający 4 kanały. Odpowiednio: surowe zdjęcie, zastosowanie filtru Unsharp Mask, zastosowanie wariancji, oraz usunięcie tła za pomocą "rolling ball".

Efekt ten jednak spowodował, że sieć „nie chciała" się uczyć. Prawdopodobnym powodem tego było, że architektura U-Net jest skonstruowana pod „obsługę" zdjęć 2,3 – kanałowych, a w tym przypadku 4 kanał mógł odpowiadać jako szum burzący konstruowane relacje. Jednak! Po usunięciu jednego z 4 kanałów sieć dalej nie chciała się uczyć, co wskazuje na to, że dobrane przeze mnie filtry nie są odpowiednie do budowania odpowiednich relacji, przez co wróciłem do pierwotnego sposobu i wykorzystanie filtru Sobela. Aby nie tracić na ilości szczegółów, zdjęcia końcowo podzieliłem na 64 części o rozdzielczości 182 x 242 pixeli, oraz zastosowałem lekkie rozmycie w celu eliminacji potencjalnych szumów (już po zastosowaniu filtru Sobela).

Archtektura sieci U-Net

Sieć U-Net jest to architektura sieci neuronowej, przeznaczona do segmentacji obrazów. Jest to sieć konwolucyjna z architekturą encoder-decoder.

Encoder część składa się z serii warstw konwolucyjnych i warstw max-pooling'u, które służą do redukcji rozmiaru obrazu oraz do ekstrakcji cech. Warstwy te działają jak filtry, które rozpoznają różne kształty i tekstury na obrazie. Im niżej znajduje się warstwa w encoderze, tym szerszy jest jej zasięg i tym bardziej abstrakcyjne są informacje, które przechwytuje.

Decoder część składa się z serii warstw konwolucyjnych i warstw up-sampling, które służą do powiększania rozmiaru obrazu i do rekonstrukcji szczegółów. Warstwy te służą do przywracania szczegółów utraconych podczas kompresji danych w encoderze. Warstwy up-sampling służą do zwiększenia rozmiaru obrazu, a warstwy konwolucyjne do wykrywania cech na tym powiększonym obrazie.

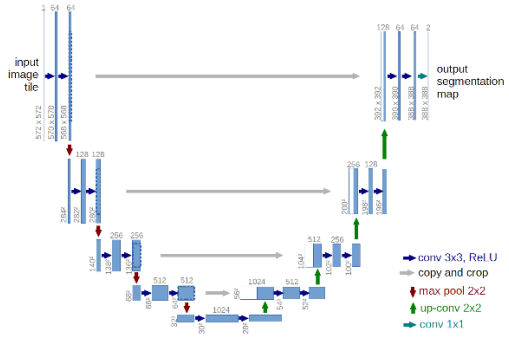
Na końcu decoder części, U-Net generuje mapę segmentacji, która jest przeskalowana do rozmiarów wejściowych. Mapa ta przedstawia prawdopodobieństwo przynależności każdego piksela do danej klasy.

Ważną cechą U-Net jest to, że warstwy encodera połączone są bezpośrednio z odpowiednimi warstwami decodera, co pozwala na przekazywanie nisko poziomowych cech z encodera do decodera. Dzięki temu decoder ma dostęp do informacji o szczegółach na wysokim poziomie abstrakcji, co pozwala na generowanie precyzyjniejszych map segmentacji.

Droga do optymalizacji sieci

Architektura

Po zbadaniu literatury, standardowy układ sieci [Rys. 4] przewiduje 4 etapy „dowsize'ingu" z 2 konwolucjami na każdym etapie, co końcowo dawało nam zwiększenie liczby kanałów do 1024!



Rysunek 4 Przykładowa struktura sieci U-Net. Źródło :https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/.

