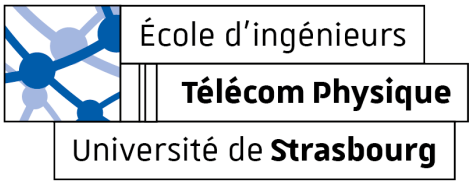

THALES

Engineering Degree : Images, Signals and Data Science

Télécom Physique Strasbourg

Master IRIV: Images and Data

University of Strasbourg

Academic year 2022 - 2023

Maritime images semantic segmentation

Using neural network architectures

Simon BERTRAND

March to August 2023 - Internship

September 2023 - Oral presentation

Tutor :

Vinkle Srivastav

Company supervisors :

Klaas Mussche, Hans de Groot

Public report

Summary

"Maritime images semantic segmentation with deep learning methods"

After a long winter in the 1980s, the field of data science, also known as statistics, has experienced significant advancements in both methodology and tools. High-dimensional differential geometry enables companies and scientists to make their data relevant by understanding the deep knowledge and patterns behind it. The applications are limitless as long as suitable data for unsupervised methods and annotated data for supervised techniques are accessible. Semantic segmentation is a challenging task that involves analyzing images to capture contextual information about the environment depicted in them. It divides an image into multiple meaningful and semantically coherent regions or segments. The goal is to separate different objects or regions of interest within an image, enabling a computer to understand and analyze the image at a more granular level. For maritime environment, those semantic labels can be the sea, sky, vehicles, land, etc. Thales Netherlands B.V. is exploring deep learning methods to enhance the features of their optical systems, such as the Gatekeeper. The image segmentation task is one of the capabilities they wish to add.

As this internship is research-based, my first step will be to analyze the current state-of-the-art in image semantic segmentation. Following this, I will develop and validate many underlying algorithms on maritime data, which will be the main focus of this report.

Résumé

“Segmentation sémantique d’images maritimes par apprentissage profond“

Après un long hiver dans les années 1980, le domaine de la science des données, aussi connu sous le nom de statistiques, a connu des avancées significatives à la fois en termes de méthodologie et d’outils. L’utilisation de la géométrie différentielle à haute dimension permet maintenant aux entreprises et aux scientifiques de rendre leurs données plus pertinentes en comprenant les connaissances et les modèles complexes qui les sous-tendent. Les applications de cette approche sont illimitées, à condition d’avoir accès à des données appropriées pour les méthodes non supervisées et à des données annotées pour les techniques supervisées. L’une des tâches les plus réalisée dans ce domaine est la segmentation sémantique, qui consiste à analyser des images afin de capturer des informations contextuelles sur l’environnement représenté. Cette technique divise une image en plusieurs régions ou segments significatifs et sémantiquement cohérents, permettant ainsi à un ordinateur de comprendre et d’analyser l’image de manière plus détaillée. Dans le contexte maritime, les étiquettes sémantiques peuvent inclure des éléments tels que la mer, le ciel, les véhicules, la terre, etc. Thales Netherlands B.V. explore les méthodes d’apprentissage profond pour améliorer les caractéristiques de ses systèmes optiques, notamment le Gatekeeper. Ils souhaitent spécifiquement ajouter la possibilité de segmenter des images à leur système.

Ce stage étant axé sur la recherche, ma première étape consistera à analyser l’état actuel des connaissances en matière de segmentation sémantique des images. Ensuite, je développerai et validerai de nombreux algorithmes sous-jacents sur des données maritimes, qui seront décrits dans ce rapport.

Table of contents

Figures

Tables

I. Introduction

As a defense company, Thales' systems need to be highly efficient and should enable humans to concentrate on the core tasks of their work. A specific example of a broader task is monitoring the sides of a military ship during a shift. It requires 24/7 surveillance, which is typically done by humans. Using optical or infrared cameras, remote monitoring can be conducted, but operators still need to be present to analyze the footage captured by the cameras. To increase the level of automation in these systems, we can utilize AI, specifically computer vision algorithms. These automatons will run all day long without interruptions, sometimes be faster and more accurate than humans and are much more stable as they are not subject to emotions. For the manual monitoring task, an image semantic segmentation algorithm can be served to identify objects appearing in the video and extract regions of interest but we can use it for many others applications.

The goal of this internship is to develop a complete deep learning pipeline for performing image semantic segmentation which aims for a real-time application. The data are time series images of the maritime environment which can be defined by the sea, the sky and the vessels sides. The concept of semantic segmentation is to find a third dimension for the image, where this new dimension is categorical and can be one-hot encoded. We will assign a class to each pixel that corresponds to different objects and parts present in the image. With military vessels, there will be multiple classes such as the land class, the sea class, the sky class, etc. The classes are predetermined by the developer. The image segmentation task can be seen as a mathematical function f, where X represents the input image, W represents the width of the image, H the height, D the number of channels, and C the total number of classes.

$$f : X \in \mathbb{R}^{W \times H \times D} \mapsto Y \in [\![1, C]\!]^{W \times H} \tag{1}$$

We obtain a final categorical feature map (i.e., the segmentation map) Y that describes the different semantic segments. Each class between 1 and C can be mapped to a word or a group of words that describe the object or the part of the image. In fact, real-time semantic segmentation is a little more complex. We can first segment the image with the hope that the computation is fast enough to be applied in real time. With longer computation, we can consider a in time domain algorithm and create slightly different function f, that takes previous time step images as input in order to consider the image flow (similar to optical flow) over time. It means that our original image timeserie is now $X \in \mathbb{R}^{W \times H \times D \times N_t}$ where N_t is the total number of time steps. Additionally, our function f now takes a time subset

of the timeserie data with (n_t, n_0) such as $n_t, n_0 \in [\![1, N_t]\!]$ and $n_0 < n_t$:

$$f : X_t \in \mathbb{R}^{W \times H \times D \times (n_t - n_0)} \mapsto Y_t \in [\![1, C]\!]^{W \times H} \tag{2}$$

