

# **Analiza zdjęć lotniczych**

**Projekt z przedmiotu TWM w semestrze 24L**

Mateusz Grzelak      Maciej Kaczkowski

Czerwiec 2024

# **Spis treści**

<b>1</b>	<b>Wstęp</b>	<b>3</b>
<b>2</b>	<b>Metody</b>	<b>4</b>
2.1	Dane . . . . .	4
2.1.1	INRIA . . . . .	4
2.1.2	Dubai . . . . .	5
2.2	Narzędzia . . . . .	6
2.3	Segmentacja semantyczna . . . . .	6
2.4	Funkcje straty . . . . .	6
2.4.1	Binarna entropia krzyżowa . . . . .	6
2.4.2	Dice . . . . .	7
2.4.3	Współczynnik korelacji Matthews . . . . .	7
2.5	Architektury modeli . . . . .	8
2.6	Monitorowanie eksperymentów . . . . .	9
<b>3</b>	<b>Wyniki</b>	<b>11</b>
3.1	INRIA . . . . .	11
3.1.1	MCCLoss + UNet . . . . .	11
3.1.2	DiceLoss + UNet . . . . .	11
3.1.3	BCELoss + UNet . . . . .	11
3.1.4	MCCLoss + DeepLabV3 . . . . .	11
3.2	Dubai . . . . .	11
<b>4</b>	<b>Dyskusja</b>	<b>20</b>
4.1	Analiza zdjęć lotniczych . . . . .	20
4.2	Trudności . . . . .	20
4.3	Inne narzędzia . . . . .	20

## **1 Wstęp**

Celem projektu była analiza problemu segmentacji semantycznej zdjęć lotniczych oraz implementacja rozwiązania dla wybranego przypadku. Wstępnej analizie poddano kilka zbiorów danych, z których wybrano 2, demonstrujące problemy związane z danymi lotniczymi oraz możliwe sposoby ich rozwiązania.

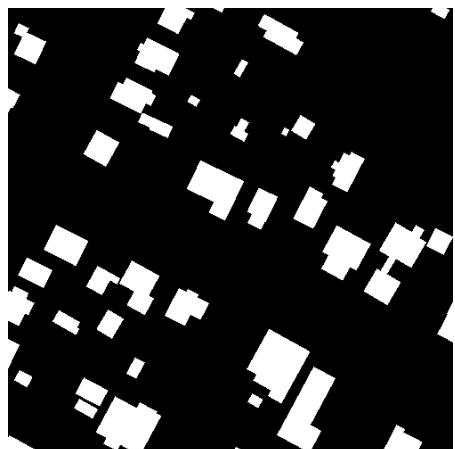
## 2 Metody

### 2.1 Dane

Po wstępnej analizie 4 zbiorów danych: INRIA [1], UAViD [2], Aerial Drone [3] oraz Dubai [4], zdecydowano się na pracę nad zbiorami INRIA oraz Dubai.

#### 2.1.1 INRIA

Zbiór INRIA, składa się z plików .tif o wymiarze 5000x5000 pikseli. Zadanie związane z tym datasetem to klasyfikacja binarna (budynek lub nie-budynek). Początkowe próby treningu na nieprzetworzonych obrazach nie przyniosły dobrych efektów, więc została podzielona na fragmenty (*patches*) 500x500 pikseli. Tym samym 100-krotnie zwiększyła się ich ilość. Poniżej przykładowe fragmenty z odpowiadającymi im maskami.

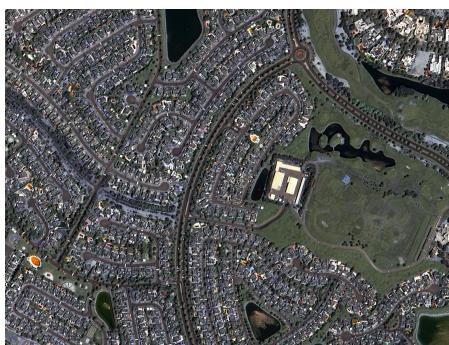
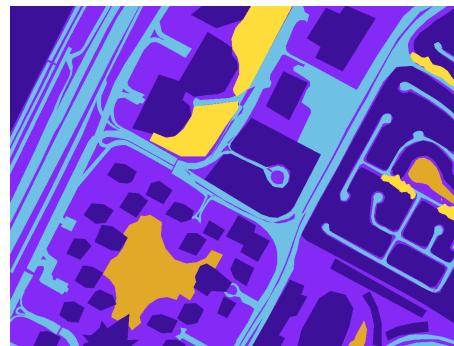
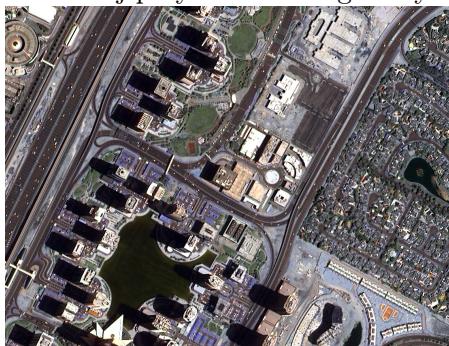


### 2.1.2 Dubai

Zbiór Dubai, składa się z 72 plików Zadanie związane z tym datasetem to klasyfikacja wieloklasowa (6 klas). Ze względu na małą ilość danych, a także ograniczoną możliwość ich augmentacji zbiór jest trudny do uzyskania dobrych wyników. Ponadto, przed treningiem należy przetworzyć maski z opisanych kolorami na klasy ze zbioru 0, 1, 2, 3, 4, 5, 6.

