Analiza zdjęć lotniczych

Mateusz Grzelak Maciej Kaczkowski

Czerwiec 2024

Spis treści

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

2.1.1 INRIA

2.1.2 Dubai

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[1].

2.4 Funkcje straty

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

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

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 [2].

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 ϵ, dla stabilności gradientu, wyrażonej wzorem jak poniżej.

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

Gdzie:

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

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

$\tilde{y}$ jest opisana wzorem:

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

Zatem funkcja straty:

$$\text{Loss}_{\text{SmoothBCE}} = \text{SmoothBCE} \tag{5}$$

2.4.2 Dice

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

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \tag{6}$$

Gdzie :

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

Zatem funkcja straty:

$$\text{Dice Loss} = 1 - \text{Dice} \tag{7}$$

2.4.3 Współczynnik korelacji Matthews

W pracy [3] autorzy zauważayli, 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.

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

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.

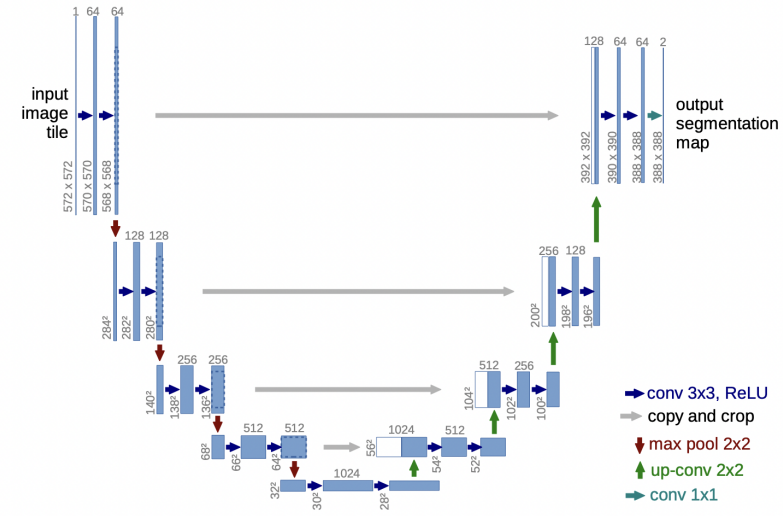
Zatem funkcja straty:

$$\mathrm{Loss}_{\mathrm{MCC}} = 1 - \mathrm{MCC} \tag{9}$$

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**[4]. 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 [5].

Przy pomocy biblioteki **segmentation_models.pytorch** [6] definiujemy model jako:



Rysunek 1: Architektura UNet

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

2.6 Monitorowanie eksperymentów

Do zarządznia 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

3.0.1 INRIA

3.0.2 Dubai

4 Dyskusja

4.1 Trudności

4.2 Analiza zdjęć lotniczych

4.3 Inne narzędzia

Bibliografia

[1] G.Csurka et al. „Semantic Image Segmentation: Two Decades of Research". W: (2023).

[2] *smp.losses*. Remote access (10.06.2024): `https://smp.readthedocs.io/en/latest/losses.html`.

[3] G.Hamarneh K.Abhishek. *Matthews Correlation Coefficient Loss For Deep Concolutional Neutworks: Application To Skin Lesion Segmentation*. Remote access (10.06.2024): `https://www.cs.sfu.ca/~hamarneh/ecopy/isbi2021.pdf`.

[4] T.Brox O.Ronneberger P.Fischer. „U-Net: Convolutional Networks for Biomedical Image Segmentation". W: (2015).

[5] Z.Zhou et al. „UNet++: A Nested U-Net Architecture for Medical Image Segmentation". W: (2018).

[6] $segmentation_m odels.pytorch$. Remote access (10.06.2024): `https://github.com/qubvel/segmentation_models.pytorch?tab=readme-ov-file`.