MRO - zadanie 4: Zalewamy wrzątkiem chińską zupkę, czyli tym razem nasza sieć przygotowana jest z użyciem gotowej bazy

Piotr Zawiślan

15 Listopad 2022

1 Wykorzystane technologie

Kod pisałem w języku Python z wykorzystaniem frameworka Pytorch.

2 Kaczko, kaczko, idź!

Jako zindywidualizowany zbiór danych wybrałem zbiór trzech rodzajów jednostek pływających:

- katamarany (*catamaran*) - jednostki dwu (lub więcej) - kadłubowe,
- jachty motorowe (*motor_yacht*) - jednostki bez masztu i żagli
- jachty żaglowe (*sailboat*) - jednostki jednokadłubowe z masztem i żaglami

Nazwijmy ten dataset 'łódkowym zbiorem danych'. W późniejszym etapie zadania będziemy używać 'dużego łódkowego zbioru danych' - odpowiednika o rozdzielczości 256×256.

ImageNet zawiera klasę *speedboat* czyli motorówka, ale nie ma za to jachtów żaglowych, więc stwierdziłem, że taki wybór klas będzie odpowiedni. Niestety wybrane klasy nieco na siebie nachodzą (np. katamaran może być jednocześnie jachtem motorowym), ale w większości przypadków powinno się dać stwierdzić, która klasa jest najbardziej odpowiednia (wyjątkiem jest katamaran pod pełnymi żaglami uchwycony od boku - bardzo ciężko go wtedy odróżnić od zwykłej żaglówki).

Dataset pobrałem z wykorzystaniem poleconej biblioteki *jmd_imagescraper*. Po ręcznym przeczyszczeniu zdjęć (które jednak zabrało dużo czasu... :)) do każdej z klas należało po 200 zdjęć. Następnie użyłem klasy *torchvision.datasets.ImageFolder* aby załadować zdjęcia z lokalnego folderu do pamięci. Jako transformacje zastosowałem normalizację i przeskalowanie do rozmiaru 32×32. Podzieliłem zbiór danych na treningowy i walidacyjny (w proporcji 5:1).

```
height, width = 32, 32

image_folder = ImageFolder(
    root=str(image_root),
    transform=Lambda(lambda X: resize(img=ToTensor()(X), size=(height, width))),
    target_transform=Lambda(
        lambda y: torch.zeros(3, dtype=torch.float).scatter_(0, torch.tensor(y), value
    =1)
    )
)
```

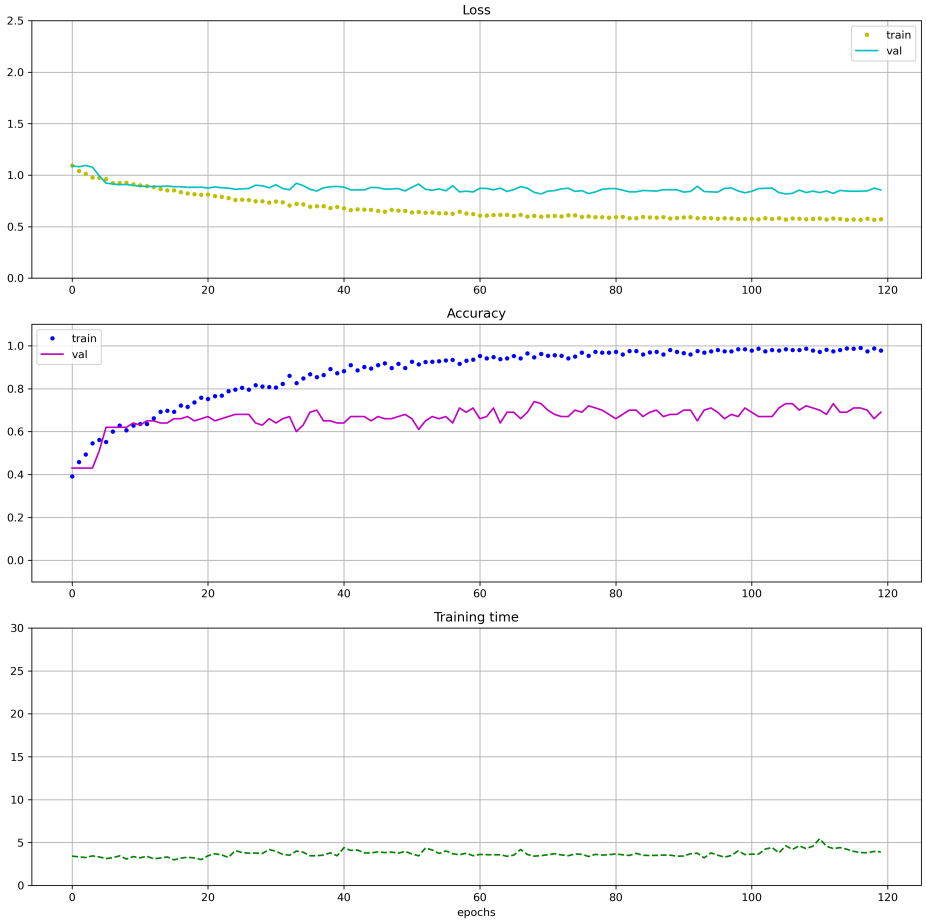
Listing 1: Zainstancjonowanie datasetu z folderu

