Warsztaty uczenia maszynowego - projekt

Paweł Drabarek, Mateusz Fabisiewicz, Jakub Gazewski,
Adrian Goworek, Adam Wodzisławski

Kwiecień 2022

1. Temat i opis

Tematem projektu jest stworzenie rozwiązania wykorzystującego uczenie maszynowe służącego do analizy obrazów, na których mogą znajdować się ryby oceaniczne. Zadaniem programu jest zakwalifikowanie obrazu do jednej z 8 kategorii - 6 z nich dotyczy gatunku ryby, która znajduje sie na obrazie (mogą to być – tuńczyk biały, tuńczyk wielkooki, koryfena, strojnik, gatunek rekina lub tuńczyk żółtopłetwy), pozostałe dwie odpowiadają sytuacji gdy na obrazie nie znajduje się ryba lub znajduje się ryba innego gatunku niż te rozpoznawane.

Szczegóły dotyczące danych i zadaniu do wykonania znajdują się pod linkiem [1].

2. Założenia technologiczne

Program zostanie wykonany w języku Python. Zostanie wykorzystana biblioteka umożliwiająca korzystanie z sieci neuronowej, prawdopodobnie PyTorch lub TensorFlow, nasze doświadczenie podczas pisania tego etapu dokumentacji nie pozwala nam jeszcze określić która z wymienionych bibliotek najlepiej pasuje do naszego problemu (być może ostatecznie wybrana zostanie zupełnie inna, nie wymieniona).

3. Podział pracy

W projekcie możemy rozróżnić takie etapy pracy jak:

- przygotowanie zbiorów treningowych i walidacyjnych;
- ustalenie odpowiedniego modelu, wyodrębnienie atrybutów i ustalenie hiperparametrów;
- implementacja i trening sieci neuronowej;
- testy modelu;
- sporządzenie dokumentacji.

W projekcie porównamy działanie modeli bez fragmentacji (model A) i z fragmentacją (model B) wejściowego obrazu. Proponujemy następujący podział prac dla poszczególnych członków zespołu:

- Paweł Drabarek: przygotowanie zbiorów treningowych i walidacyjnych, implementacja modelu B, pomoc w sporządzaniu dokumentacji;
- Mateusz Fabisiewicz: ustalenie szczegółów modelu B i jego implementacja, sporządzenie dokumentacji na temat modelu B;
- Jakub Gazewski: ustalenie szczegółów modelu B i jego implementacja, testowanie modelu B, pomoc w sporządzaniu dokumentacji;
- Adrian Goworek: ustalenie szczegółów modelu A i jego implementacja, testowanie modelu A, pomoc w sporządzeniu dokementacji;
- Adam Wodzisławski: implementacja modelu A, sporządzenie dokumentacji na temat modelu A.

4. Eksploracja danych

Pod linkiem [1] w sekcji 'Data' znajdują się dane treningowe do pobrania. Jest to około 3,8 tys. zdjęć, na których można znaleźć ryby, podzielonych na 8 kategorii:

- ALB: Albacore tuna (Tuńczyk biały),

- BET: Bigeye tuna (Tuńczyk wielkooki),
- DOL: Dolphish/Mahi Mahi (Koryfena),
- LAN: Opah/Moonfish (Lampris),
- SHARK: różne gatunki rekinów,
- YFT: Yellowfin tuna (Tuńczyk żółtopłetwy),
- NoF: brak ryby,
- OTHER: inny gatuenk niż wyżej wymienione.

Zdjęcia są zrobione na różnych łodziach, pod różnymi kątami, a ryby znajdują na nich w różnych miejscach, czasami są częściowo zasłonięte. Niektóre zdjęcia mają nienaturalne kolory, np. są zbyt zielone, a inne są niewyraźne (np. z powodu kropli wody na obiektywie kamery). Podejrzewamy, że zabiegi takie jak odszumianie albo balans bieli mogłyby zwiększyć skuteczność działania naszych modeli.

5. Prace implementacyjne (stan na 21.04.2022)

W ramach prac implementacyjnych stworzyliśmy jak dotąd cztery różne modele sieci neuronowych, które są umieszone w repozytorium projektu [2]. Przygotowaliśmy następujące modele:

- Model korzystający z transfer learningu do użycia wytrenowanej sieci VGG16. Maksymalna skuteczność: około 88% (korzystający z biblioteki *tensorflow*),
- Model używający sieci Resnet50 (korzystający z biblioteki *pytorch*),
- Model korzystający z niewytrenowanej sieci VGG16 (korzystający z biblioteki *pytorch*),
- Wlasny model sieci CNN, składający się z dwudziestu warstw (korzystający z biblioteki *tensorflow*).

Korzystając z tego, że niektóre modele zostały utworzone z wykorzystaniem wytrenowanych sieci neuronowych, a inne z niewytrenowanych, będziemy mogli porównać skuteczności, jakie można otrzymać używając każdego z tych podejść. Podjerzewamy, że sprawność sieci można jeszcze zwiększyć lepiej dobierając hiperparametry oraz za pomoca lepszego przystosowania danych wejściowych.

6. Prace implementacyjne (stan na 26.05.2022)

Porównując metrykę *accuracy* powyższych modeli doszliśmy do wniosku, że najlepiej w naszym problemie sprawują się modele:

- korzystający z transfer learningu do użycia wytrenowanej sieci VGG16,
- własny model sieci CNN, składający się z dwudziestu warstw.

To na nich skupiliśmy się w daleszej części sprawozdania. Model pierwszy działał ze skutecznością około 88% (skuteczność z walidacji), drugi około 75%.

W ramach zwiększenia skuteczności modeli, zastosowaliśmy poprawienie zdjęć poprzez wyrównanie kolorów, co poskutkowało podwyższeniem widoczności sylwetek ryb na tle statku oraz ogólnym pojaśnieniem obrazu. W tym celu wykorzystaliśmy funkcję *automatic_ color_ equalization* z biblioteki *colorcorrect* dla języka Python. Porównaliśmy działanie wcześniej wytrenowanych modeli, z modelami wytrenowanymi przy użyciu tych samych sieci, lecz poprawionych zdjęć. Skuteczność modeli wzrosła wtedy o kilka punktów procentowych, do poziomu blisko 94% w przypadku modelu pierwszego oraz 83% w drugim modelu.

Rysunek 1: Zdjęcie przedstawiające tuńczyka białego przed obróbką.

Rysunek 2: To samo zdjęcie po obróbce.

7. Efekty końcowe

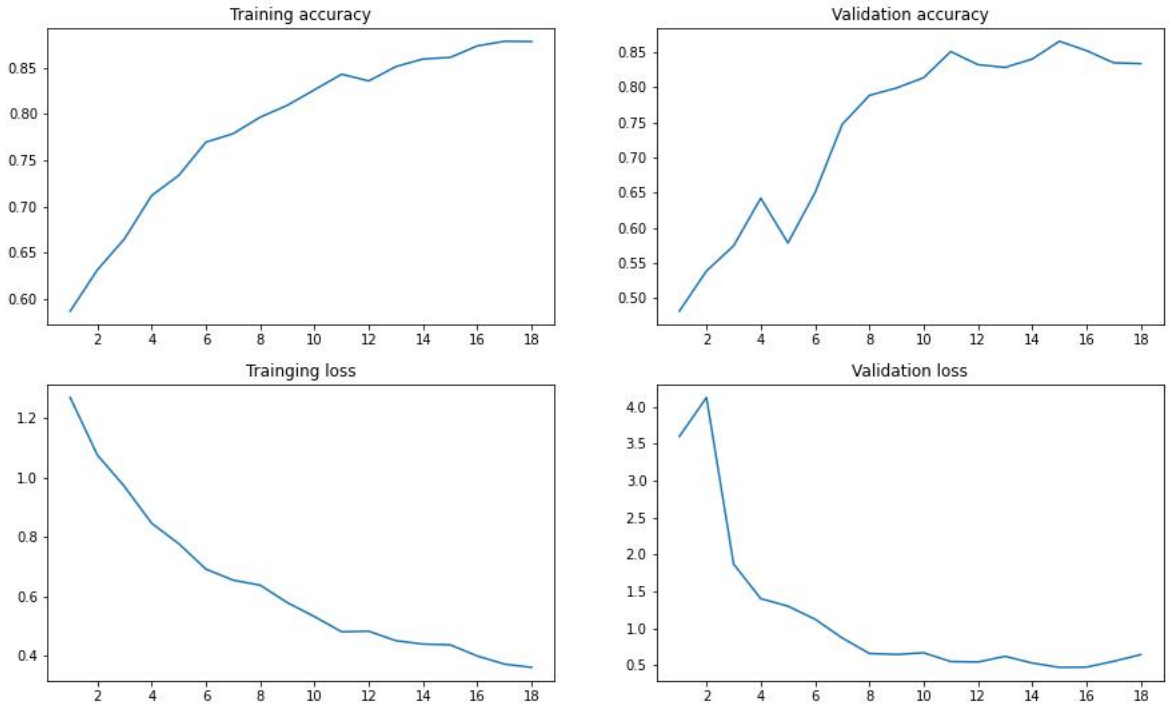
Obie wyżej wymienione sieci były trenowane wielokrotnie korzystając z różnych hiperparametrów. Metodą prób i błędów wyznaczyliśmy miedzy innymi: optymalną liczbę neuronów w warstwach wewnętrznych, rozmiar pakietów (ang. *batch*), czy liczbę epok treningowych modeli.
Podczas dobierania i poprawiania tych parametrów główną metryką na którą zwracaliśmy uwagę była metryka *accuracy* zbioru walidacyjnego, która informowała nas ile procent wszystkich zdjęć ze zbioru walidacyjnego została poprawnie dopasowana do klasy ryby. Jak się dalej okaże, niekoniecznie był to najlepszy sposób decydowania który model działa najlepiej.
Oba modele na początku działania tworzą 2 zbiory: zbiór treningowy i walidacyjny. Rozmiar zbioru walidacyjnego ustaliliśmy na 20% wszystkich udostępnionych nam zdjęć, pozostałe trafiły do zbioru treningowego. Zdjęcia są następnie skalowane do rozmiaru 224 na 224 pikseli, zachowując przy tym informację o kolorze. Pierwszą warstwą naszych sieci będzie więc wejście składające się z $224 \cdot 224 \cdot 3 = 150528$ neuronów. Dalsze etapy trenowania sieci nieznacznie się różnią:

- Dla sieci pretrenowanej VGG16:
  - pobieramy pretrenowaną sieć VGG16, wytrenowaną na bazie zbioru *imagenet* do rozpoznawania części zwierząt,
  - dodajemy własne warstwy: *average pooling* z 512 neuronami, *dense* z 512 neuronami i ostatnią również *dense* wyjściową, a więc z 8 neuronami (odpowiadającymi poszczególnym klasom), wykorzystującą funkcję *softmax*
  - trenujemy tak skonstruowaną sieć, blokując trenowanie pretrenowanej części sieci, trenujemy do momentu aż nie wykryjemy przetrenowania,
  - ponownie trenujemy sieć, tym razem umożliwiając trenowanie pretrenowanej części sieci, zmniejszamy *learning rate*, trenujemy aż do przetrenowania;
- Dla własnej sieci CNN:
  - dodajemy do warstwy wejściowej warstwy konwolucyjne (*Conv2D*), korzystamy przy tym z normalizacji zbiorów (*BatchNormalization*), max poolingu (*MaxPool2D*), dropoutu i zamiany macierzy

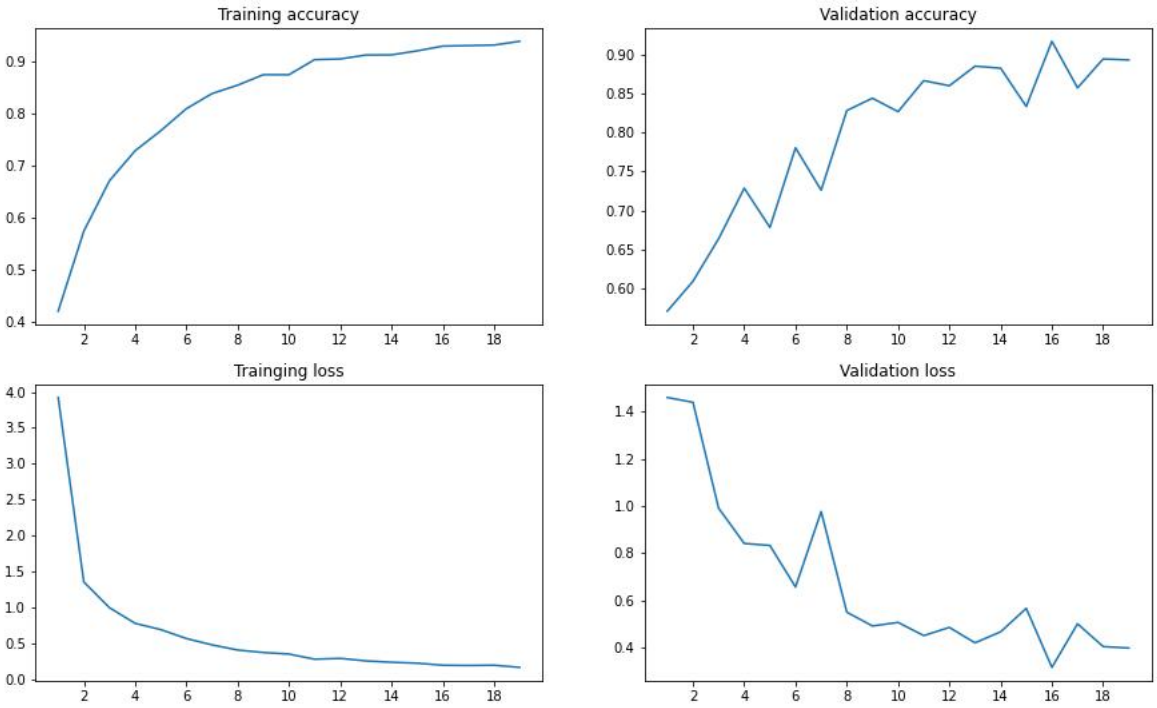
na wektory, na koniec dodajemy tak jak poprzednio warstwę *dense* z 8 neuronami, korzystającymi z funkcji *softmax*,

– trenujemy sieć aż do momentu przetrenowania.

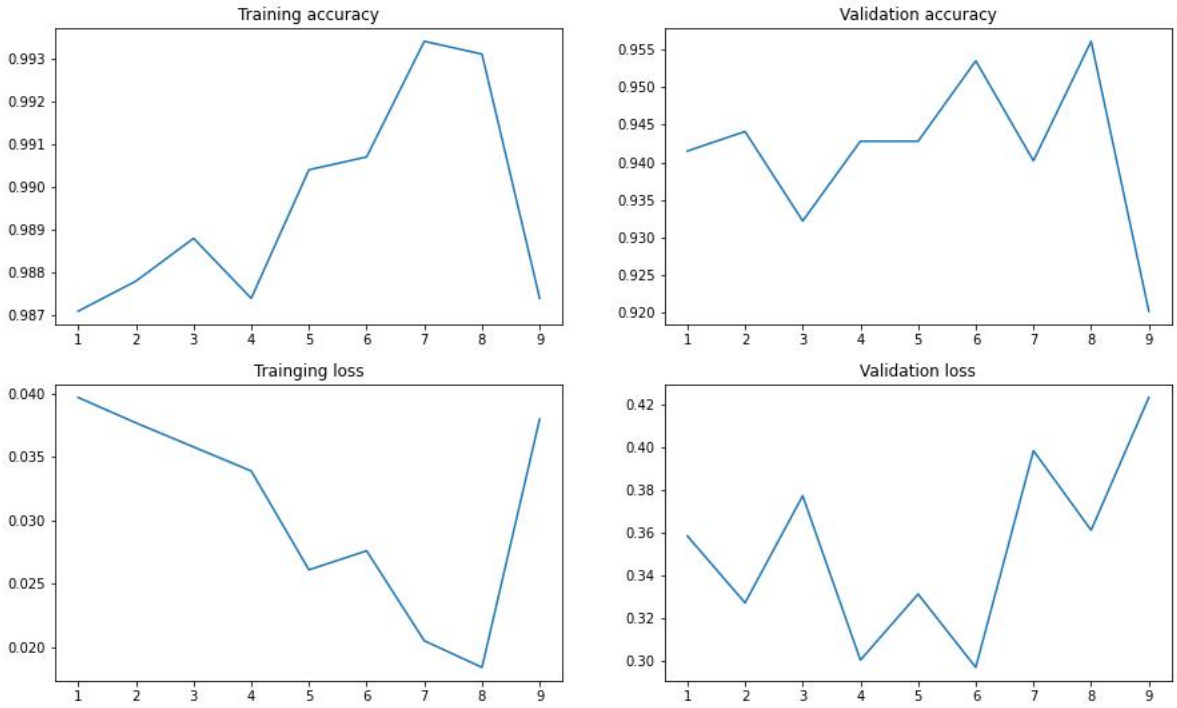
Tak powstałe sieci zapisujemy w modelu, który potem możemy bezpiecznie testować bez możliwości utraty odnalezionych współczynników sieci.