The main technology we will use to resolve this problem is deep learning. We need to modify the function a slightly in order to adjust the above function for the neural network output. To accomplish this, we will utilize an intermediary function that will calculate saliency maps. A saliency map is a spatial probability map that assigns pixels a probability to belong a given class on an element-wise basis. As we have C classes, we will have C saliency maps, one for each class.

$$f_1 : X \in \mathbb{R}^{W \times H \times D} \mapsto Y \in [0, 1]^{W \times H \times C} \tag{3}$$

Then, we will compose f_1 with f_2 which is defined in the following equation and corresponds to the argument of the maximum over the channel depth :

$$f_2 = \arg\max_{c \in C} \tag{4}$$

Thus, we have :

$$f_2 \circ f_1 = f \tag{5}$$

Our model is now theorized, so we can focus on finding f_1 using a neural network architecture.

II. Company presentation

1. Thales

Thales is a world-renowned French technology and defense company. Founded in 2000, Thales has grown over the decades to become a world leader in aerospace, defense, security and information technology. The company specializes in the design, development and supply of cutting-edge technological solutions to meet the complex needs of its customers in the defense, aerospace, transportation, security and identification sectors. In 1893, Compagnie Française Thomson-Houston (CFTH) was established with the purpose of operating the patents of the US Thomson-Houston Electric Corp., focusing on the emerging power generation and transmission market. On other hand, CSF, founded in 1918, gained recognition as a pioneer in broadcasting. In 1957, CSF acquired Société Française Radioélectrique (SFR), solidifying its position as a prominent player in the development of broadcasting, short wave, electro-acoustics, radar, and television systems during the 1930s. As time passed, CFTH underwent changes, eventually merging with Compagnie Générale de Télégraphie Sans Fil (CSF) to create Thomson-CSF. Before 2000, Thales group was previously named Thomson-CSF, which is a merge of the company Thomson-Brandt and CSF that occurred in 1968. The company is now presided by Patrice Caine, a French Polytechnique engineer and is mainly stock-owned by the French government, Dassault Aviation (e.g., Rafale fighter-jet manufacturer) and the Thales workers. The head office is located in the west of Paris in the La Défense district. Thales offers a wide range of products and services, from secure communication systems and cyber-defense solutions to command and control systems, radars, navigation equipment and air traffic management systems. The company invests heavily in research and development, collaborating with academic and industrial partners to push back the boundaries of technology. As a multinational company, Thales operates in over 68 countries worldwide, with a significant presence in Europe, North America, Asia-Pacific, the Middle East and Africa. It has more than 81 000 employers worldwide and generated a sale revenue of 16,2 billions in 2021 with 1,1 billion of net income. The company has a number of subsidiaries such as Thales Communications & Security, Thales Alenia Space, Thales (Portugal), Thales Group (Germany) and also Thales Nederland with which we will focus on the next section. The application sectors of the Thales group are many : aeronautic, naval, ground. For example, Thales Bordeaux is producing the radars of the Rafale (airplanes radars) and Thales Hengelo is developing the ground radars but also the military vessels included systems.

2. Thales Netherlands B.V.

Thales Nederland B.V., formerly Hollandse Signaalapparaten B.V. or in short Signaal, is located in Hengelo, Netherlands and was founded in 1922 as a naval fire-control systems producer by Hazemeyer and Siemens & Halske. The company counts 2250 employers in 2022 and it's now parts of the Thales Nederland subsidiary. During the Second World War, the German Army seized and looted the factory which has been nationalized by the Dutch government after the war. In 1956, the Netherlands-based electronics company Philips made a substantial leap forward by acquiring a majority stake in Signaal. This strategic move resulted in a significant expansion for the company. Over the course of the Cold War era, Signaal thrived by producing a wide range of naval electronics and defense systems for a global customer base. In 1990, Signaal was acquired by Thomson-CSF, a prominent French electronics and defense contractor which was described in the previous section, leading to its rebranding as Thomson-CSF Signaal. Subsequently, in 2000, as part of the renaming of Thomson-CSF to Thales, the company underwent a transformation and became known as Thales Nederland. Thales Nederland's primary emphasis lies in the development of advanced naval defense systems, including sensors, radars, and infrared technology. Furthermore, the company is actively involved in air defense, communications, optronics, cryogenic cooling systems, and navigation products.

3. EO team and the Gatekeeper

I will be interning with the EO (Electrical-Optical) team at Thales Nederland B.V. The EO team is responsible for the research and development of electrico-optical systems, such as the Gatekeeper and the Mirador. The team is composed of software engineers, systems engineers, research engineers, and systems architects. Additionally, the team is currently seeking interns in the fields of data science and computer vision. Klaas Mussche, the main person responsible for my internship, has been working at the company for 11 years as a system and software engineer in various fields. Initially, he worked on performance modeling and simulation, software architecture, data analysis, machine learning, and gun fire control algorithms. Hans de Groot is the most experienced member of the team, with 40 years of active work at Thales Nederland. He is a systems architect who primarily conducts research on image and signal processing, as well as computer vision, related to vessel systems. Guido Gosselink has been working for Thales Nederland for 27 years, and he is the system architect of the Gatekeeper system. He has experience with performance analysis and calculations, electro-optical system design, image enhancement and detection

algorithms. I will be interning with Joshua Koopmans, a fellow computer science student who, like me, shares a keen interest in data science. He will work on a similarity measure of features vector generated by a neural network encoder in order to ensure the continuity of the current tracking system. His assignment can be seen as the computation of a signature for one vessel, which allows the re-identification of the same vessel on the tracking system when the tracking has been cut for specific reasons.