Rysunek 1: Jacht żaglowy 32×32

3 Sieci konwolucyjne na bazie gotowej kostki rosołowej

3.1 Architektura z zadania 03

Po wytrenowaniu od zera architektury z poprzedniego zadania (4 bloki konwolucyjne + warstwa gęsta) na zbiorze łódkowym udaje się uzyskać takie statystyki:



Rysunek 2: Statystyki treningu modelu na bazie architektury z zadania 03

Wnioskując po rozstępie między dokładnością na zbiorze treningowym a walidacyjnym możemy stwierdzić, że model się przetrenował (ale może, dla takiego małego zbioru danych, jest to OK).

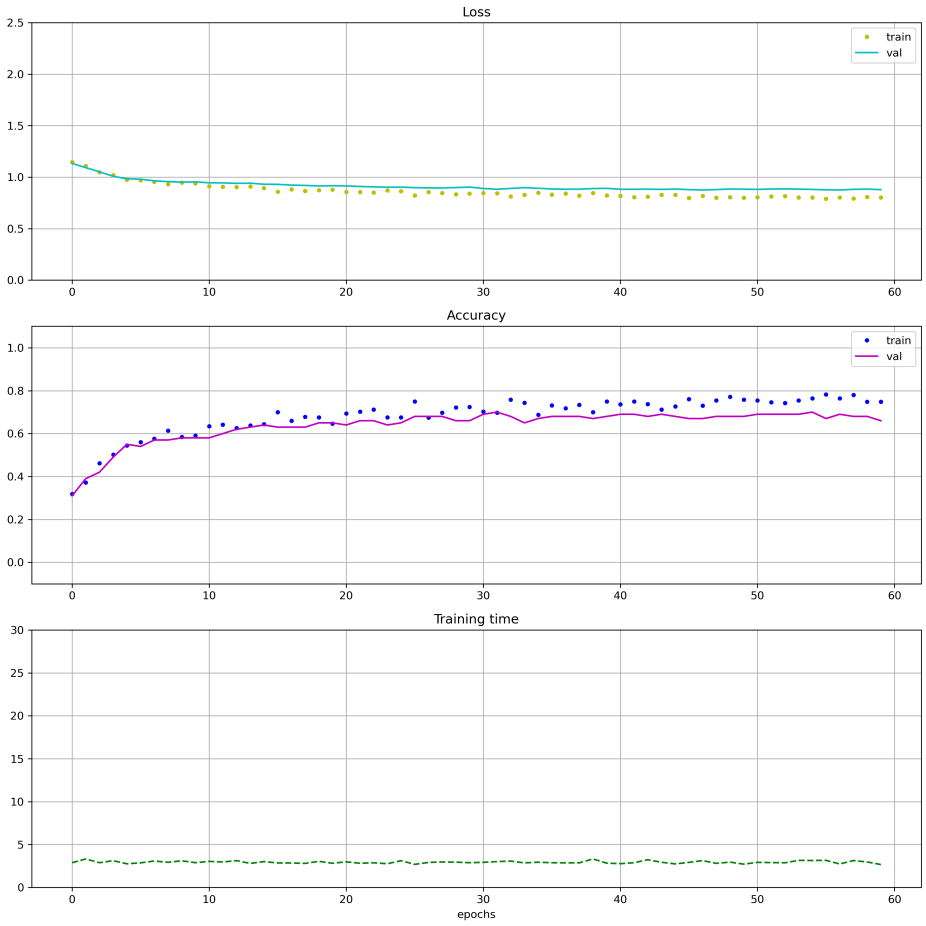
3.2 *Transfer learning* na bazie modelu z zadania 03