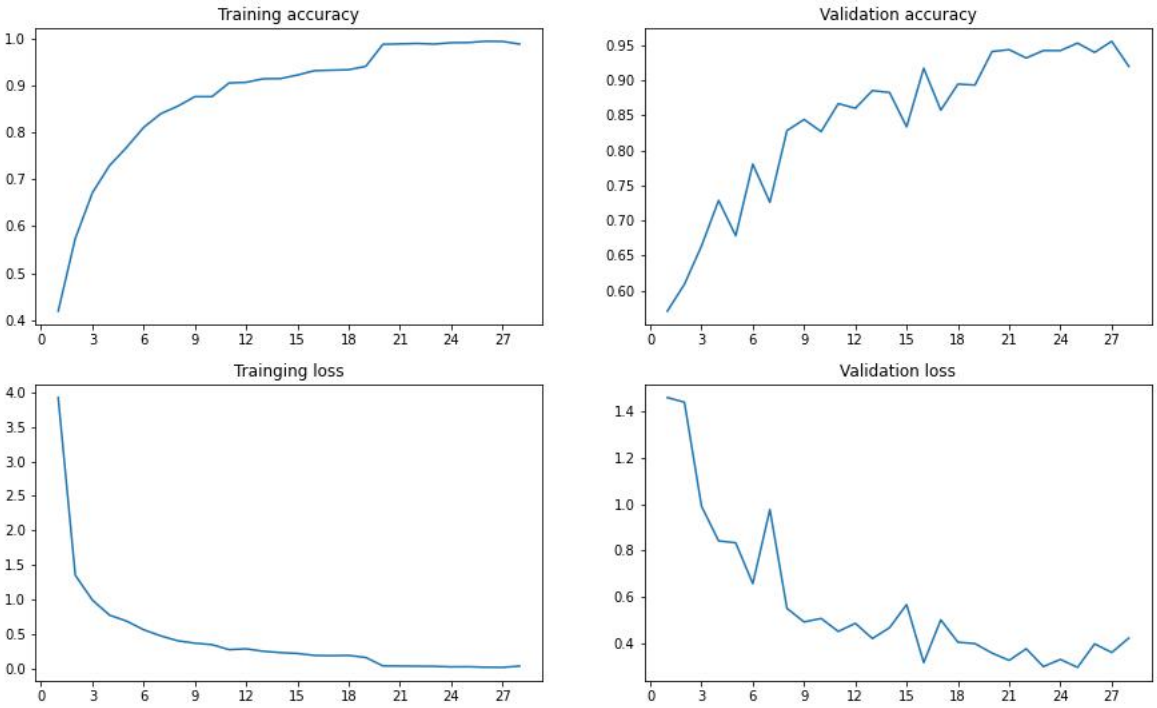
Rysunek 3: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci CNN.



Rysunek 4: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci pretrenowanej VGG16 (pierwszy trening).



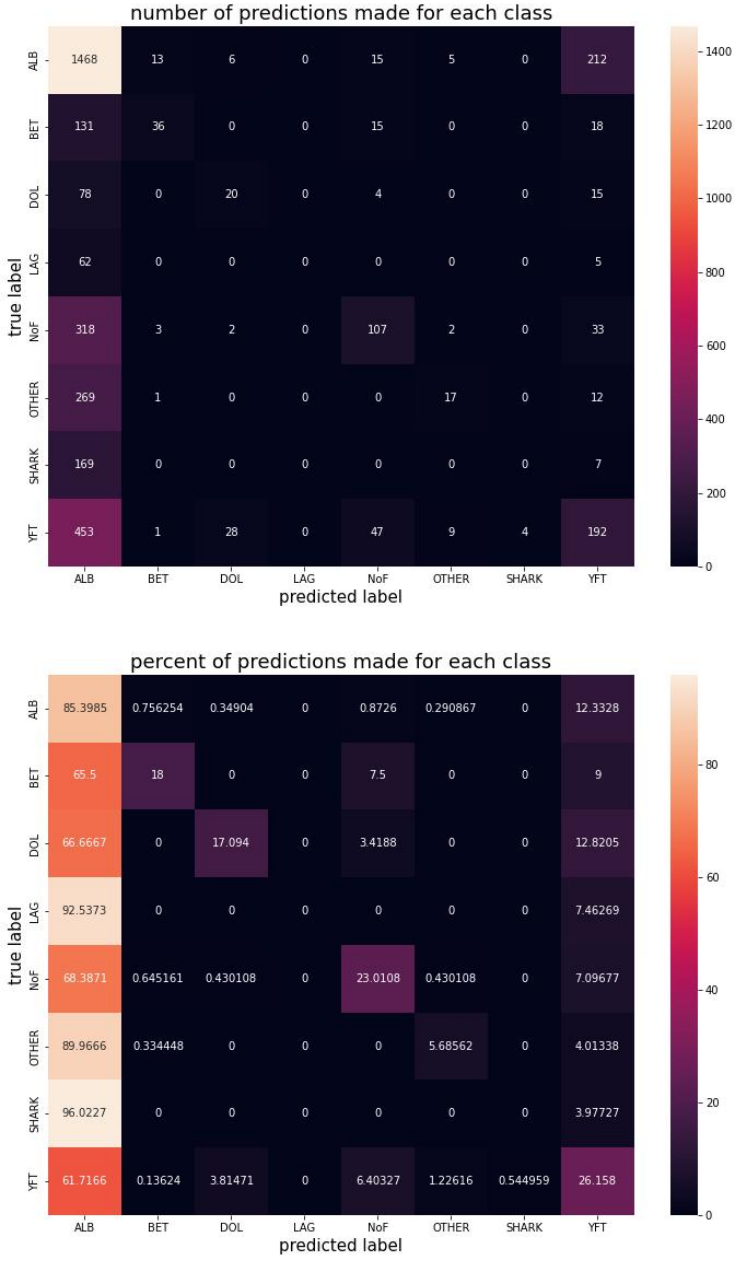
Rysunek 5: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci pretrenowanej VGG16 (drugi trening).



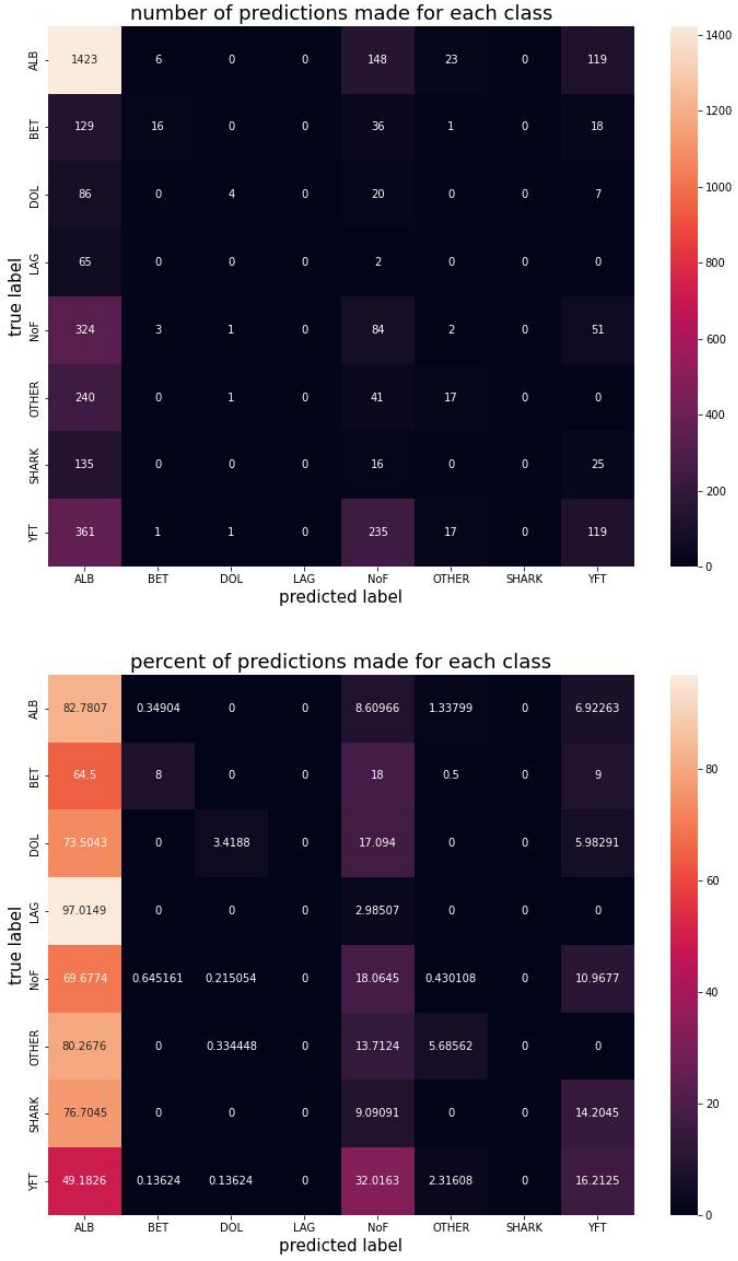
Rysunek 6: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci pretrenowanej VGG16 (cały trening).