- Lista klas:
- Building: 3C1098
- Land (unpaved area): 8429F6
- Road: 6EC1E4
- Vegetation: FEDD3A
- Water: E2A929
- Unlabeled: 9B9B9B

Poniżej przykładowe fragmenty z odpowiadającymi im maskami.



## 2.2 Narzędzia

Do wykonania projektu użyto, między innymi następujących narzędzi:

- Python v3.12.3
- pytorch v2.3.1+cu118
- pytorch-lightning v2.2.5
- torchmetrics v1.4.0
- segmentation-models-pytorch v0.3.3

## 2.3 Segmentacja semantyczna

Segmentacja semantyczna jest zadaniem, którego polega na przypisaniu każdemu pikselowi wejściowego obrazu klasy ze skończonego zbioru. Zbiór klas zazwyczaj liczy kilka-kilkanaście klas. Segmentacja semantyczna jest jednym z najtrudniejszych zadań w wizji komputerowej. Mając dany wynik segmentacji możemy rozwiązać także inne zadania, na przykład detekcję i klasyfikację obiektów[5].

## 2.4 Funkcje straty

Najczęściej stosowaną miarą jakości segmentacji semantycznej jest **IoU** (ang. *Intersection-Over-Union*).

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

Gdzie:

- $A$  oznacza przewidziany zbiór pikseli danej klasy (*prediction*)
- $B$  oznacza prawdziwy zbiór pikseli danej klasy (*groundtruth*)

Funkcje straty najczęściej stanowią warianty **IoU**, dopasowane do specyficznych dla domeny wymagań lub zaprojektowane pod kątem dobrych właściwości matematycznych. W projekcie porównano kilka znanych funkcji straty z popularnych bibliotek [6].

### 2.4.1 Binarna entropia krzyżowa

Przypadek segmentacji binarnej możemy sprowadzić do klasyfikacji binarnej każdego piksela, a więc używać funkcji straty typowo używanych w klasyfikacji, na przykład entropii krzyżowej (*Binary Cross-Entropy, BCE*). W projekcie użyto "wygładzonej" wersji **BCE**, to znaczy z dodanym  $\epsilon$ , dla stabilności gradientu, wyrażonej wzorem jak poniżej.

$$\text{Smooth BCE}(y, \hat{y}) = -[\hat{y} \cdot \log(\hat{y}) + (1 - \hat{y}) \cdot \log(1 - \hat{y})] \quad (2)$$

Gdzie:

- $y$  jest prawdziwą klasą (0 lub 1).
- $\hat{y}$  jest przewidzianym prawdopodobieństwem.
- $\tilde{y}$  jest "wygładzoną" klasą.

$$\text{Smooth BCE}(y, \hat{y}) = -[\tilde{y} \cdot \log(\hat{y}) + (1 - \tilde{y}) \cdot \log(1 - \hat{y})] \quad (3)$$

$\tilde{y}$  jest opisana wzorem:

$$\tilde{y} = y \cdot (1 - \epsilon) + \frac{\epsilon}{2} \quad (4)$$

Zatem funkcja straty:

$$\text{Loss}_{\text{SmoothBCE}} = \text{SmoothBCE} \quad (5)$$

#### 2.4.2 Dice

Współczynnik Dice opisuje stopień podobieństwa pomiędzy dwoma próbками. Jest uznawany za szczególnie przydatny przy niezbalansowanych zbiorach danych. Współczynnik Dice opisujemy wzorem jak poniżej.

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \quad (6)$$

Gdzie :

- $A$  oznacza przewidziany zbiór pikseli danej klasy (*prediction*)
- $B$  oznacza prawdziwy zbiór pikseli danej klasy (*groundtruth*)

Zatem funkcja straty:

$$\text{Loss}_{\text{Dice}} = 1 - \text{Dice} \quad (7)$$

#### 2.4.3 Współczynnik korelacji Matthews

W pracy [7] autorzy zauważali, że typowe funkcje straty, tak jak te opisane powyżej, w niewystarczającym stopniu penalizują błędy polegające na klasyfikacji obszarów należących do danej klasy jako należące do tła (*False Negatives, FN*). Współczynnik korelacji Matthews (*Matthews Correlation Coefficient, MCC*) jest, wywodzącym się z domeny obrazowania medycznego, sposobem na poradzenie sobie ze wspomnianym problemem. Jest to o tyle ważne, że w zastosowaniach medycznych *FN* oznaczają najgorszy możliwy scenariusz, czyli chorą tkankę zaklasyfikowaną jako zdrową. Współczynnik korelacji Matthews wyraża się wzorem jak poniżej.