Powodowało to, że końcowo dochodziliśmy do ponad 3 milionów parametrów. W trakcie pracy nad zmniejszeniem tej sieci, doszedłem do najlepszych wyników zmieniając liczbę etapów z 4 na 3, oraz liczbę końcowo otrzymywanych kanałów na 24 (zamiast 3 -> 64 -> 128 -> 256 -> 512 -> 1024, 3 -> 6 -> 12 -> 24). Co spowodowało, że końcowo sieć miała niewiele ponad ~ 84 tysiące parametrów. W przypadku innych architektur i typów sieci ta liczba wydaję się ogromna, ale w przypadku architektury U-Net i tego typu danych, potrzebna jest wysoka liczba parametrów do dokładnego odwzorowania relacji międzykanałowych.

Przekształcenia konwolucyjne

Kolejną rzeczą, za którą się zabrałem była praca nad rozmiarem jądra konwolucyjnego - standardowo podczas obu konwolucji w etapach downsize'ingu miało ono wymiary 3x3 z paddingiem jeden. Zauważyłem jednak, że zmiana rozmiaru jądra w drugiej konwolucji (gdzie pierwsza konwolucja to input_size -> output_size, a druga to ouput_size -> output_size) na 9x9 z paddingiem 4 również poprawiło skuteczność uczenia się sieci. Powodem tego jest: 1) zwiększona liczba parametrów już przekształconego obrazu, co jednocześnie wpływa na relację również z etapem upsize'ingu, oraz 2) zwiększenie paddingu pozwoliło na zachowanie większej informacji na brzegach zdjęć.

Optymalizator

Jako optymalizator wybrałem Adam'a z learning rate = 0.001, głównie z dwóch powodów:

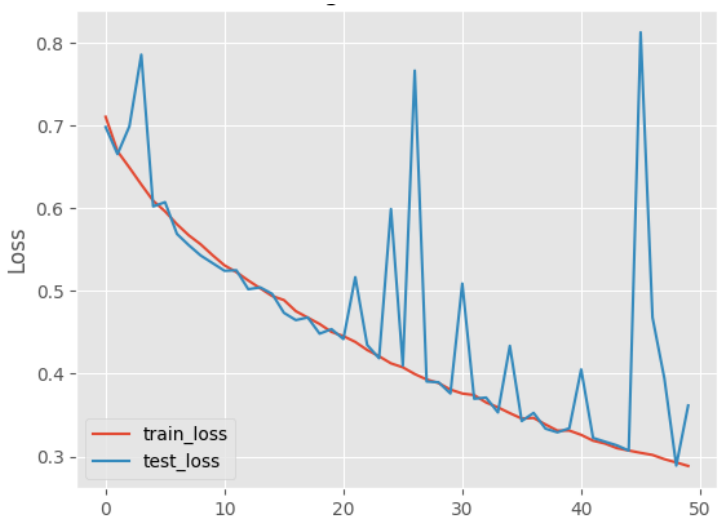
1) Jest on adaptacyjnym optymalizatorem – czyli automatycznie dostosowuje wartość kroku uczenia do różnych parametrów modelu. Jest to szczególnie ważne w tej architekturze sieci, ponieważ zawiera ona wiele warstw i różne parametry/relację, które mogą mieć różną skalę.
2) Wykorzystuje on momentum, dzięki czemu nie obawiam się blokady w przypadku dojścia do minimum lokalnego, na co narażona jest ta sieć ze względu na ilość parametrów

Funkcja strat

Jako funkcję strat wybrałem BCEWithLogitsLoss, czyli błąd krzyżowy binarny z logitami, ponieważ jest ona specjalnie zaprojektowana do klasyfikacji binarnej.