Kolejnym etapem projektu było przetestowanie naszych modeli. Dysponowaliśmy jedynie 3777 zdjęciami podzielonymi na kategorie, w treningu i walidacji użyliśmy wszystkich. Dalsze testy musieliśmy więc przeprowadzić na tych samych zdjęciach. Przeprowadziliśmy testy zarówno dla zdjęć oryginalnych, jak i tych po obróbce (na których były trenowane modele). Jako metodę wizualizacji wyników wybraliśmy macierz konfuzji przedstawioną jako *heatmap*. Macierz konfuzji zapisuje informację ile zdjęć i z jakiej klasy zostało przyporządkowanych przez model do każdej z 8 klas. *Heatmapa* oferuje przedstawienie danych zapisanych w macierzy w sposób przystępny wizualnie. Jako iż każda klasa ryb zawierała różne liczby zdjęć, stworzyliśmy

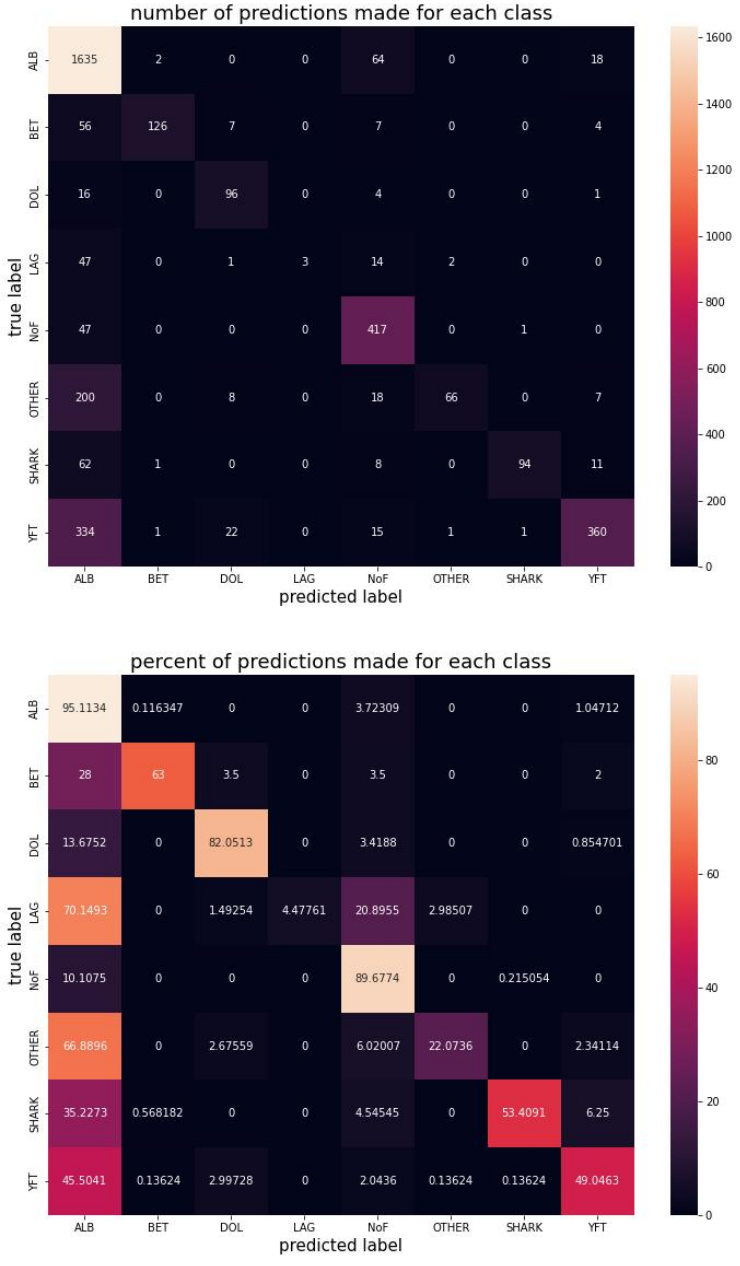
heatmapy przedstawiające liczbę zdjęć przyporządkowanych do każdej klasy, jak i procent zdjęć z danej klasy przyporządkowany do wszystkich 8 klas.



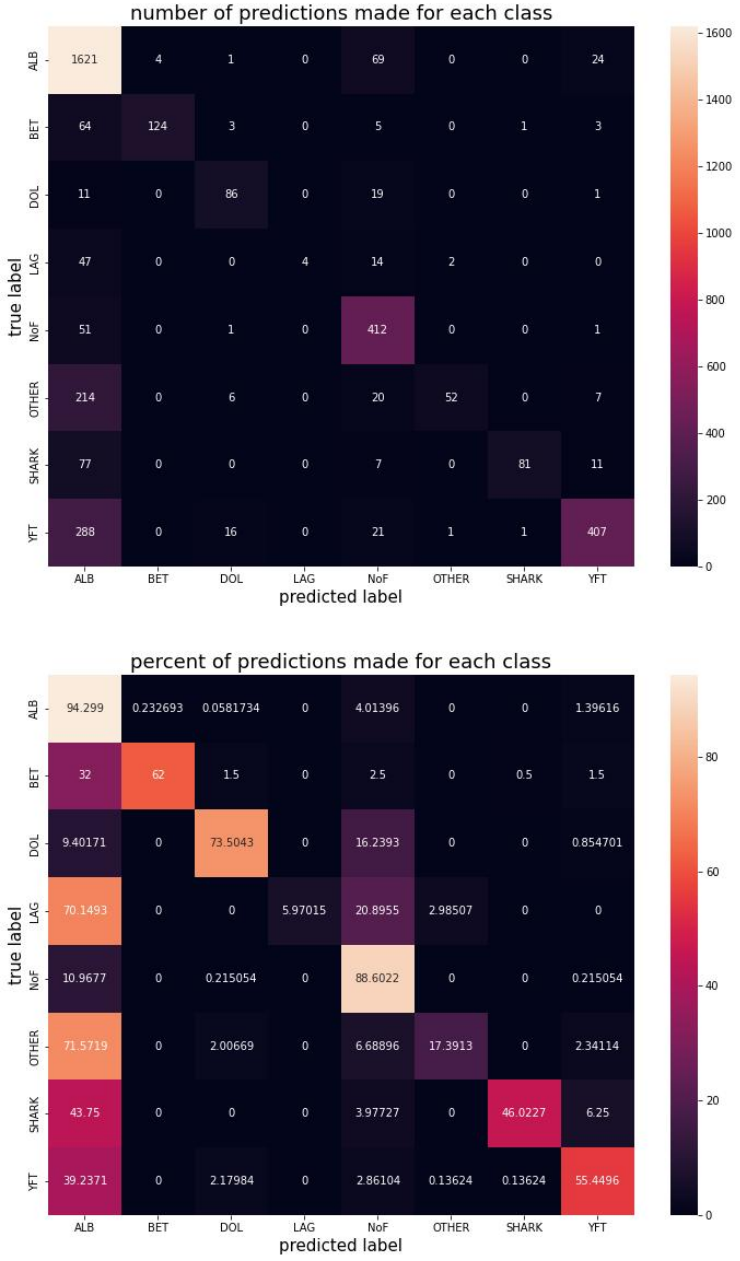
Rysunek 7: *Heatmapy* dla sieci CNN dla zdjęć obrobionych.



Rysunek 8: *Heatmapy* dla sieci CNN dla zdjęć oryginalnych.



Rysunek 9: *Heatmapy* dla sieci pretrenowanej VGG16 dla zdjęć obrobionych.



Rysunek 10: *Heatmapy* dla sieci pretrenowanej VGG16 dla zdjęć oryginalnych.

Jeśli model działa prawidłowo, na głównej diagonali macierzy konfuzji znajdują się większe liczby niż poza nią. Widoczne jest, że w przybliżeniu dla każdego modelu taka zależność zachodzi, w modelach sieci CNN słabiej, w modelu pretrenowanym VGG16 zależność tę widać wyraźniej. Na powyższych *heatmapach* możemy zaobserwować, że:

- klasy które posiadają więcej zdjęć są częściej przyporządkowane zdjęciom, widać to szczególnie dla ALB, ale też dla NoF i YFT, co zgadza się z romiarami tych klas (ALB posiada 1719 zdjęć, NoF 465, a YFT 734),
- klasy które mają bardzo mało zdjęć niemal (lub w ogóle!) nie są przewidywane jako klasa zdjęć podawanych do modelu.

Wyciągnąć można więc następujące wnioski: nierówna liczba zdjęć w poszczególnych klasach przy treningu nie wpływa prawidło na działanie modelu oraz metryka *accuracy* nie jest najlepszym wyznacznikiem poprawności działania sieci (nasze sieci mają duże *accuracy*, bo poprawnie klasyfikują dużą liczbę zdjęć ALB, ale nasze modele słabo rozpoznają inne ryby, np. LAG).

