Deep Learning Homework

Deep Learning Házi feladat

Airbus Ship Detection Challenge

Nagy Dániel (UU5SCQ)
Szénássy Márton (HIYXIQ)
Varga-Labóczki Vazul (H947XW)

January 7, 2026

Abstract

This report presents our solution to the Airbus Ship Detection Challenge using a fully convolutional encoder–decoder architecture for ship segmentation in satellite images. We discuss our initial attempts with Segment Anything and the challenges we faced, leading us to adopt the FCN model. Our implementation includes various callbacks to enhance training, and we address challenges such as class imbalance through data augmentation. Overall, our solution demonstrates the effectiveness of deep learning techniques in addressing real-world problems in maritime surveillance.

Kivonat

Ebben a dolgozatban bemutatjuk a megoldásunkat az Airbus Ship Detection Challenge kihívásra, amelyben FCN architektúrát alkalmaztunk a műholdfelvételeken megjelenő hajók szegmentálására. Ismertetjük kezdeti kísérleteinket a Segment Anything modellel, valamint azokat a kihívásokat, amelyekkel szembesültünk, és amelyek arra késztettek minket, hogy a FCN modellt válasszuk. Megvalósításunk különböző callback mechanizmusokat tartalmaz a tanulási folyamat javítása érdekében, és foglalkozunk olyan kihívásokkal, mint az osztályok közötti egyensúlyhiány, amelyet adat augmentációs technikákkal kezelünk. Összességében megoldásunk bemutatja a mélytanulási technikák hatékonyságát a tengeri megfigyelés valós problémáinak kezelésében.

Contents

1 Introduction

This report presents our solution to the Airbus Ship Detection Challenge [1]. This challenge focuses on the task of detecting ships in satellite images using deep learning techniques [3, 8]. The challenge addresses a real-world problem in maritime surveillance and ocean monitoring, where automated detection systems can significantly reduce manual labor and improve response times. We solved this problem by developing a convolutional neural network-based segmentation model that accurately identifies ship locations in satellite imagery, drawing on recent advances in fully convolutional networks [9].

2 Description of topic and previous solutions

The Airbus Ship Detection Challenge is a competition hosted on the Kaggle platform, aiming to develop algorithms for detecting ships in satellite images. The challenge provides a dataset of high-resolution satellite images, each annotated with masks indicating the presence and location of ships. Participants are tasked with creating models that can accurately segment ships from the background in these images.
Previous solutions include various deep learning architectures, such as U-Net [11], Mask R-CNN [4], DeepLabv3+ [2], and Segment Anything [7]. These models leverage convolutional neural networks (CNNs) to learn spatial hierarchies of features from the input images, often building on strong backbones like VGG [13]. Many participants have also employed data augmentation techniques to enhance model robustness and improve generalization to unseen data [10, 12].

3 Architecture

3.1 First attempt

Our initial approach involved implementing Segment Anything [7], a state-of-the-art segmentation model known for its versatility and performance across various segmentation tasks. However, we encountered significant challenges during this phase due to the model's way of working. Segment Anything is designed to generate segmentation masks based on user-provided prompts, such as points or bounding boxes. This interactive nature made it difficult to adapt the model for fully automated ship detection in satellite images, as required by the challenge. This made it impractical for our specific use case, leading us to explore alternative architectures better suited for automated segmentation tasks.

3.2 Final model

After evaluating various architectures, we implemented a fully convolutional network (FCN). The chosen FCN is a simple encoder–decoder network composed exclusively of convolutional layers (no dense layers), with learned downsampling via strided convolutions and learned upsampling via transpose convolutions. Unlike U-Net, this architecture does not use skip-connections between encoder and decoder paths.