Po wczytaniu modelu z zadania 03, wytrenowanym na zbiorze *CIFAR10*, zamroziłem jego wagi z użyciem metody *nn.Module.requires_grad_(False)* oraz ustawiłem warstwy takie jak *Dropout* oraz *BatchNorm2d* w trybie ewaluacji *nn.Module.eval()*.

```
list(model_z3.children())[0][-2] = nn.Linear(in_features=640, out_features=3, bias=
    True)
```

Listing 2: Podmiana warstwy liniowej na nową

Statystyki treningu takiej sieci po 60 epokach:



Rysunek 3: Statystyki treningu modelu z zadania 03 z nową warstwą liniową

Rozstęp pomiędzy dokładnością na zbiorze treningowym a walidacyjnym zanikł, ale dokładność też nie jest wyjątkowo wysoka.

3.3 *Transfer learning* na bazie sieci *Xception*

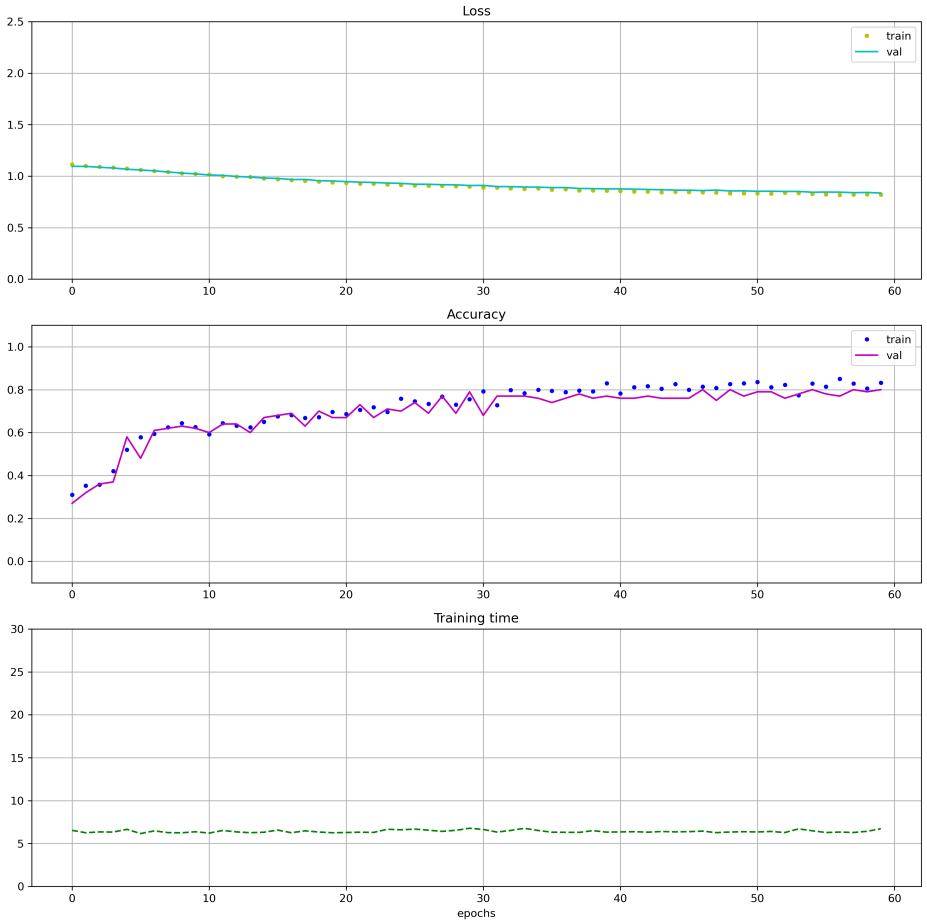
Tym razem zastosujemy *Transfer learning* na bazie modelu *Xception* wytrenowanego na *ImageNet*. Pretrenowany model pobrałem z wykorzystaniem modułu *pretrainedmodels*.