Odnaleźliśmy informację o innych dostępnych metrykach. Są to metryki *precision*, *recall* i *f1-score*, które niesie informację o dwóch poprzednich metrykach.

Metryka *precision* mówi nam jaka część zdjęć przyporządkowanych do klasy przez model została przyporządkowana poprawnie. Im większe *precision* tym mniej model będzie błędnie oznaczał zdjęcia z innych klas do danej klasy, będzie mniej tak zwanych fałszywych pozytywów. Niekoniecznie oznacza to jednak, że model będzie przydzielał poprawnie więcej zdjęć do danej klasy, może przydzielać nawet mniej zdjęć poprawnie, jeśli ma to oznaczać zmniejszenie liczby zdjęć niepoprawnie przydzielonych do danej klasy. Metrykę tę wyprowadza się dla każdej klasy.

Drugą metryką jest metryka *recall*, która niesie informację, jaka część wszystkich zdjęć należących do danej klasy została przyporządkowana przez model poprawnie. Im większe *recall* tym dokładniej model dopasowuje zdjęcia należące do danej klasy do tej właśnie klasy. Nie oznacza to jednak, że model nie będzie przydzielał zdjęć z innych klas do tej klasy, metryka ta takiej sytuacji po prostu nie mierzy.

Ostatnia metryka zapisuje informację o obu wcześniejszych metrykach.

Widzimy więc, że dopiero te informacje dokładniej opiszą naszą sieć. Biblioteka *sklearn* pozwala nam uzyskać takie informacje na podstawie przewidywań modelu i poprawnych przypisań do klas.

	precision	recall	f1-score	support
ALB	0.498	0.854	0.629	1719
BET	0.667	0.180	0.283	200
DOL	0.357	0.171	0.231	117
LAG	0.000	0.000	0.000	67
NoF	0.569	0.230	0.328	465
OTHER	0.515	0.057	0.102	299
SHARK	0.000	0.000	0.000	176
YFT	0.389	0.262	0.313	734
accuracy			0.487	3777
macro avg	0.374	0.219	0.236	3777
weighted avg	0.459	0.487	0.418	3777

Rysunek 11: Metryki zwrócone przez bibliotekę *sklearn* dla sieci CNN dla zdjęć obrobionych.

	precision	recall	f1-score	support
ALB	0.515	0.828	0.635	1719
BET	0.615	0.080	0.142	200
DOL	0.571	0.034	0.065	117
LAG	0.000	0.000	0.000	67
NoF	0.144	0.181	0.160	465
OTHER	0.283	0.057	0.095	299
SHARK	0.000	0.000	0.000	176
YFT	0.351	0.162	0.222	734
accuracy			0.440	3777
macro avg	0.310	0.168	0.165	3777
weighted avg	0.393	0.440	0.369	3777

Rysunek 12: Metryki zwrócone przez bibliotekę *sklearn* dla sieci CNN dla zdjęć oryginalnych.

	precision	recall	f1-score	support
ALB	0.682	0.951	0.794	1719
BET	0.969	0.630	0.764	200
DOL	0.716	0.821	0.765	117
LAG	1.000	0.045	0.086	67
NoF	0.762	0.897	0.824	465
OTHER	0.957	0.221	0.359	299
SHARK	0.979	0.534	0.691	176
YFT	0.898	0.490	0.634	734
accuracy			0.741	3777
macro avg	0.870	0.574	0.615	3777
weighted avg	0.791	0.741	0.713	3777

Rysunek 13: Metryki zwrócone przez bibliotekę *sklearn* dla sieci pretrenowanej VGG16 dla zdjęć obrobionych.

	precision	recall	f1-score	support
ALB	0.683	0.943	0.792	1719
BET	0.969	0.620	0.756	200
DOL	0.761	0.735	0.748	117
LAG	1.000	0.060	0.113	67
NoF	0.727	0.886	0.798	465
OTHER	0.945	0.174	0.294	299
SHARK	0.976	0.460	0.625	176
YFT	0.896	0.554	0.685	734
accuracy			0.738	3777
macro avg	0.870	0.554	0.601	3777
weighted avg	0.788	0.738	0.710	3777

Rysunek 14: Metryki zwrócone przez bibliotekę *sklearn* dla sieci pretrenowanej VGG16 dla zdjęć oryginalnych.

Z podanych metryk ponownie wyciągmy informację, że sieć pretrenowana VGG16 dla zdjęć obrobionych daje najlepsze wyniki oraz, że klasy które mają mniej zdjęć osiagają słabsze wyniki. Analizując dokładniej *precision* i *recall* widzimy, że klasy które mają więcej zdjęć osiągają niskie wyniki w metryce

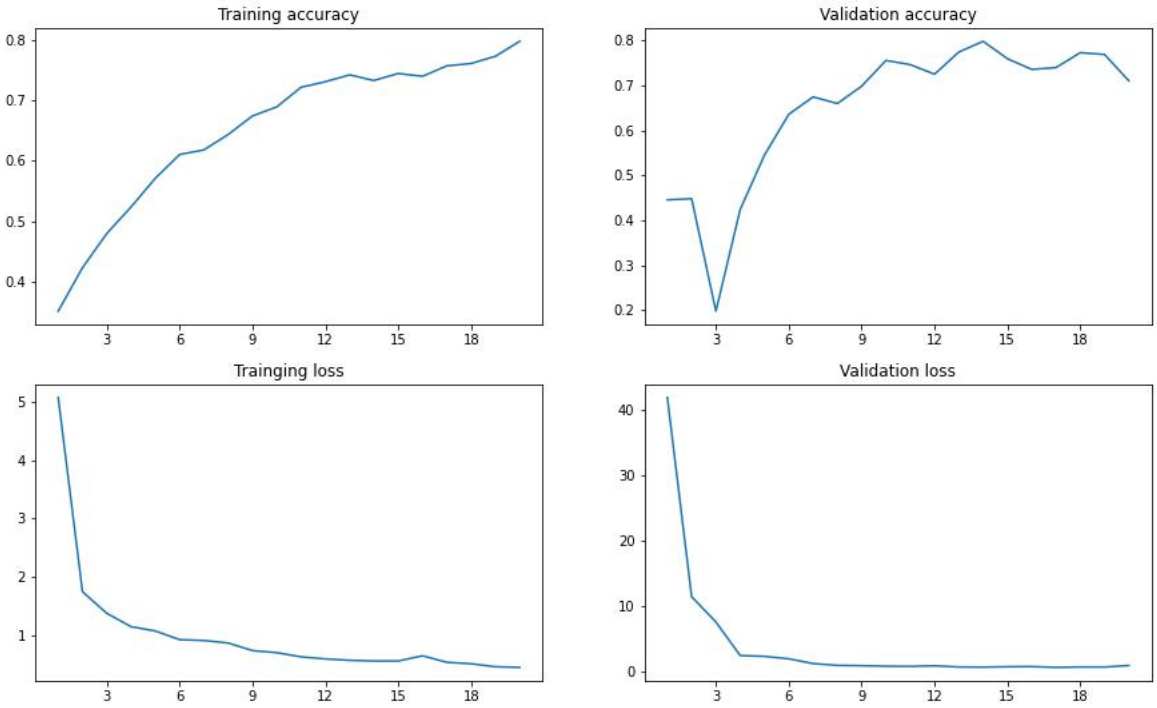
precision, co oznacza, że do tych klas częściej są przydzielane zdjęcia należące do innych klas. Sieci nauczyły się rozpoznawać klasy których było najwięcej, doszukują się ich nawet tam, gdzie ich nie ma. Prowadzi to po raz kolejny do wniosku że należy doprowadzić do równych rozmiarów klas.