The Gatekeeper is an electrical-optical system that mainly represents the passive surveillance product family, providing advanced security capabilities. The system includes non-rotating infrared/TV cameras that provide a seamless 360° panoramic view of the ship's surroundings. These cameras are equipped with robust tracking capabilities that enable the detection and monitoring of even the smallest surface targets. Gatekeeper is designed specifically as an automatic close-range security sensor for ships. It proves invaluable during harbor operations, while anchored, or when navigating near unfriendly shores. The system is designed to seamlessly integrate with existing platforms as a clip-on device. The system provides 24-hour surveillance, allowing the crew to be deployed more efficiently since they can stay inside a protected and air-conditioned environment instead of having to stand watch on deck. A Gatekeeper system typically consists of up to four sensor units, each equipped with three non-cooled infrared and three TV cameras. Together, they provide a combined view of 96° in azimuth. The 360-degree surveillance image can be accessed from multiple operator stations. Each operator has their own "software-controlled window" to amplify track data and zoom in without disrupting the panoramic view for others at the bridge, operations room, or watch stand. The panoramic view can also be displayed on large screens in the Combat Information Center.

Figure 1: Gatekeeper system

III. Project development

1. Development environment

1.1. Computer systems

In this section, we will describe the development environment used by all developers at Thales Nederland, and potentially on a more global scale. The general system architecture for the computer systems uses Citrix Workspace. This software allows for complete remote control of your computer session on various operating systems, including Windows and Linux. When booting computers, the Citrix software will prompt your login credentials. Once authenticated, users can select the virtual machine they wish to run. None of the virtual machines are connected to the internet. To retrieve information from the internet, users must first run a Windows virtual machine. Then, run a green context that is completely separate from the base context (also known as the blue context). Finally, they can access the necessary information. Downloading files is only possible in the green zone. To transfer downloaded files from the green zone to the blue zone, users have to wait until the antivirus scan finishes. By working in this manner, all files downloaded from the internet are scanned by the antivirus software. If the green context is infected, the probability of the blue context being infected is lower than under normal conditions. The term "green zone" can also refer to a demilitarized zone (DMZ). This environment greatly enhances the global security of systems, but it can also be burdensome for developers and increase the time required for simple tasks. The decision appears to have been made to prioritize safety over productivity, which is understandable given the confidential nature of the work being carried out at Thales.

1.2. Languages and framework

To complete the given assignment, we can choose between Python and Matlab as programming languages for research and production purposes. I chose Python because of its rapid prototyping capabilities. The language is also a scripting language, which makes it more suitable for deploying scripts on remote machines such as the GPU station. It is also the language with which I am most experienced. With the largest community in data science, it is easy to find numerous references for neural networks created in Python. In the associated ecosystem, there exists many framework for deep learning, the mains are : PyTorch from Meta, TensorFlow [1] from Google which is based on Keras, a library created by the French François Chollet, Caffe created by Berkeley AI Research, Theano from the University of

Montreal and more recently the Jax framework from Google which exploits the power of XLA (Accelerated Linear Algebra), etc. I have experience with PyTorch, TensorFlow and a few of Jax but my greater familiarity with TensorFlow leads me to choose it. When it comes to developing neural architectures, PyTorch offers more flexibility than TensorFlow. The eager execution graph sometimes makes the development of architecture difficult and does not allow for easy experimentation with architecture features. An example of this issue is that TensorFlow does not allow the application of a strided and dilated convolution, which can result in the loss of some features. In contrast, PyTorch allows for this type of convolution. However, TensorFlow is much more optimized and better suited for production purposes. The training phase on a GPU is clearly faster than in PyTorch, enabling faster testing of multiple architectures. Concerning Jax, it is more quicker than the previous frameworks, but it lack high-level layers because it is an XLA rewrite of the Numpy API. We can still discuss Flax, the deep learning framework built on top of Jax, but its development stage is too early for it to be used for production nor research.

2. Project requirements

2.1. Targeted data

The task of image semantic segmentation is well-known in the field of autonomous driving. Understanding the surrounding environment can be even more challenging, especially when it comes to navigating the roads. Multiple elements can be depicted, such as vegetation, vehicles, roads and intersections, road signs and traffic lights, people and animals, etc. All of these objects have varying characteristics and are highly dynamic: a significant amount of noise can be found in the information. It is necessary to use robust methods that are not sensitive to outliers and noisy data.

The first point is about the nature of the data being targeted. In our case, the data is not as diverse as in the autonomous driving field, which can make the task easier. The maritime environment is specific and highly similar. Sky and water textures always look the same and belong to a cluster with a small standard deviation. Lands are more noisy, but the colors are often identical. Vehicles share the same macro features, etc. The maritime environment is not making it difficult, thanks to the nature of targeted data.

The second point is about the availability of the data. Public maritime datasets for semantic segmentation tasks, with authorized commercial usage and containing the necessary semantic labels, are not easily available and don't run around. This will be documented in the datasets section of this report.