Po zamrożeniu wag, końcową warstwę liniową zastąpiłem warstwami: *GAP*, *Flatten* oraz *Linear* z aktywacją *Softmax*.W celu wytrenowania ostatniej warstwy gęstej, wykorzystamy duży łódkowy zbiór danych.

```
xception_gap = nn.Sequential(
    *list(xception_model.children())[:-1],
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten(),
    nn.Linear(in_features=2048, out_features=3),
    nn.Softmax(dim=1)
)
```

Listing 3: Podmiana warstwy liniowej na inny potok przetwarzania

Statystyki treningu takiej sieci po 60 epokach:



Rysunek 4: Statystyki treningu ostatniej warstwy liniowej modelu xception

Tym razem model osiąga bardzo dobre wyniki mimo braku nadmiernego dopasowania do zbioru treningowego.

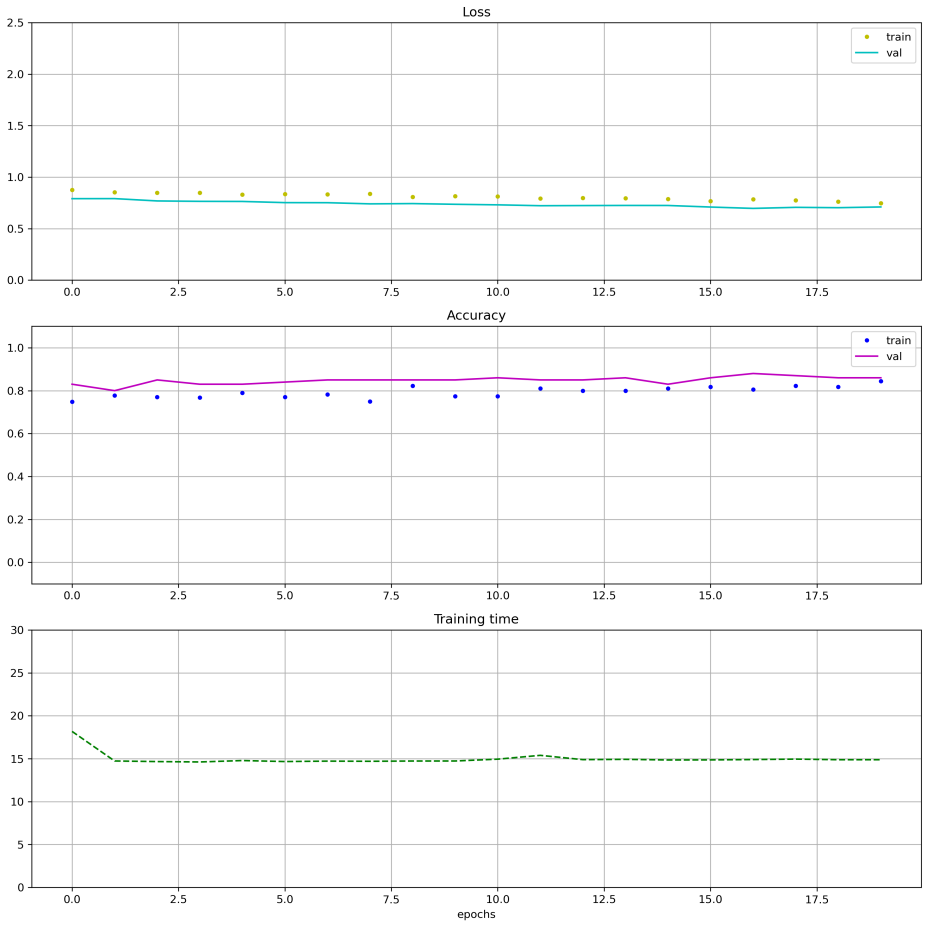
3.4 Fine-tuning sieci Xception

Po rozmrożeniu wag we wszystkich warstwach oraz ustawieniu warstw *BatchNorm2d* w trybie inferencji, rozpocząłem fine-tuning modelu z *learning rate* równym 0.0001 (10× mniej niż przedtem).

```
xception_gap_fine_tuned.eval()
for p in xception_gap_fine_tuned.parameters():
    p.requires_grad = True
```

Listing 4: Ustawienie modelu w trybie inferencji i rozmrożenie wag

Statystyki dotrenowania modelu przez 20 epok:



Rysunek 5: Statystyki dotrenowania modelu xception

Wartość funkcji kosztu stale spadała - więc bardzo dobrze.