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (8)$$

Gdzie:

- $TP$  to ilość prawidłowych klasyfikacji pozytywnych.
- $TN$  to ilość prawidłowych klasyfikacji negatywnych.
- $FP$  to ilość błędnych klasyfikacji pozytywnych.
- $FN$  to ilość błędnych klasyfikacji negatywnych.

**MCC** przyjmuje wartości od  $-1$  do  $+1$ :

- $+1$  oznacza idealną predykcję.
- $0$  oznacza predykcję losową.
- $-1$  oznacza, że przewidziane klasy są odwrotnością prawidziwych.

Zatem funkcja straty:

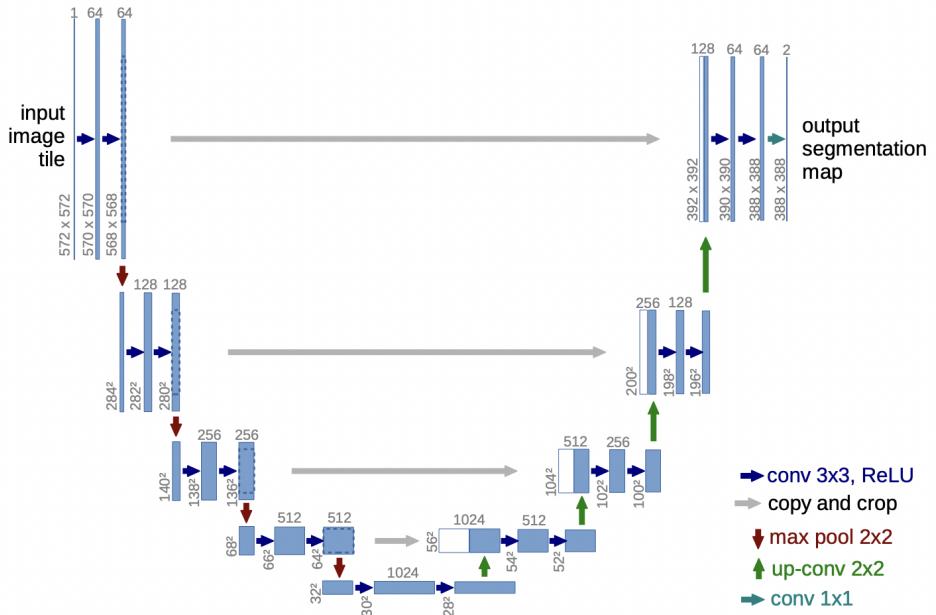
$$\text{Loss}_{\text{MCC}} = 1 - \text{MCC} \quad (9)$$

## 2.5 Architektury modeli

Typowe architektury używane do segmentacji semantycznej opierają się najczęściej na ogólnym schemacie "ekstrakcja cech -> klasyfikacja" lub "enkoder-dekoder". W projekcie skorzystano z jednej z popularniejszych architektur: **UNet**[8]. Opiera się ona na kodowaniu, a następnie dekodowaniu obrazu, w celu ekstrakcji cech, przy czym pomiędzy odpowiadającymi sobie warstwami "piramidy" występują połączenia rezydualne, a latent ma wymiar 1024. UNet nie jest nową architekturą, od momentu powstania doczekał się wielu modyfikacji [9].

Przy pomocy biblioteki **segmentation\_models.pytorch** [10] definiujemy model jako:

```
model = smp.Unet(
    encoder_name="resnet18",
    encoder_weights="imagenet",
    in_channels=3,
    classes=1,
    activation="sigmoid",
).to(device)
```



Rysunek 1: Architektura UNet

## 2.6 Monitorowanie eksperymentów

Do zarządzania eksperymentami wykorzystano bibliotekę **pytorch-lightning**, przykładowy moduł wygląda jak poniżej.

```
class SegmentationModel(pl.LightningModule):
    def __init__(self, model, learning_rate=1e-3):
        super(SegmentationModel, self).__init__()

        self.model = model
        self.learning_rate = learning_rate
        self.criterion = smp.losses.DiceLoss(mode="binary")
        self.train_iou = JaccardIndex(num_classes=2, task="binary")
        self.val_iou = JaccardIndex(num_classes=2, task="binary")

    def forward(self, x):
        output = self.model(x.to(device))
        return output

    def training_step(self, batch, batch_idx):
        images, masks = batch
        masks = torch.div(masks, 255).float()
        preds = self(images)
```

```

loss = self.criterion(preds, masks)

self.log("train_loss", loss, on_epoch=True, on_step=True)
self.log("train_iou", self.train_iou(preds, masks), on_epoch=True, on_step=True)

return loss

def validation_step(self, batch, batch_idx):
    images, masks = batch
    masks = torch.div(masks, 255).float()
    preds = self(images)
    loss = self.criterion(preds, masks)

    self.log("val_loss", loss, on_epoch=True, on_step=True)
    self.log("val_iou", self.val_iou(preds, masks), on_epoch=True, on_step=True)

    return loss

def test_step(self, batch, batch_idx):
    # just here to activate the test_epoch_end
    # callback SaveTestPreds starts on_test_epoch_end
    pass

def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)