2.2. Semantic labels

The selection of semantic labels can impose limitations on the public dataset we intend to utilize. If a specific label is missing, we will need to consider another strategy. We will also need to consider the distribution of classes to avoid extreme imbalance problems, even if we correct it later with appropriate class weighting. A basis for the labels should be: sky, water, vehicles, land, others and an ignore class. Sky and water are relatively easy to define, but the others can be a bit more confusing. We will define the semantic labels in the following section.

The "vehicles" label should contain all transportation objects created by humans. By this, we mean boats, airplanes, cars, trains, buses, bicycles, motorcycles, etc. The "land" label has to include vegetation, ground, dirt, sand, rock, houses, buildings, and everything that is fixed on the land. The "others" label will mainly include small object classes that do not belong to the previous labels. For example, both people and animals will fit into the "others" category. Finally, the "ignore" class is a special label that is excluded from the calculation of the loss score. Excluded in this case means that the ignored pixel loss is constantly equal to zero, and the total number of considered pixels is reduced by one for each ignored pixel. Some datasets contain the ignore class because they have missing properly labeled regions or because the image is not fully labeled. We also need to incorporate this special label.

3. Theorical study of the state-of-the-art architectures

3.1. U-Net, SegNet and ENET

In this theoretical section, we will introduce the state-of-the-art in image semantic segmentation. We will try to summarize and synthesize the previously done work by the data science researchers.

To begin this summary, I will introduce the famous U-Net [2] neural network. The U-Net is a famous architecture when it comes to image segmentation. Normally used for binary pixel-wise classification, this architecture has been at the forefront of the image segmentation field for a long time. The concept of U-Net is essentially simple as it utilizes the encoder-decoder technique. The encoder part will reduce the dimension of our signal in order to filter out noise and useless information for loss optimization. Once the dimension is reduced, a process known as downsampling, the output of the encoder is fed into the input of the decoder. The role of the decoder is to increase the dimension of the signal, a process also

referred to as upsampling. The novelty in U-Net is its skip connections, also known as residual blocks. After each downsampling step, we store the output to use it in the associated upsampling part. For example, we down sample an image from "1/1" to "1/2", then we store the result. When we upsample the "1/2" image to a "1/1", we will do it using an upsampling technique. Additionally, we will concatenate the previously corresponding stored result along the depth channels dimension. This allows our neural network to retain the original image information. One important issue with this technique is that low-scale and low-resolution details are easily filtered by the encoder. As a result, these details are not considered in the final segmentation map. The following figure illustrates the U-Net architecture. This

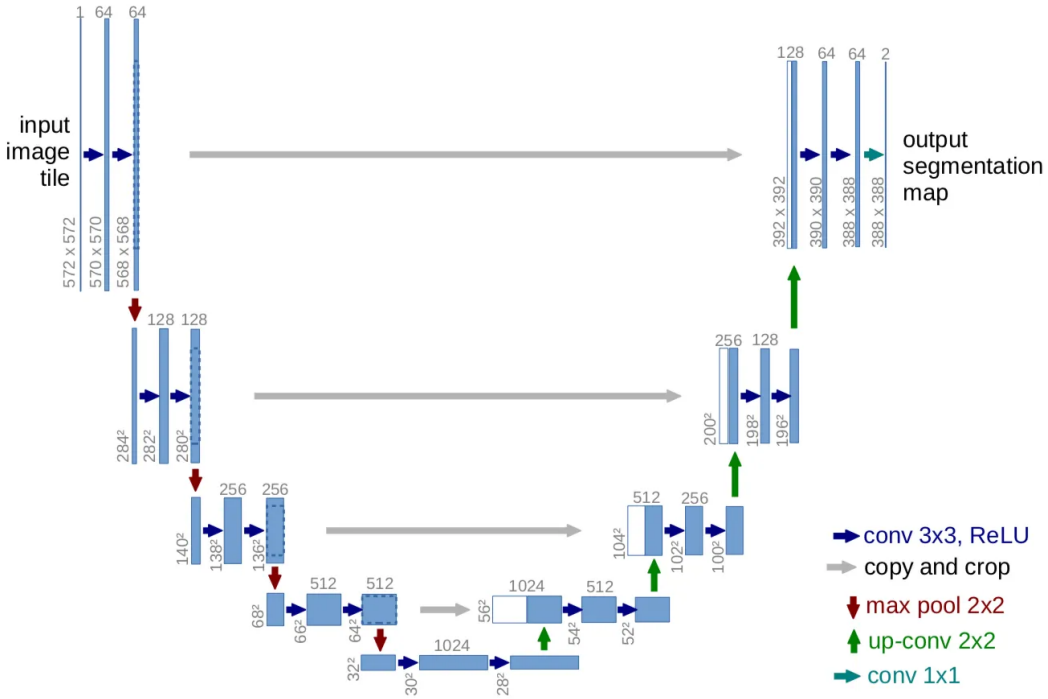

Figure 2: U-Net architecture

architecture is quite complex, with over 35 million parameters. The generation time of the semantic map can be close to one second, which makes it very difficult to apply in real time. We will be able to streamline the architecture by eliminating "1/16" of the layers and transferring its job responsibility to the "1/8" layers. We can also perform only one convolution in the horizontal pass instead of two. The U-Net architecture will serve as the foundation for the state-of-the-art methods we will introduce below and it is also the first architecture I will try during the internship, using the MobileNetV2 backbones to achieve the best convergence with our loss function. There are some available improvements to this architecture, such as U-Net++ [3] and U-Net3+ [4]. The first one focuses on the skip

connection multi-scale functionalities, aiming to reduce the semantic gap between the feature maps of the encoder and decoder sub-networks. The latter, U-Net3+, is an improvement of U-Net++ and claims to achieve full-scale skip connections and deep supervisions.