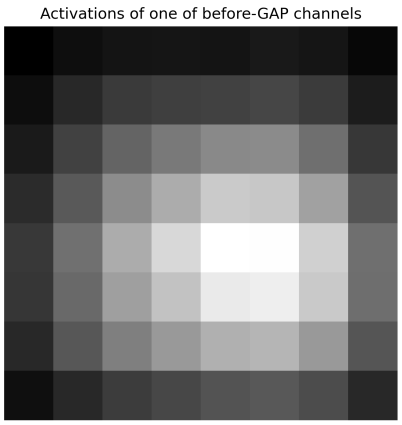
4 No dobra, ale w sumie na jakiej podstawie taka sieć podejmuje decyzje?

4.1 Poprawnie zaklasyfikowane obrazy

Przykładowy obraz (jacht motorowy):

Rysunek 6: Jacht motorowy

Przewidziana klasa: jacht motorowy



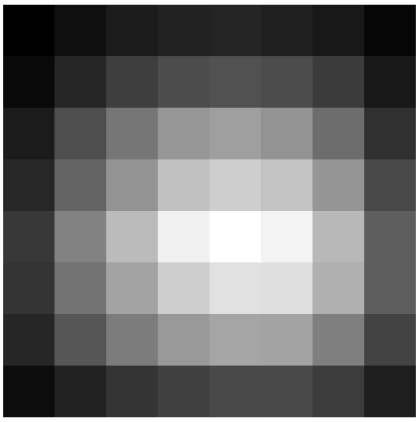
Rysunek 7: Aktywacje jednego z 2048 kanałów przed warstwą *GAP* dla powyższego zdjęcia

```python
def get_influence_map(
    image: tuple[torch.Tensor, torch.Tensor],
    model=xception_fine_tuned
) -> torch.Tensor:

    xception_fine_tuned_before_gap = nn.Sequential(
        *list(model.children())[:-4])

    xception_fine_tuned_before_softmax = nn.Sequential(
        *list(model.children())[:-1])

    X = image[0]
    res_before_gap = xception_fine_tuned_before_gap(X.to('cuda')).to('cpu')
    res_before_softmax = xception_fine_tuned_before_softmax(X.to('cuda')).to('cpu')

    class_int = model(X.to('cuda')).squeeze().argmax().item()
    last_layer_weights = list(list(xception_fine_tuned_before_softmax.children())[-1].
    parameters())[0]
    last_layer_weights_for_class = last_layer_weights[class_int].unsqueeze(1).
    unsqueeze(2)

    return torch.sum(res_before_gap.squeeze() * last_layer_weights_for_class.to('cpu')
    , dim=0).to('cpu')
```

Listing 5: Funkcja zwracająca mapę wpływu obszarów obrazu

Przykładowe, poprawnie zaklasyfikowane obrazy oraz ich mapy wpływu obszarów:

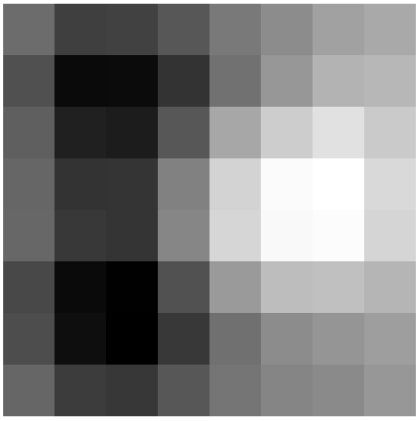
Obraz



Mapa wpływu obszarów

Rysunek 8: Poprawny przykład A - jacht motorowy. Model 'spogląda' na podniesiony dziób jachtu motorowego.