Problem który doświadczamy jest problemem niezrównoważenia zbiorów, można go rozwiązać na kilka sposobów:

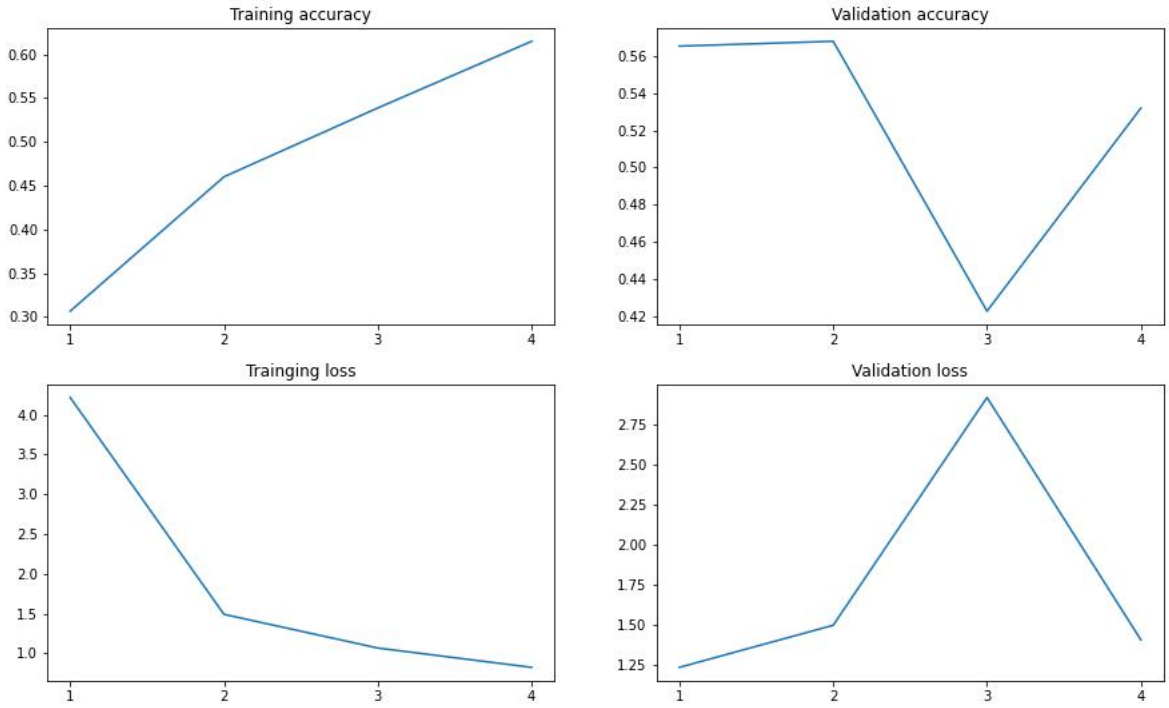
- poprzez *undersampling* - zmniejszenie najliczniejszych klas, istnieje jednak zagrożenie ominięcia istotnych danych o tych klasach,
- poprzez *oversampling* - zwiększenie najmniej liczbych klas poprzez powielanie zdjęć lub powielanie i stosowanie niewielkich zmian w zdjęciach, które nie zmieniają klasy zdjęcia. Niesie ze sobą ryzyko nadmiernego dopasowania tych klas, które nauczą się rozpoznwania konkretnie tych zdjęć dla tych klas, a nie rozwiązywania problemu,
- poprzez kombinację *undersamplingu* i *oversamplingu*,
- poprzez zastosowanie wag dla klas (im liczniejsza klasa tym mniejsza waga zmian jakie niesie ze sobą ta klasa).

Napotykamy tutaj jednak pewny problem implementacyjny. Podczas implementacji sieci staraliśmy się użyć *oversamplingu* poprzez obroty, transformacje i odbicia w pionie zdjęć na podstawie których potem wytrenowaliśmy nasze sieci. Jak wskazują dane, prawdopodobnie nie udało nam się poprawie go zastosować. Wynikać to może chociażby z małego doświadczenia jakie posiadamy przy pracy z bibliotekami do uczenia maszynowego.

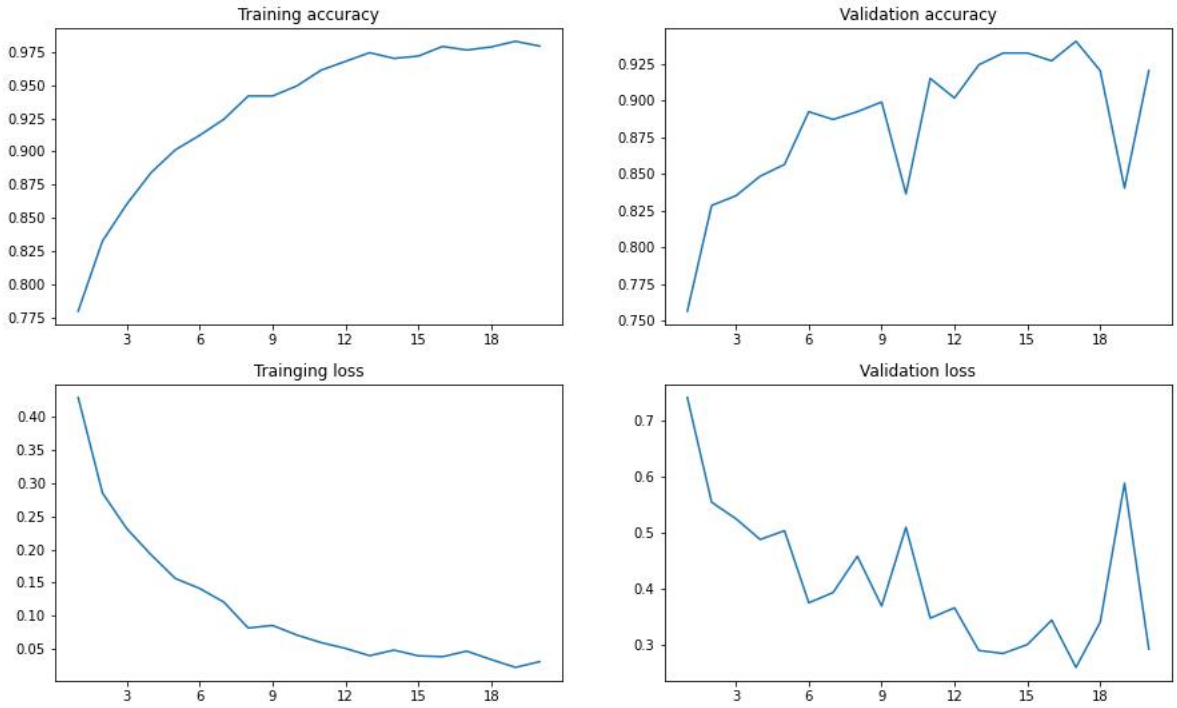
W tym przypadku spróbujemy więc rozwiązać problem niezrównoważenia zbiorów przy pomocy wag. Przeprowadzamy proces treningu sieci na nowo.



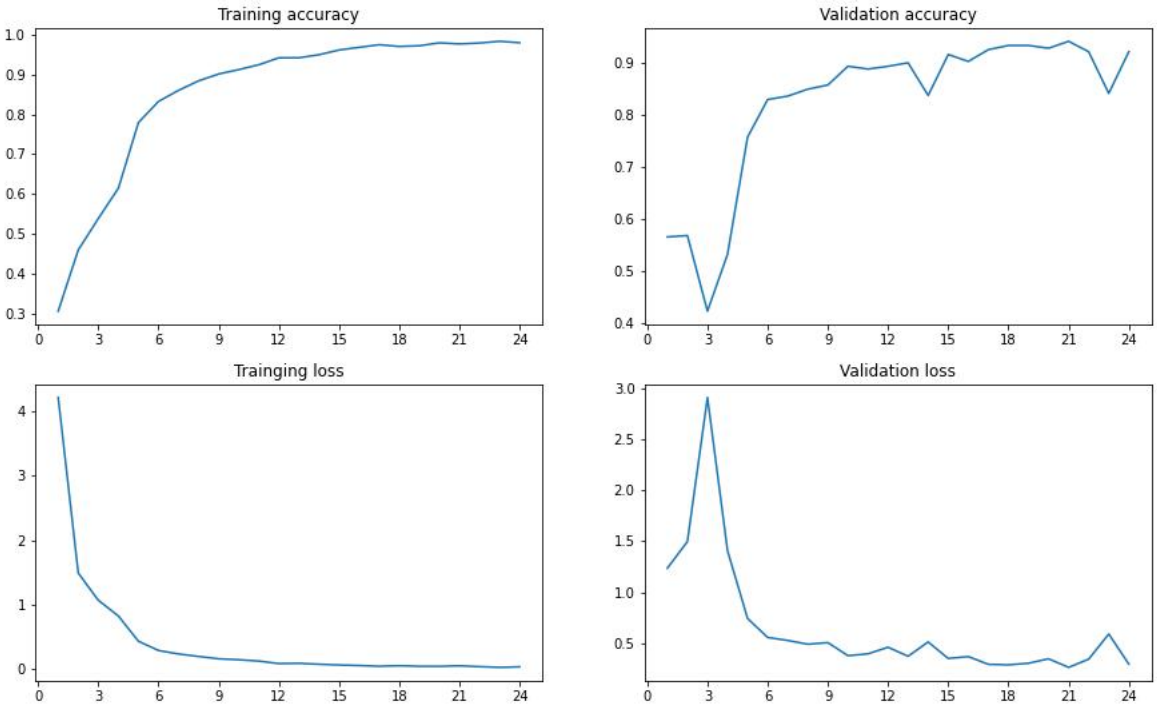
Rysunek 15: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci CNN po zastosowaniu wag.



Rysunek 16: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci pretrenowanej VGG16 po zastosowaniu wag (pierwszy trening).