The SegNet [5] architecture uses the principle of an encoder-decoder network. Instead of using skip connections of previously downsampled feature maps like in U-Net, this method utilizes the max pooling indices generated during the downsampling process. The decoder will receive the indices and upsample the feature map based on the previous layer and these indices. The novelty of this architecture is that it demonstrates the possibility of combining spatial information and color intensity information within the same architecture, even when the dimensions are not homogeneous. This network architecture will not be tested during our internship because it has too many significant differences from the U-Net architecture.

The ENet [6] architecture is also based on the principle of an encoder-decoder network, but it introduces a new kind of block that improves the results of image segmentation. This time, the skip connection was removed to prioritize the layers of the encoder and decoder. The authors created two novel blocks: the initial ENet block, which is used only once as the first block of the architecture, and the bottleneck block, which is used at each layer of the structure. The initial block performs two parallel operations: a 2-stride 3x3 convolution

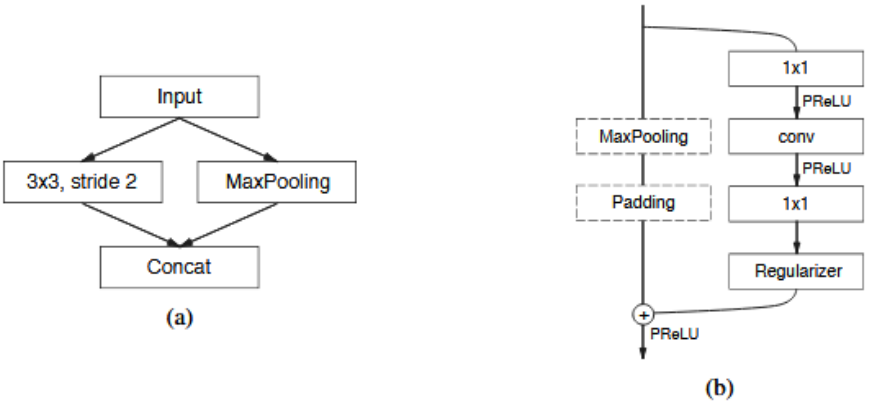

Figure 3: ENet blocks : (a) Initial block, (b) Bottleneck

and a 2x2 max pooling. The results of both operations are then concatenated along the channel depth. This allows us to benefit from both the max pooling layer and the strided convolution. The downsampling bottleneck block is a two-branch block. The first branch performs a max pooling operation followed by a padding operation. On the other branch, we can see three convolutions with different kernel sizes. The bottleneck block can be adapted to be used as an upsampling layer or a downsampling one. The ENet won't be tested in

practice, but it has provided important information about creating new specific blocks to enhance a current model.

3.2. RegSeg

When discussing novel architectural blocks, it is important to note the D-Block from the RegSeg [7] architecture. The D-Block utilizes the power of dilated-grouped convolution for image segmentation tasks. Grouped convolutions are also useful for parallelizing the computation of different convolutions, as they do not depend on the output of another convolution computed within. The use of dilation in image segmentation is important for expanding the field of view of the neural network, thereby increasing the distance of local neighbourhood definition. The D-Block will be used to conduct complementary work on the existing architecture and search if the introduction of it is improving the models or not.

3.3. DDRNet and PIDNet

We previously discussed the filtering issue with the single path encoder-decoder. The low resolution/scale details are not correctly retained and are filtered out by the encoder. To address this issue, advancements in the field have introduced two-path architecture solutions, including Fast-SCNN [8], BiSeNet [9], and DDRNet [10]. The first path is the same as before : it is composed of the encoder-decoder part of the neural network. The lowest resolution can reach a size of 1/64 or 1/32, depending on the backbone we use. But on the other hand, the second approach maintains a higher resolution to preserve the low resolution details without filtering them out. Adding a new branch also creates a new need to merge these branches. Different merging techniques are used in BiSeNet and Fast-SCNN, but I will focus on DDRNet because it has outperformed the state-of-the-art results and serves as the basis for the PIDNet architecture, which will be introduced in the next paragraph. The DDRNet is a two-path architecture with intermediate merging layers and a deep aggregation pyramid pooling module (DAPPM), as shown in the figure [4]. The upper branch has a higher resolution, while the lower branch has a lower resolution. We can see that at the end of each residuals block, we merge the information from the lowest resolution with the highest resolution and vice versa. The intermediate layer merging technique is called bilateral fusion. It bilinearly resizes the lower resolution and performs stride convolution for the higher resolution in order to combine both signals. The DAPPM block performs multiple parallel convolutions with different strides and then concatenates them after applying a 1x1 convolution. This allows the architecture to have a better field of view of the depth channels