```

### 3 Wyniki

Eksperymenty przeprowadzane z parametrami:

- learning rate = 0.001
- batch size = 16 (dla INRIA), lub
- batch size = 1 (dla Dubai)

Przetestowano kombinacje innych hiperparametrów jak poniżej:

Zbiór danych	architektura	funkcja straty
INRIA	UNet	DiceLoss
INRIA	UNet	MCCLoss
INRIA	Unet	BCELoss
INRIA	DeepLabV3	MCCLoss
Dubai	UNet	JaccardLoss

**Tabela 1:** Zbiory danych, architektury i funkcje straty

#### 3.1 INRIA

Dla zbioru INRIA uzyskano porównywalne wyniki pomiędzy różnymi funkcjami straty.

Przykładowe wyniki segmentacji poniżej.

##### 3.1.1 MCCLoss + UNet

##### 3.1.2 DiceLoss + UNet

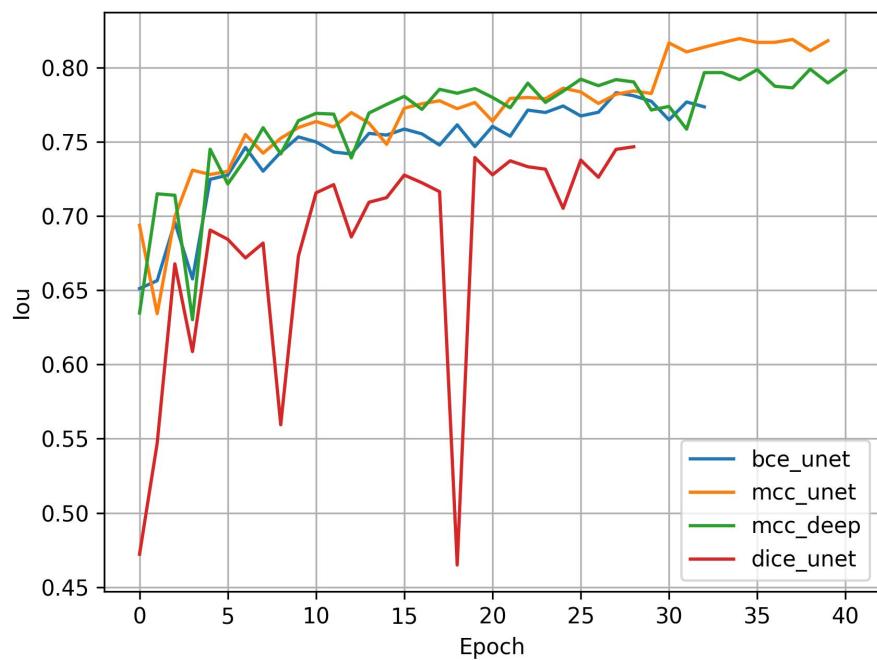
##### 3.1.3 BCELoss + UNet

##### 3.1.4 MCCLoss + DeepLabV3

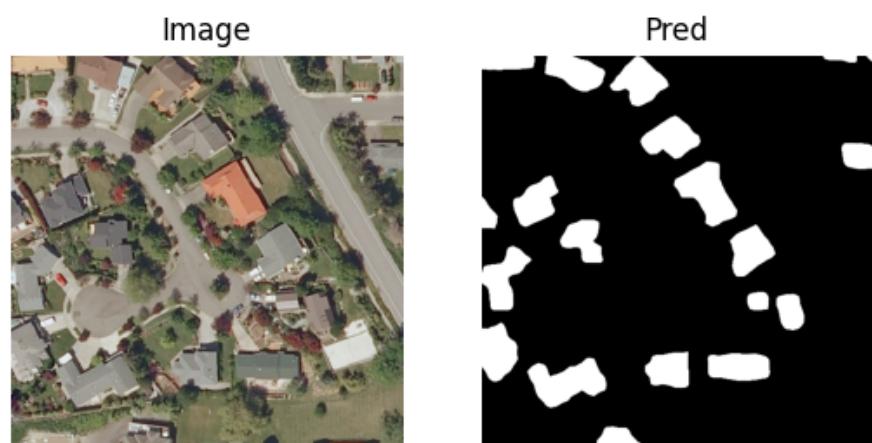
#### 3.2 Dubai

Dla zbioru Dubai uzyskano ogólnie gorsze wyniki, w porównaniu z INRIA.