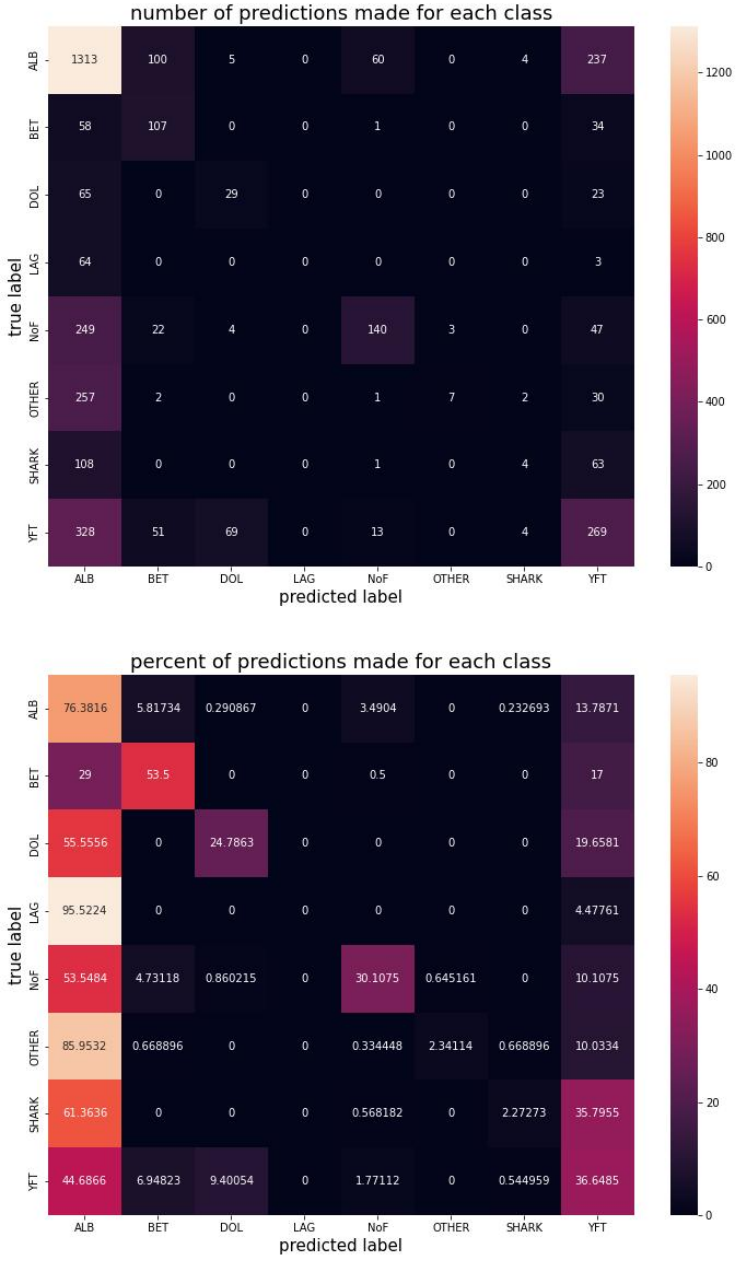
Rysunek 17: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci pretrenowanej VGG16 po zastosowaniu wag (drugi trening).



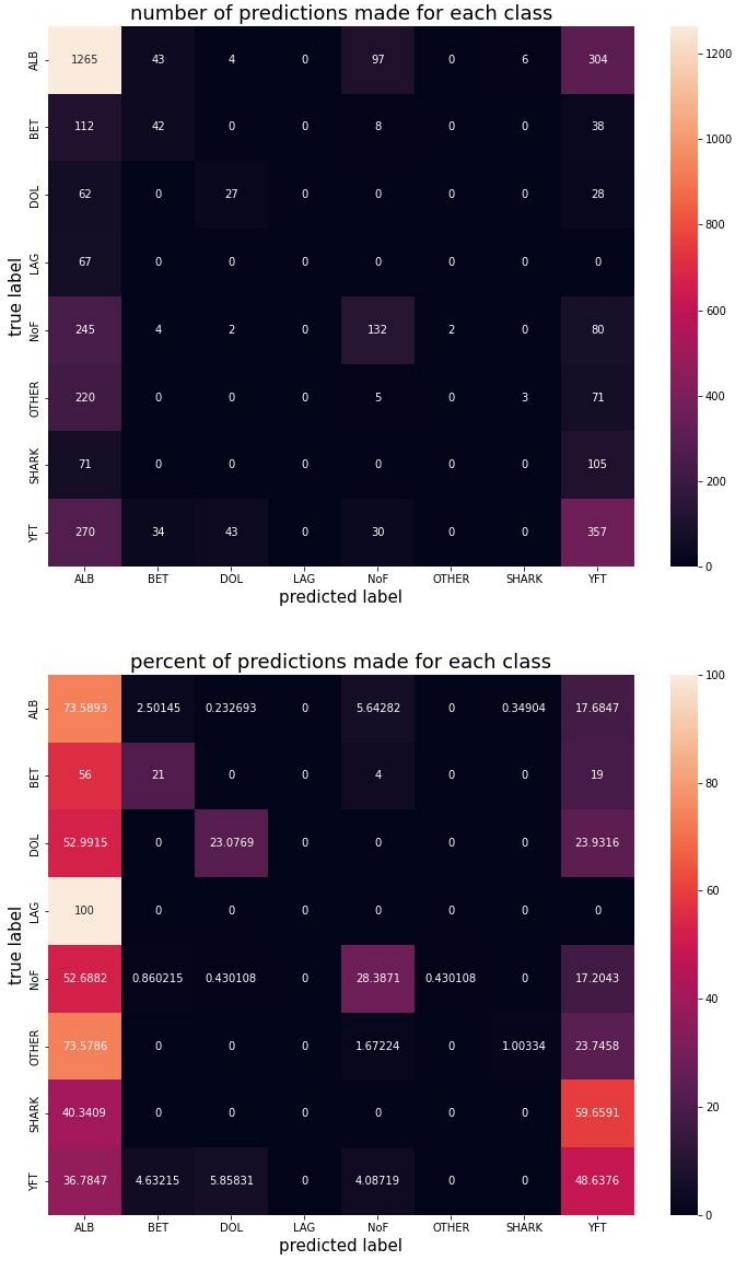
Rysunek 18: Wykresy przedstawiające wartości metryki skuteczności i wartość *loss* dla kolejnych epok zwrócone przez bibliotekę *tensorflow* dla zbioru treningowego i walidacyjnego dla sieci pretrenowanej VGG16 po zastosowaniu wag (cały trening).

Widzimy, że w obu sieciach wartość *accuracy* dla zbiorów walidacyjnych spadła o kilka punktów procentowych. Jednak jak wcześniej pokazaliśmy, metryka ta nie niesie pełnej informacji o działaniu sieci.

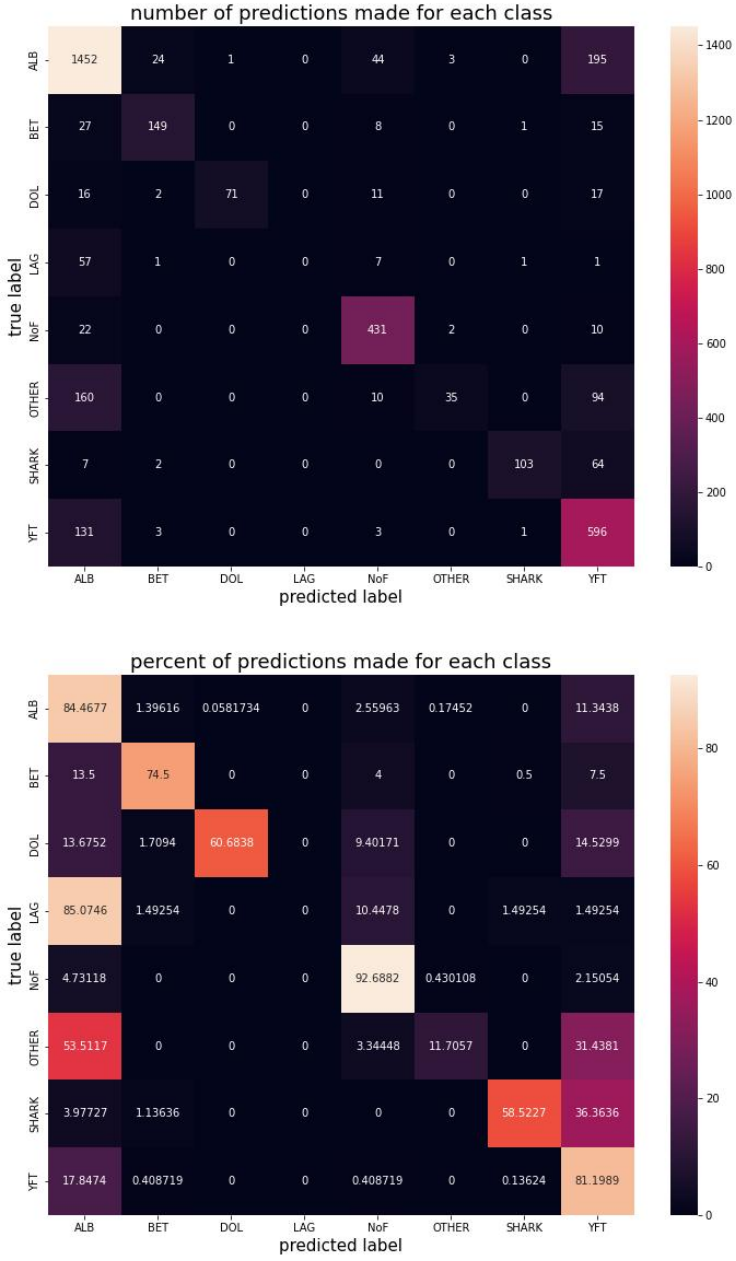
Wyznaczamy macierze konfuzji dla każdej sieci dla zbiorów zdjęć obrobionych i oryginalnych.