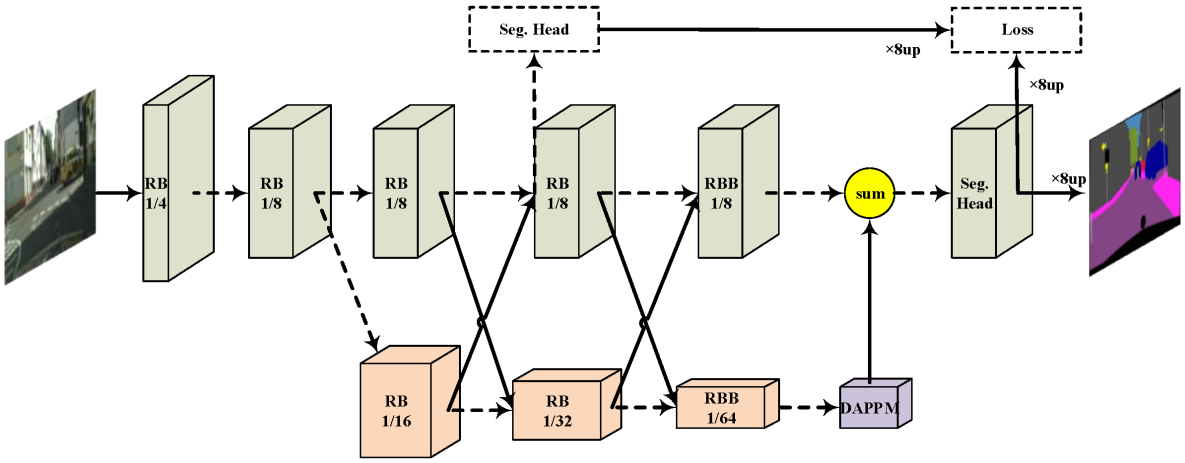

Figure 4: DDRNet architecture

in our lowest resolution block. Finally, the DAPPM output and the higher resolution output are summed before being input into the segmentation head. The branch merging technique used here is simply a summation. The DDRNet will be tested during my internship because it is fast, efficient, and also preserves low-resolution details.

The state-of-the-art (March 2023) architecture for the real-time segmentation task is the PIDNet one [11]. It is based on the Proportional, Integral, and Derivative controllers in automatics. To summarize this controller, we have a first transfer function that is proportional to the error of the feedback control loop and also corresponds to the proportional part (P) of the controller. We have a derivative (D) part of the controller which is deriving the error of the feedback control loop in order to anticipate the future error, it has a predictive effect. The last one is the integrative part (I), where the previous errors are integrated through time. This can introduce a delay that can be compensated with the different weights. We finally sum the three branches with different pondering and then obtain the final PID controller. The work done by the Texas A&M University team is to recreate this behavior in a neural network fashion. The architecture is composed of three branches: proportional, integrate, and derivate branches. To achieve their goal, they use different layer modules: the Pag modules, which take information from the integrative branch to put it into the proportional branch, the PAPPM modules, which are modules that fast aggregate the context of the data by using multiple parallel average pooling layers of increasing sizes followed by a simple convolution 1x1 and a bilinear up sampling layer. All these parallel layers are concatenated and then summed with the input data. Another module used is the bag module, which is a module that merges the three branches. The PIDNet will be tested and we will implement it in Google Tensorflow using a pre-trained backbone which will be

the integral branch. This architecture is also the only one which focus on having a similar image gradient between the predicted output of the neural network and the input image thank to its associated derivative branch and boundary loss function. This boundary loss

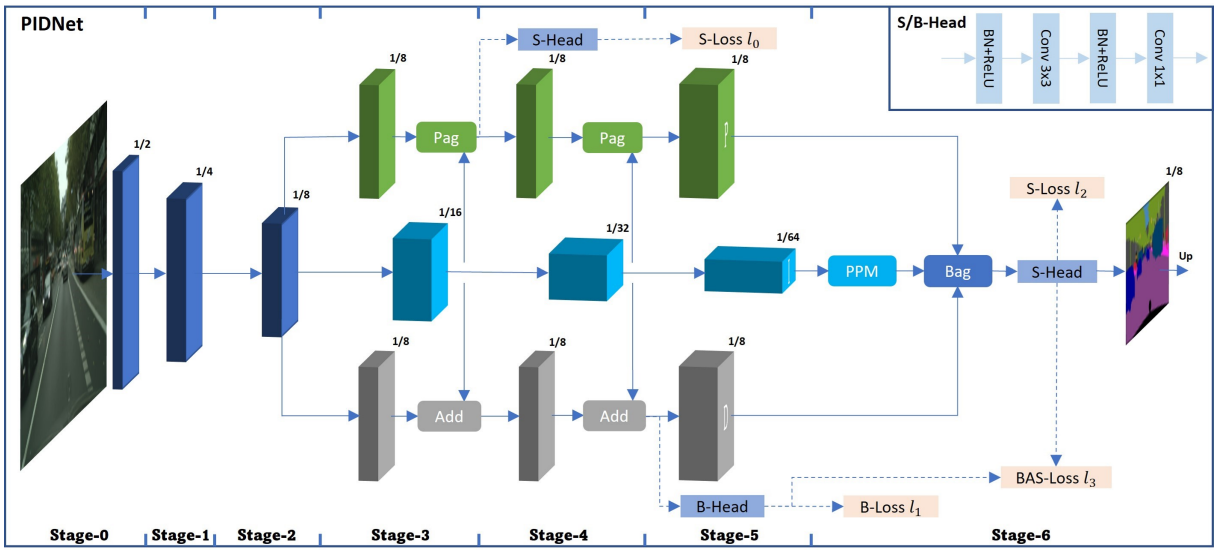

Figure 5: PIDNet architecture

function for the PIDNet is a customized one. It is a weighted summation of four different losses where some of them are edges-aware. The PIDNet loss function is described in the following equation [6].

$$l = w_0 l_0 + w_1 l_1 + w_2 l_2 + w_3 l_3 \tag{6}$$

Where the weights are : $(w_i)_{i\in\{0,1,2,3\}} = (0.4, 20, 1, 1)$. The loss l_0 corresponds to a basic cross entropy loss but applied to the middle of the proportional branch. The l_1 function is a binary cross entropy loss function between the derivative branch and the gradient of the predicted mask. The gradient here is computed with the Canny filter followed by a morphological dilation transformation. The l_2 is a categorical cross entropy between the output of the model and the ground-truth mask. Finally, l_3 is also a categorical cross entropy but some pixel-wise values are masked conditionally. The condition is simply a threshold based on the output of the derivative branch. It means that l_2 and l_3 are quite similar but the regions where the estimated gradient is high are more pondered than others regions.