Key characteristics of our FCN:

- Encoder: several Conv2D blocks with ReLU activations and BatchNormalization [5]. Downsampling is performed using strided convolutions that reduce spatial resolution (e.g. 256 -> 128 -> 64 -> 32 -> 16).
- Decoder: Conv2DTranspose layers with ReLU activations and BatchNormalization [5] to restore spatial resolution back to the input size.
- Classifier: a final Conv2D producing a single-channel output (kernel size 5, padding='same'). The model emits logits for binary segmentation; a sigmoid can be applied at loss/metric time.
- BatchNormalization is applied after each hidden convolutional layer to stabilize training.

```
class SegmentationModel():
    def __init__(self):
        self.model = Sequential()
        # Encoder
        self.model.add(Conv2D(filters=16, kernel_size=3, activation='relu', padding ='same', input_shape=(IMG_HEIGHT,IMG_WIDTH,3)))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=32, kernel_size=3, activation='relu', padding ='same', strides = 2))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=32, kernel_size=3, activation='relu', padding ='same'))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=64, kernel_size=3, activation='relu', padding ='same', strides = 2))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=64, kernel_size=3, activation='relu', padding ='same'))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=128, kernel_size=3, activation='relu', padding ='same', strides = 2))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=128, kernel_size=3, activation='relu', padding ='same'))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=256, kernel_size=3, activation='relu', padding ='same', strides = 2))
        self.model.add(BatchNormalization())
        self.model.add(Conv2D(filters=256, kernel_size=3, activation='relu', padding ='same'))
        self.model.add(BatchNormalization())

        # Decoder
        self.model.add(Conv2DTranspose(filters=128,  kernel_size=3,  strides=2, padding='same', activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Conv2DTranspose(filters=64,  kernel_size=3,  strides=2, padding='same', activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Conv2DTranspose(filters=32,  kernel_size=3,  strides=2, padding='same', activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Conv2DTranspose(filters=16,  kernel_size=3,  strides=2, padding='same', activation='relu'))
        self.model.add(BatchNormalization())

        # Classifier
        self.model.add(Conv2D(filters=1, kernel_size=5, padding = 'same'))
```

Figure 1: Model architecture (fully convolutional encoder–decoder).

3.2.1 Callbacks

To enhance the training process and improve model performance, we incorporated several callbacks into our training routine:

- **Model Checkpointing**: We used model checkpointing to save the best model weights based on validation loss during training. This ensures that we retain the most effective model configuration.
- **Early Stopping**: Early stopping was implemented to monitor the validation loss and halt training if no improvement was observed for a specified number of epochs. This helps prevent overfitting and reduces unnecessary computation.
- **Learning Rate Reduction**: We employed a learning rate reduction strategy that decreases the learning rate when the validation loss plateaus. This allows the model to fine-tune its weights more effectively during later stages of training.

4 Implementation

4.1 Training

We trained our FCN model using the Adam optimizer [6] with a learning rate of 0.003 and a batch size of 32. The training was conducted on our own hardware, utilizing a GPU to accelerate the training process. The model was trained for 1000 epochs, with early stopping implemented to prevent overfitting.

4.1.1 Preprocessing

To enhance the model's performance, we incorporated data augmentation techniques during training, such as random rotations, flips, and zooms [10, 12]. This is the main task of the preprocessing step, as it increases the diversity of the training data and helps prevent overfitting. Additionally, we applied batch normalization [5] to stabilize the training process.

4.1.2 Problems

During the training process, we encountered several challenges that required careful consideration and adjustments to our approach. One of the primary issues was dealing with class imbalance in the dataset, as the number of pictures containing ships was significantly lower than the pictures without ships. To address this, we implemented data augmentation techniques to increase the representation of ship-containing images in the training set. The final ratio is 50-50 between images with and without ships.

5 Summary

In this report, we presented our approach to the Airbus Ship Detection Challenge using a fully convolutional encoder–decoder architecture for ship segmentation in satellite images. We discussed our initial attempts with Segment Anything and the challenges we faced, leading us to adopt the FCN model. Our implementation included various callbacks to enhance training, and we addressed challenges such as class imbalance through data augmentation. Overall, our solution demonstrates the effectiveness of deep learning techniques in addressing real-world problems in maritime surveillance [9, 2, 14].

References

[1] Airbus. Airbus ship detection challenge. Kaggle, 2018. URL `https://www.kaggle.com/c/airbus-ship-detection`.

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision*, pages 833–851, 2018.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Alexander Kirillov, Eric Mintun, Nikhila Ravi, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[10] Luis Pérez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[12] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(60), 2019.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 3–11, 2018.