Rozmiar próbkowania

Rozmiar próbkowania okazał się największym wyzwaniem, ponieważ ze względu na chęć otrzymania jak najlepszych wyników z 648 zdjęć wyłoniłem 10, w których szczegółowo poprawiałem ręcznie automatycznie otrzymane maski, czyli końcowo uzyskałem 640 zdjęć wejściowych, które podzieliłem na treningowe i testowe w stosunku 0.8:0.2 (czyli 512 treningowych i 128 testowych). Ze względu na architekturę tej sieci, oraz charakter danych (bardzo duża różnorodność/charakterystyka zdjęć), zbyt małe próbkowanie powodowało nagłe „wyskoki" wartości strat [Rys. 5].



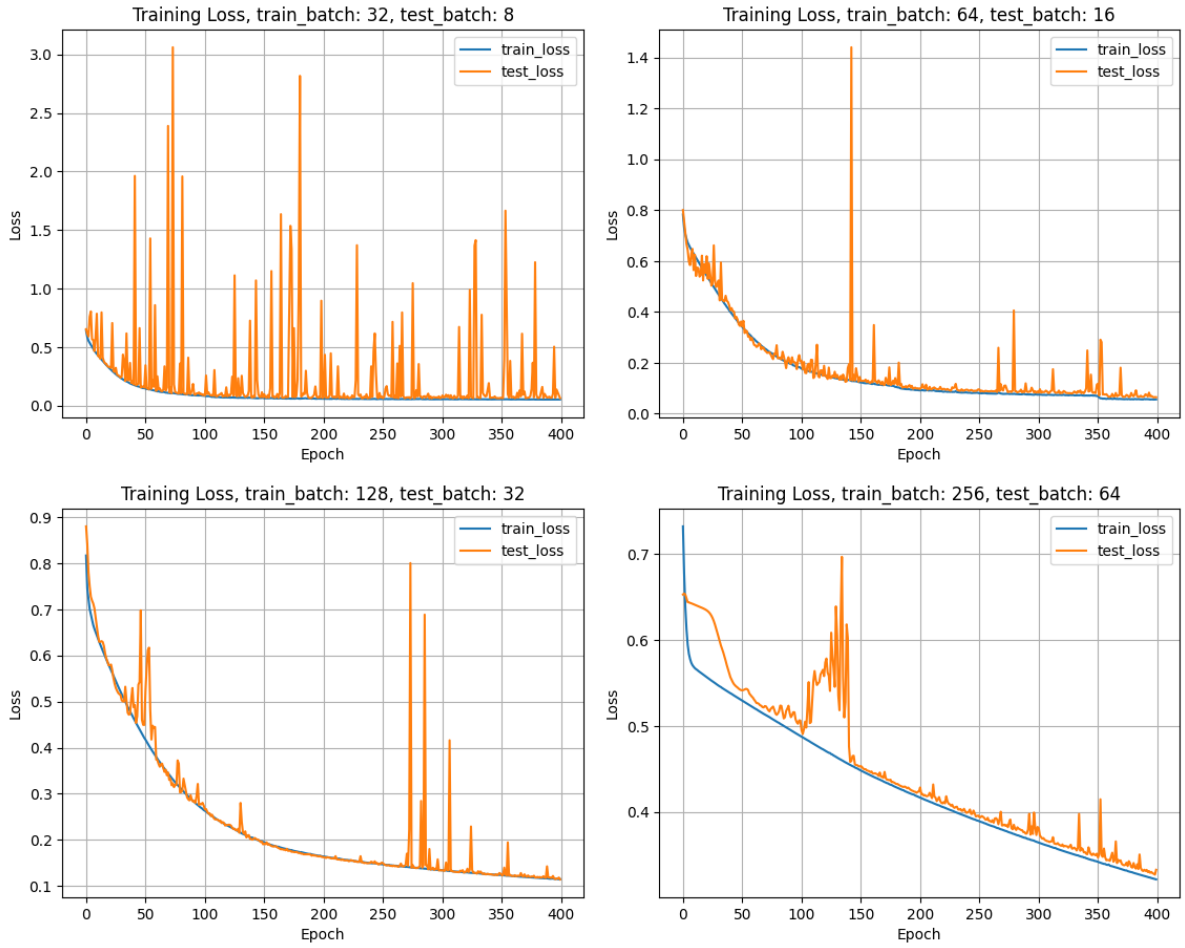
Rysunek 5 Wykres przedstawiający ewolucję wartości funkcji kosztów na zbiorze treningowym (kolorem czerwonym), oraz testowym (kolorem niebieskim)

Z tego też powodu przeprowadziłem testy różnych wielkości próbkowania danych treningowych i testowych, z ewaluacją w epoch'ach 50, 100, 200, 400 [Tabela 1][Rys. 6].