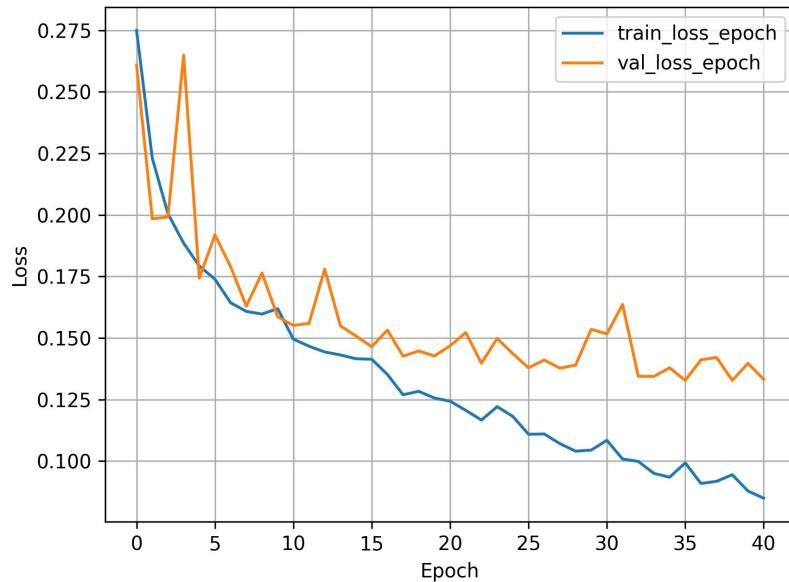
Przykładowe wyniki segmentacji poniżej.



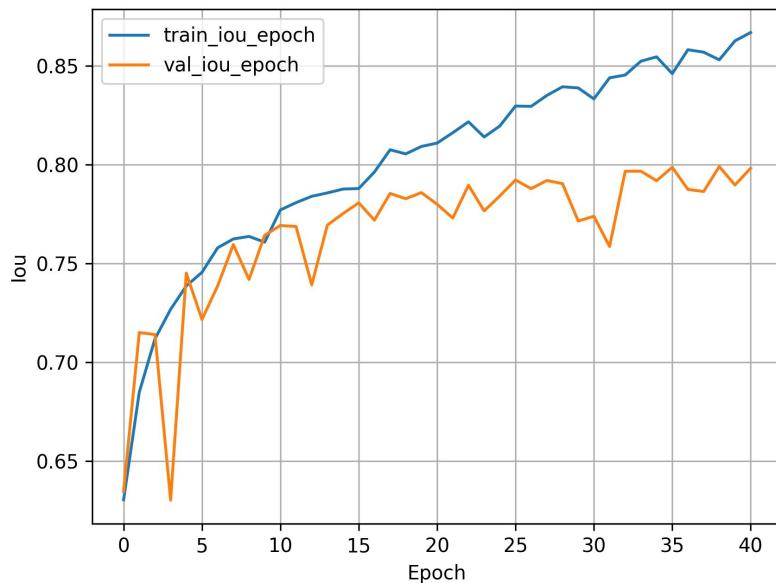
Rysunek 2: INRIA wyniki łączne



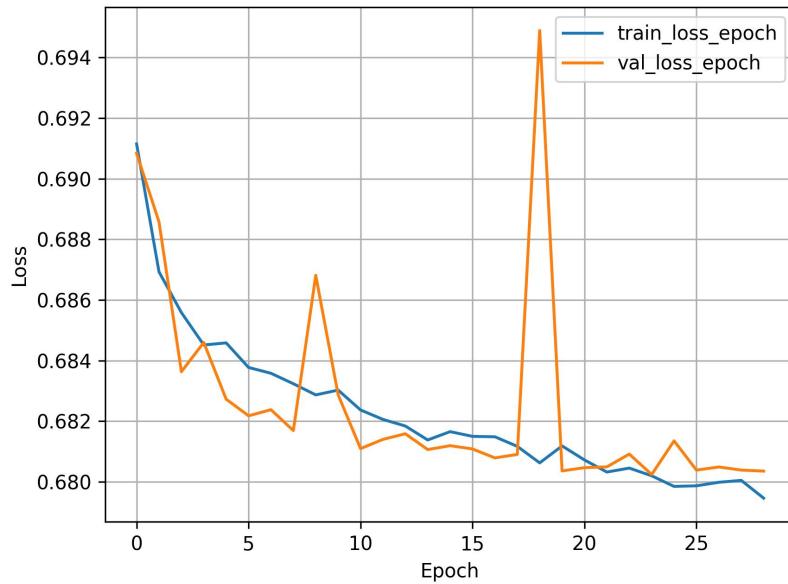
**Rysunek 3:** INRIA przykład ze zbioru testowego