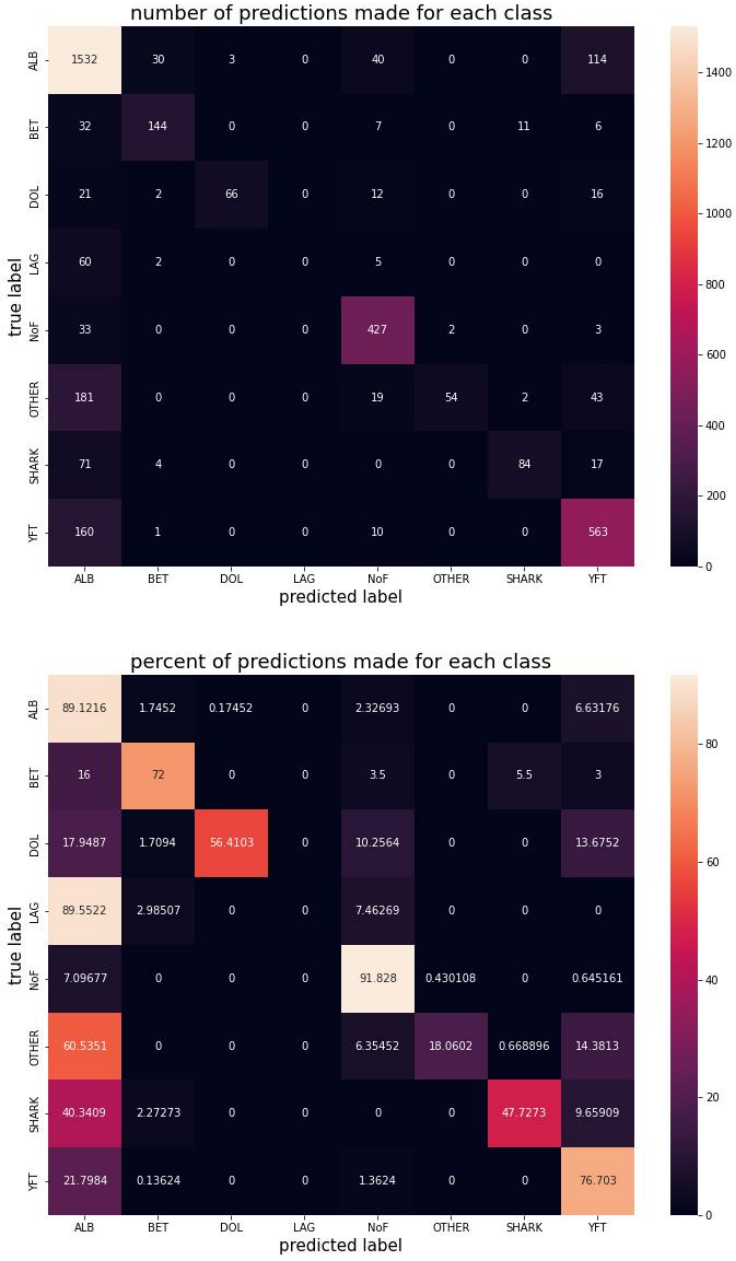
Rysunek 19: *Heatmapy* dla sieci CNN dla zdjęć obrobionych po zastosowaniu wag.



Rysunek 20: *Heatmapy* dla sieci CNN dla zdjęć oryginalnych po zastosowaniu wag.



Rysunek 21: *Heatmapy* dla sieci pretrenowanej VGG16 dla zdjęć obrobionych po zastosowaniu wag.



Rysunek 22: *Heatmapy* dla sieci pretrenowanej VGG16 dla zdjęć oryginalnych po zastosowaniu wag.

Na podstawie przedstawionych *heatmap* możemy przypuszczać, że zastosowanie wag w pewien sposób poprawiło nasze sieci, mimo zmniejszenia *accuracy* zbioru walidacyjnego. Nowe modele poprawniej klasyfikują zdjęcia pochodzące z mniej licznych klas.

	precision	recall	f1-score	support
ALB	0.538	0.764	0.631	1719
BET	0.379	0.535	0.444	200
DOL	0.271	0.248	0.259	117
LAG	0.000	0.000	0.000	67
NoF	0.648	0.301	0.411	465
OTHER	0.700	0.023	0.045	299
SHARK	0.286	0.023	0.042	176
YFT	0.381	0.366	0.374	734
accuracy			0.495	3777
macro avg	0.400	0.283	0.276	3777
weighted avg	0.496	0.495	0.448	3777

Rysunek 23: Metryki zwrócone przez bibliotekę *sklearn* dla sieci CNN dla zdjęć obrobionych po zastosowaniu wag.

	precision	recall	f1-score	support
ALB	0.547	0.736	0.628	1719
BET	0.341	0.210	0.260	200
DOL	0.355	0.231	0.280	117
LAG	0.000	0.000	0.000	67
NoF	0.485	0.284	0.358	465
OTHER	0.000	0.000	0.000	299
SHARK	0.000	0.000	0.000	176
YFT	0.363	0.486	0.416	734
accuracy			0.483	3777
macro avg	0.262	0.243	0.243	3777
weighted avg	0.408	0.483	0.433	3777

Rysunek 24: Metryki zwrócone przez bibliotekę *sklearn* dla sieci CNN dla zdjęć oryginalnych po zastosowaniu wag.

	precision	recall	f1-score	support
ALB	0.776	0.845	0.809	1719
BET	0.823	0.745	0.782	200
DOL	0.986	0.607	0.751	117
LAG	0.000	0.000	0.000	67
NoF	0.839	0.927	0.880	465
OTHER	0.875	0.117	0.206	299
SHARK	0.972	0.585	0.730	176
YFT	0.601	0.812	0.691	734
accuracy			0.751	3777
macro avg	0.734	0.580	0.606	3777
weighted avg	0.762	0.751	0.726	3777

Rysunek 25: Metryki zwrócone przez bibliotekę *sklearn* dla sieci pretrenowanej VGG16 dla zdjęć obrobionych po zastosowaniu wag.

	precision	recall	f1-score	support
ALB	0.733	0.891	0.804	1719
BET	0.787	0.720	0.752	200
DOL	0.957	0.564	0.710	117
LAG	0.000	0.000	0.000	67
NoF	0.821	0.918	0.867	465
OTHER	0.964	0.181	0.304	299
SHARK	0.866	0.477	0.615	176
YFT	0.739	0.767	0.753	734
accuracy			0.760	3777
macro avg	0.733	0.565	0.601	3777
weighted avg	0.766	0.760	0.734	3777

Rysunek 26: Metryki zwrócone przez bibliotekę *sklearn* dla sieci pretrenowanej VGG16 dla zdjęć oryginalnych po zastosowaniu wag.

Finalnie, dla klas o mniejszej liczbie zdjęć wartości metryk nieznacznie się poprawiły. Oznacza to, że w pewnym stopniu poprawiliśmy działanie naszych sieci poprzez zastosowanie wag. Prawdopodobnie mimo wszystko, prawidłowe użycie *oversamplingu* mogłoby przynieść najlepsze rezultaty, ponieważ

wygenerowałoby nowe dane nie zmniejszając istotności danych które już posiadamy dla liczniejszych klas.

Otrzymujemy więc najlepszą sieć poprzez zastosowanie pretrenowanej sieci VGG16 w naszej sieci, wytrenowanie jej na zdjęciach poddanych wyrównaniu kolorów i używaniu później w modelach zdjęć poddanych wyrównywaniu kolorów.

Literatura

[1] www.kaggle.com/c/the-nature-conservancy-fisheries-monitoring

[2] https://github.com/JakubGazewski/WTUM_7_2022