Tabela 1 Wartości train_loss, test_loss, accuracy (jako DC) dla różncyh wielkości próbkowania w kolejnych seriach epoch.

train_batch	test_batch	no. Epochs											
		50			100			200			400		
		Train_loss	Test_loss	Accuracy	Train_loss	Test_loss	Accuracy	Train_loss	Test_loss	Accuracy	Train_loss	Test_loss	Accuracy
32	8	**0,149**	**0,153**	**0,877**	**0,084**	**0,093**	**0,891**	**0,057**	**0,065**	**0,9**	**0,052**	**0,064**	**0,9**
64	16	**0,35**	**0,363**	**0,872**	**0,181**	**0,205**	**0,855**	**0,092**	**0,099**	**0,897**	**0,056**	**0,065**	**0,905**
128	32	**0,421**	**0,449**	**0,732**	**0,265**	**0,275**	**0,811**	**0,164**	**0,163**	**0,899**	**0,115**	**0,115**	**0,909**
256	64	**0,53**	**0,542**	**0,828**	**0,488**	**0,506**	**0,84**	**0,417**	**0,424**	**0,868**	**0,322**	**0,327**	**0,882**

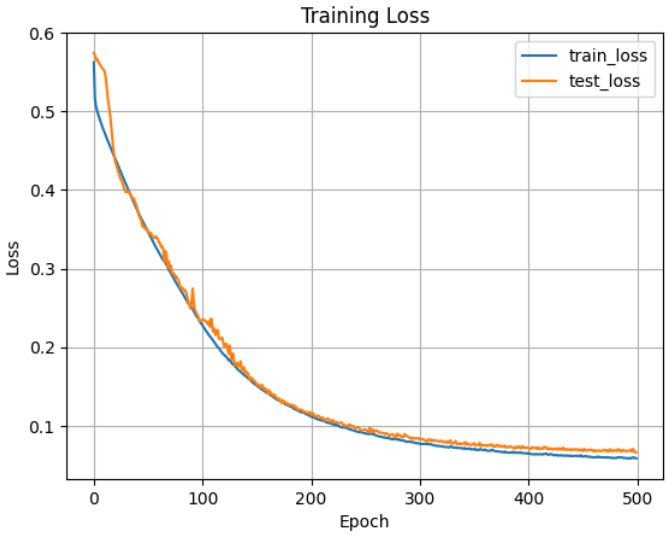
Na podstawie przeprowadzonych testów [Rys. 6], doszedłem do wniosku, że kombinacja wielkość próbkowania danych treningowych 128 i danych testowych 32 jest najoptymalniejszym połączeniem dla tego problemu. W przypadku mniejszych wartości widoczne są duże odchyły test_loss w porównaniu do train_loss,a większa wielkość próbkowania, pomimo lepszych odchyłów, nie zbliżyła się do wyniku otrzymanego przy wybranej wielkości próbkowania [Tabela 1] (0,909 vs 0,882).



Rysunek 6 Ewolucja train_loss, test_loss w kolejnych poechach dla różnych kombinacji wielkości próbkowania.

Otrzymane wyniki

Po dobraniu parametrów próbkowania, sieć została puszczona do nauki na 500 epoch (Rys. 7).