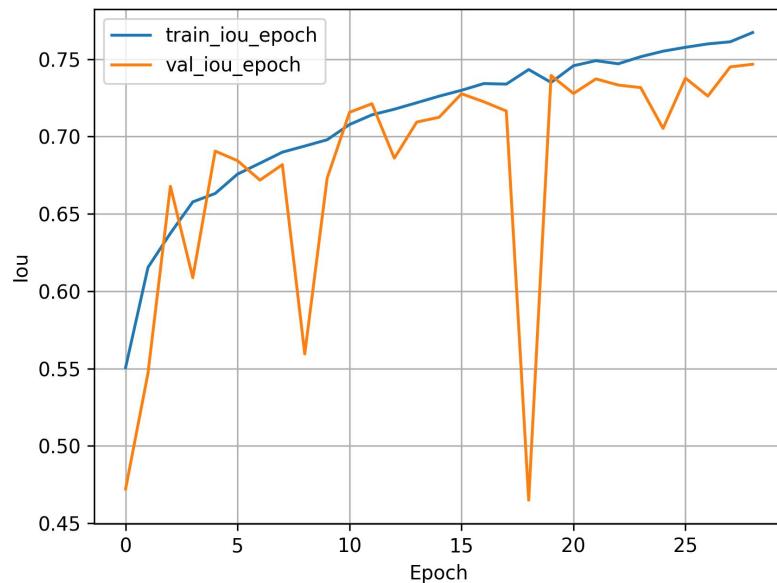
Rysunek 4: INRIA MCCLoss Unet Loss



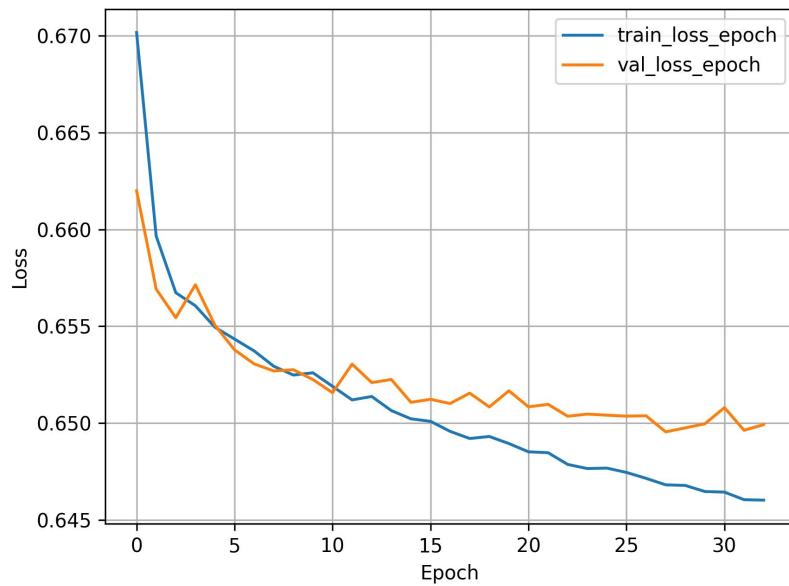
Rysunek 5: INRIA MCCLoss Unet IoU



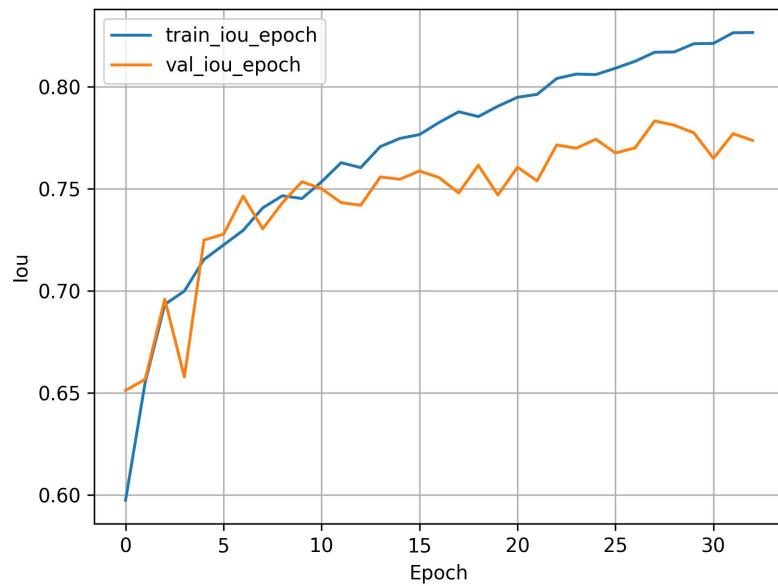
Rysunek 6: INRIA DiceLoss Unet Loss



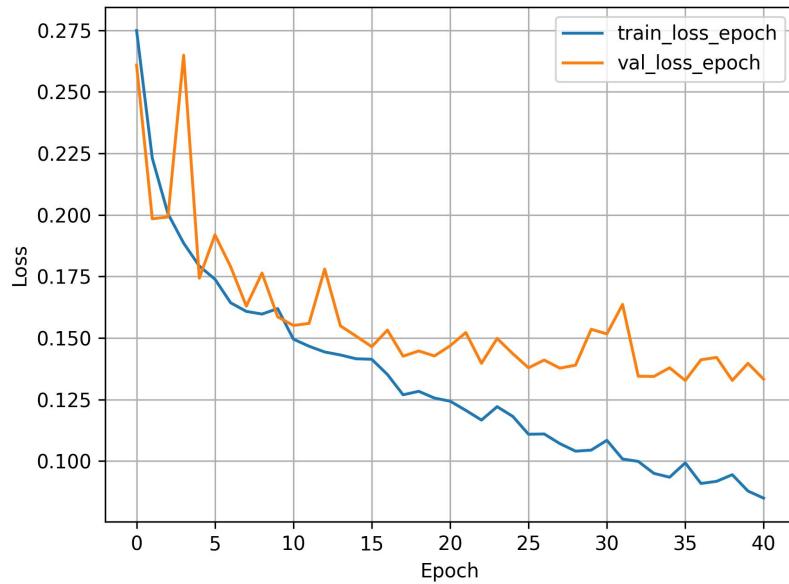
Rysunek 7: INRIA DiceLoss Unet IoU



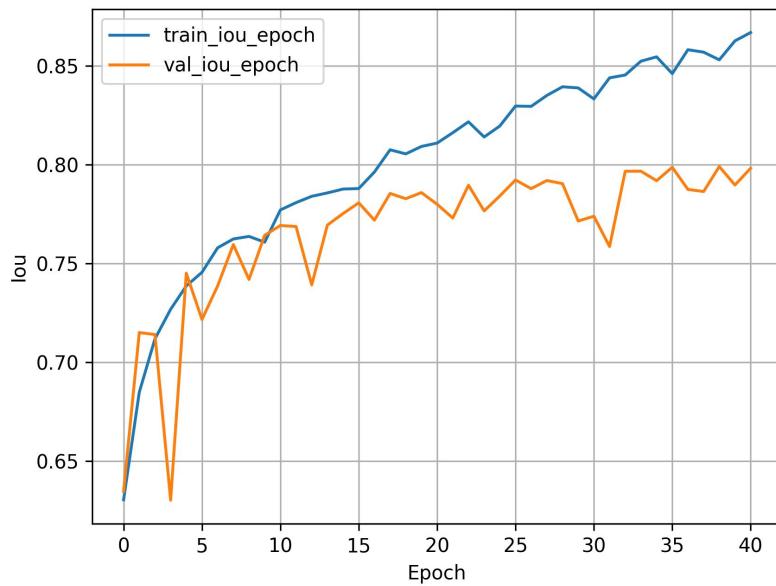
Rysunek 8: INRIA BCELoss Unet Loss



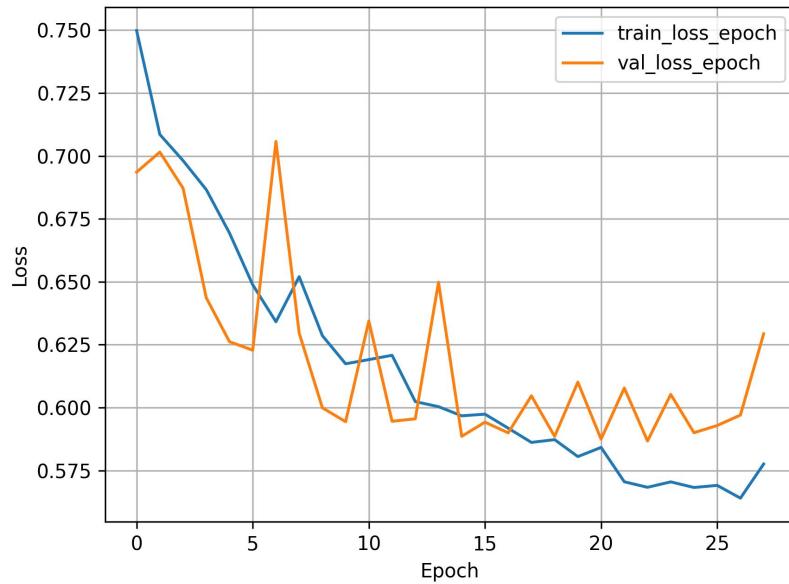
Rysunek 9: INRIA BCELoss Unet IoU



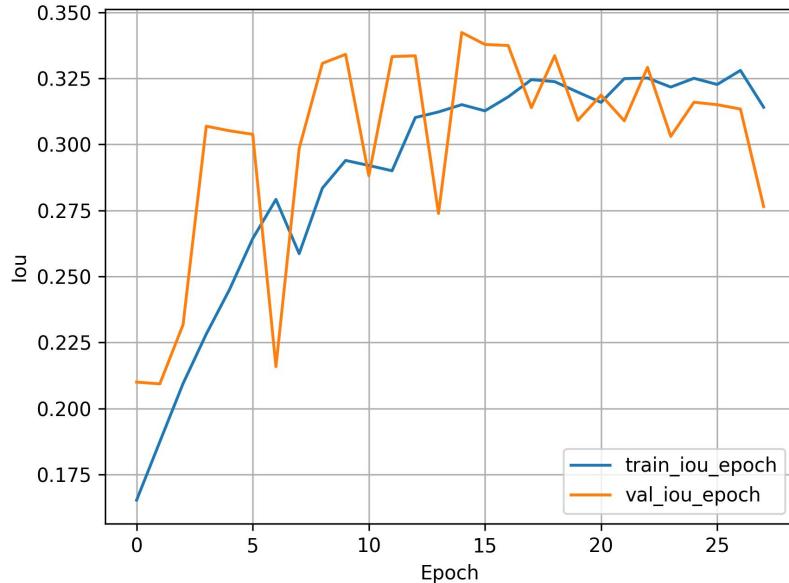
Rysunek 10: INRIA MCCLoss DeepLabV3 Loss



Rysunek 11: INRIA MCCLoss DeepLabV3 IoU



Rysunek 12: Dubai Loss



Rysunek 13: Dubai IoU



**Rysunek 14:** Dubai przykład ze zbioru testowego

## 4 Dyskusja

### 4.1 Analiza zdjęć lotniczych

Główne trudności, które napotkano, można podzielić na 2 główne kategorie: związane z danymi oraz związane z ograniczeniami sprzętowymi. Dane lotnicze zawierają najczęściej obrazy o bardzo dużej rozdzielczości (rzędu 5k x 5k), obejmujące duże obszary. Przed rozpoczęciem treningu lepiej podzielić je na mniejsze fragmenty *chips*, *patches*, ze względu na fakt, że w przeciwnym razie może dojść do błędów *out-of-memory*. Oprócz tego, obrazy mogą być zapisywane są w