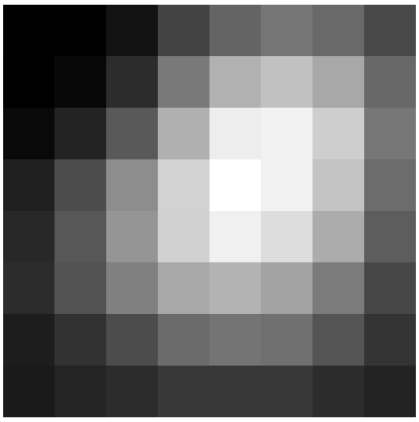
Obraz



Mapa wpływu obszarów

Rysunek 9: Poprawny przykład B - jacht żaglowy. Model skupia się na foku / grocie jachtu żaglowego.

Obraz



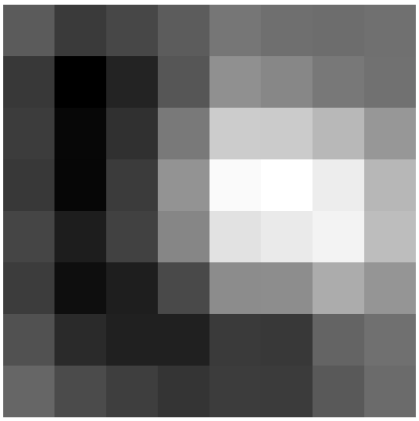
Mapa wpływu obszarów

Rysunek 10: Poprawny przykład C - katamaran. Model podejmuje decyzję na podstawie szerokiej nadbudówki katamaranu.

4.2 Niepoprawnie zaklasyfikowane obrazy

Przykładowe, niepoprawnie zaklasyfikowane obrazy oraz ich mapy wpływu obszarów:

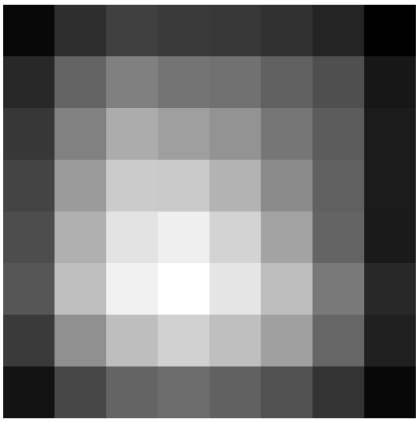
Obraz



Mapa wpływu obszarów

Rysunek 11: Nieoprawny przykład A - katamaran. Przewidziana klasa: jacht żaglowy. Model podjął tą decyzję na podstawie pozycji grota i foka - jednak człowiek (a może raczej żeglarz) jest w stanie stwierdzić, że to katamaran na podstawie szerokiej nadbudówki.

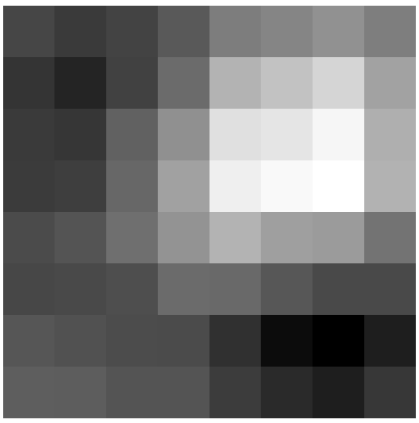
Obraz



Mapa wpływu obszarów

Rysunek 12: Nieoprawny przykład B - katamaran. Przewidziana klasa: jacht motorowy. W tym przypadku można stwierdzić, że model zachował się poprawnie. Mimo, że jednostka ma dwa kadłuby, to jest ona też jachtem motorowym (nie ma masztu ani żagli).

Obraz



Mapa wpływu obszarów

Rysunek 13: Nieoprawny przykład C - jacht żaglowy. Przewidziana klasa: katamaran. W tym przypadku pięknie widać, jak model podjął tę błędną decyzję. Po prostu założył, że jacht przycumowany po drugiej stronie Y-bomu jest ciągłą kontynuacją jachtu. Przy takim założeniu - jacht rzeczywiście byłby dwukadłubowy.