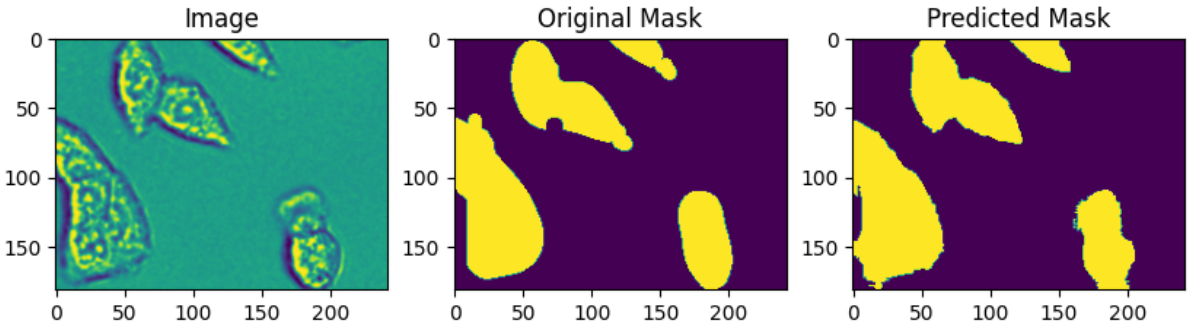
Rysunek 7 Ewolucja train_loss, test_loss w kolejnych epoch'ach.

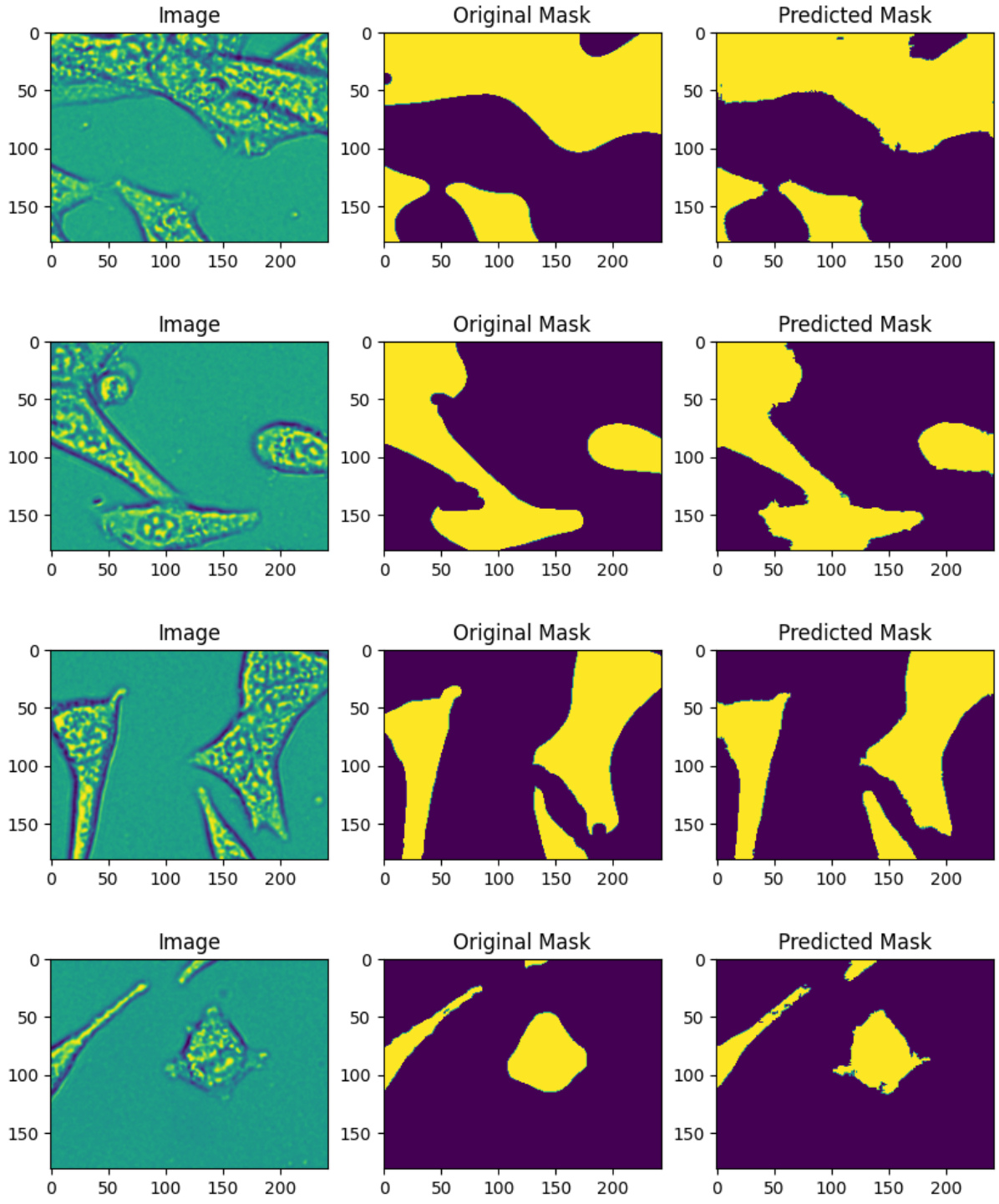
Co ciekawe, podczas tego uczenia, nie doszło do nagłych odchyleń wartości train_loss, i widoczne jest bardzo ładne nałożenie wykresów obu wartości strat.