nietypowych formatach, np. .tiff. W zastosowaniach produkcyjnych, konieczne będzie także zapewnienie wysokiej przepływności systemów służących do analizy danych lotniczych oraz ich szybkości, jeżeli wymagane jest działanie w czasie prawie rzeczywistym.

### 4.2 Trudności

Ograniczenia sprzętowe, takie jak brak dostępu do GPU z odpowiednią ilością *cuda cores* oraz VRAM, powodują, że trening trwa więcej czasu, co zmniejsza ilość eksperymentów, które można przeprowadzić.

### 4.3 Inne narzędzia

Ze względu na opisane powyżej cechy zdjęć lotniczych, na dłuższą metę i w zastosowaniach profesjonalnych prawdopodobnie warto korzystać z dedykowanych rozwiązań oferujących bogatsze, predefiniowane narzędzia i/lub *framework* programistyczny do pracy z danymi lotniczymi. Przykładowe narzędzia, której możemy zaliczyć do tej kategorii to **rastervision** [11] oraz **torchgeo** [12].

## Bibliografia

- [1] *INRIA dataset*. Remote access (10.06.2024): <https://project.inria.fr/aerialimagelabeling>.
- [2] *UAVid dataset*. Remote access (10.06.2024): <https://www.kaggle.com/datasets/dasmehdixtr/uavid-v1>.
- [3] *Aerial Drone dataset*. Remote access (10.06.2024): <https://www.kaggle.com/datasets/nunenuh/semantic-drone>.