3.4. Vision Transformers (ViT)

Now that we have reviewed most of the recent convolutional neural network (CNN) architectures, we will shortly talk about the vision transformers [12]. With the current famous trend around GPT models, transformers architectures prooved their efficiency with

huge amount of data. The computer vision equivalent of NLP transformers are quite new but promising. Traditionally, convolutional neural networks have been the dominant architecture for image classification and other computer vision tasks. However, ViT presents an alternative approach by using self-attention mechanisms from Transformers instead of CNN layers. The key idea behind ViT is to treat an image as a sequence of patches, where each patch is considered as a token. These tokens are then fed into a Transformer model, consisting of multiple layers of self-attention and feed-forward neural networks. The self-attention mechanism enables the model to capture relationships between different patches, allowing for global context understanding. The following figure [6] is describing how a ViT works. These architectures won't be used during my internship because transformers need a lot more data to train and we will not be able to find this huge amount of annotated data for the maritime environment. Additionally, the computational cost can be too high.

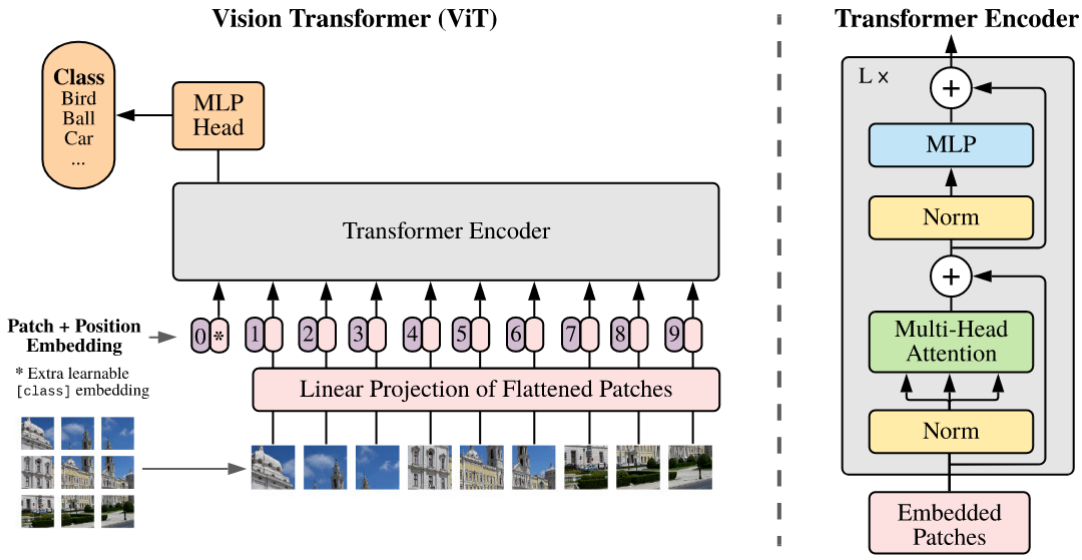

Figure 6: ViT architecture

3.5. Focus on the complement cross-entropy loss

All the previous architectures uses a default loss function such as the cross-entropy. Except the PIDNet one, which includes a composite of four loss functions as previously

described. The categorical pixel-wise cross entropy is defined in equation [7].

$$\frac{1}{N_p} \sum_{(k,c) \in P \times C} -\mathbb{1}_{(y_k=c)} \log(p_{k,c}) \tag{7}$$

We define $P = [\![0, N_p[\![$ and $C = [\![0, N_c[\![$ respectively as the sets of the indices of pixels and classes. Both N_c and N_p respectively correspond to the number of classes and the number of pixels. $\forall k \in P$, the value y_k is the ground-truth class for the k^{th} pixel and belongs to the set of the classes values. Additionally, $\forall c \in C$, the value $p_{k,c}$ corresponds to the probability for the k^{th} pixel to belong to the class c. As is it a probability, the value $p_{k,c}$ is in $[0, 1]$. Our problem is assigning an unique class for each pixel, we can fix y_k and the equation $\forall c \in C,\ \mathbb{1}_{(y_k=c)} = 1$ is giving an unique solution for c. We can simplify the previous sum in the equation [7] to the following.

$$\frac{1}{N_p} \sum_{k \in P} -\log(p_{k,y_k}) \tag{8}$$

On a element-wise basis, each pixel loss is equal to minus the logarithm of the probability to this pixel to belong to the ground-truth class. Intuitively we could ask : why should the probabilities of a pixel to belong to the wrong classes are not considered in the loss score ? There are many reasons for that and we will explore in this section why is it unnecessary to consider these probabilities. First of all, this question have been already partly asked in the paper [13] and has been partly answered with a new loss function that is called complement cross-entropy. The principle is simple, rather than adding zero to the loss function when the predicted probability to belong to the wrong class is high, we adjusted it by increasing the loss depending how high is the predicted probability of the wrong classification. We define the complement cross-entropy function in the following.