Do oceny skuteczności otrzymanych masek wykorzystałem Dice coefficient (DC), który jest stosunkiem sumy pikseli wspólnych w obrazie oryginalnym i przewidywanym. Wartości są z przedziału 0, 1, gdzie 0 to brak podobieństwa, a 1 to idealne podobieństwo, dodatkowo metryki F1, czułość, oraz precyzja

Za pomocą otrzymanej sieci, udało się uzyskać bardzo dobry wynik DC na poziomie 0.904, pozostałe metryki: F1 = 0.904, czułość = 0.912, precyzja = 0.909

W celach reprezentatywnych kilka przykładów porównań poniżej. Widać, że pomimo niedoskonałości wzorcowych mask w niektórych miejscach, sieć nauczyła się poprawnie rozpoznawać obszary komórek (wszystkie przykłady ze zbioru testowego w notatniku).





Dyskusja

Udało się skutecznie odtworzyć i zmodyfikować architekturę sieci U-Net, która wraz z przygotowanymi danymi z wysoką precyzją zwraca maski binarne. Wynik precyzji powyżej 0.9 w porównaniu z pracą nad podobnym zagadnieniem (Falk, T., Mai, D., Bensch, R. et al. U-Net: deep learning for cell counting, detection, and morphometry. Nat Methods 16, 67–70 (2019). https://doi.org/10.1038/s41592-018-0261-2) również wskazuje na duży sukces.

Oczywiście widoczne też są jej niedoskonałości w otrzymywanych maskach, ale w głównej mierze wynika to z małej ilości/niedoskonałości danych. Przy większej ilości danych na pewno było by też można jeszcze bardziej zoptymalizować hiperparametry tej sieci.

Kolejnym etapem, który podejmuję w rozwoju tej sieci, to jest znalezienie sposobu na detekcję krawędzi przez sieć i oddzielanie komórek na otrzymanych maskach binarnych. Możliwe, że zostanie też zmieniona architektura na Mask R-CNN, która wydaje się lepszą metodą do tego zagadnienia, gdzie jednocześnie pozwoli na klasyfikacje stanu komórek. Mam nadzieję, że dobre wyniki uda się już przedstawić na najbliższej szkole zimowej podczas sesji posterowej – „Development of a Convolutional Neural Network (CNN) model for automated cell counting and classification in transmitted light microscopy images of cancer cell lines “, na co serdecznie Pana zapraszam do przyjścia i obejrzenia.