- [4] *Dubai dataset*. Remote access (10.06.2024): <https://www.kaggle.com/datasets/humansintheloop/semantic-segmentation-of-aerial-imagery>.
- [5] G. Csurka et al. „Semantic Image Segmentation: Two Decades of Research”. W: (2023).
- [6] *smp.losses*. Remote access (10.06.2024): <https://smp.readthedocs.io/en/latest/losses.html>.
- [7] K. Abhishek i G. Hamarneh. *Matthews Correlation Coefficient Loss For Deep Convolutional Networks: Application To Skin Lesion Segmentation*. Remote access (10.06.2024): <https://www.cs.sfu.ca/~hamarneh/ecopy/isbi2021.pdf>. 2024.
- [8] O. Ronneberger, P. Fischer i T. Brox. „U-Net: Convolutional Networks for Biomedical Image Segmentation”. W: (2015).
- [9] Z. Zhou et al. „UNet++: A Nested U-Net Architecture for Medical Image Segmentation”. W: (2018).
- [10] *segmentation\_models.pytorch*. Remote access (10.06.2024): [https://github.com/qubvel/segmentation\\_models.pytorch?tab=readme-ov-file](https://github.com/qubvel/segmentation_models.pytorch?tab=readme-ov-file).
- [11] *rastervision repository*. Remote access (10.06.2024): <https://github.com/azavea/raster-vision>.
- [12] *torchgeo repository*. Remote access (10.06.2024): <https://github.com/microsoft/torchgeo>.