$$\frac{1}{N_p} \sum_{(k,c) \in P \times C} f_c(p_{k,c}, y_k) \tag{9}$$

$$f_c(p_{k,c}, y_k) = \begin{cases} -\log(p_{k,c}) & \text{if } y_k = c \\ \dfrac{-\log(1 - p_{k,c})}{N_c - 1} & \text{else} \end{cases} \tag{10}$$

From now on, we search to maximize the probability for a pixel to belong to the ground-truth class, and we also search to minimize the probability - by using the complement probability - for a pixel to belong to the wrong class. The scale factor $1/(N_c - 1)$ is here to reduce the weights of the wrong classes as we have only one correct prediction and $N_c - 1$ wrong

predictions. To interpret this, when the probability of the right class is getting higher, the loss score will go to lower values. Antagonistically, when the probability of the wrong class is getting lower, the loss score goes to lower values.

The idea about the complement cross-entropy loss comes from a gentle mind but to full fill correctly the reasoning, we need to understand how the probabilities are computed and make a fast gradient analysis. In general, neural networks return in the last layer what TensorFlow calls "logits", it is simply unbounded values in $\mathbb{R}$. In theory, these values are unbounded but in practical, they are bounded due to numerical limitations. To obtain a probability for these logits, the softmax function is needed and applied along the channel depth dimension. For a vector of N values, $\mathbf{r} = (r_i)_{i \in [\![1,N]\!]} \in \mathbb{R}^N$, we can compute the corresponding probability vector using the softmax function.

$$\hat{\mathbf{r}} = \left(\frac{\exp(r_i)}{\sum_{j=1}^{N} \exp(r_j)} \right)_{i \in [\![1,N]\!]} \in [0,1]^N \quad \wedge \quad \sum_{\hat{r_i} \in \hat{\mathbf{r}}} \hat{r_i} = 1 \tag{11}$$

$\forall k, c \in P \times C$, we can apply this concept to compute the value $p_{k,c}$. We consider the output of the last layer of our neural network defined for each pixel as :

$$\mathbf{r_k} := (r_{k,c})_{c \in C} \tag{12}$$

This corresponds to the logits that are returned for one pixel k along the channel dimension, we apply to it the softmax function and we obtain :

$$\hat{\mathbf{r_k}} = (p_{k,c})_{c \in C} = \left(\frac{\exp(r_{k,c})}{\sum\limits_{j \in C} \exp(r_{k,j})} \right)_{c \in C} \tag{13}$$

By analyzing this equation, we can understand that if the value of a logit $r_{k,c}$ increases a lot, normalization will also decreases all others values. It means that for each pixel, there is an equivalence between increasing a specific probability and decreasing all the others probabilities. So to relate with the previous question, during our optimization, if we maximize the probability for a pixel to belong to the ground-truth class, then it is strictly equivalent to minimize the probabilities for a pixel to belong the wrong classes thanks to the softmax function. This thought eliminates the need to independently maximize and minimize probabilities and makes unnecessary the complement cross-entropy function. With a gradient analysis, by rewriting the normal cross-entropy loss function and derive it according to

logits:

$$l = \frac{1}{N_p} \sum_{k \in P} -\log \left(\frac{\exp\left(r_{k,y_k}\right)}{\sum\limits_{j \in C} \exp\left(r_{k,j}\right)} \right) \tag{14}$$

$$\begin{aligned} \forall (k,i) \in P \times C, \quad \frac{\partial l}{\partial r_{k,i}} &= -\frac{1}{N_p} \frac{\partial p_{k,y_k}}{\partial r_{k,i}} \frac{1}{p_{k,y_k}} \\ &= -\frac{1}{N_p} \frac{1}{p_{k,y_k}} \begin{cases} p_{k,i}(1 - p_{k,i}) & \text{if } i = y_k \\ -p_{k,i} \cdot p_{k,y_k} & \text{else} \end{cases} \\ &= \frac{1}{N_p} \begin{cases} p_{k,i} - 1 & \text{if } i = y_k \\ p_{k,i} & \text{else} \end{cases} \end{aligned} \tag{15}$$

We can see that, with the normal cross-entropy, the model weights are updating even for the misclassification logits. In other words, the else part of the bracket is not null. It is likely that the complement's cross-entropy only emphasizes this default behavior when the softmax function is used to compute the probabilities. Thanks to Guillaume Bourmaud, associate professor at Enseirb - Matmeca and from the IMS laboratory at Bordeaux who lightened me on this.

4. Datasets

4.1. Available datasets

The current section intends to describe the datasets used. First, we will check about the licence of these semantic segmentation public datasets.

Dataset name	Licence	Images context
ImageNet-S	Non-commercial use	General
COCO-Stuff	Creative Commons	General
Cityscapes	Non-commercial use	Urban streets
MassMIND	Non-commercial use	Maritime infrared
MaSTr1325	Publicly available	Maritime
Atlantis	Creative Commons	Water and maritime
MarSyn	Not found	Maritime
Tempere-WaterSeg	Creative Commons	Maritime
ROSEBUD	Non-commercial use	Maritime
ShipSG	Non-commercial use	Maritime

As expected, a lot of datasets are for research purpose only and cannot be used to train an algorithm which will be a released commercial product.

4.2. Atlantis, MaSTr1325 and MarSyn

Among the above datasets, we will focus on Atlantis, MaSTr1325 and MarSyn. The choice have been made because these datasets are in maritimne environment, have at least the requested semantic labels and their licence is allowing the commercial use. Let's introduce them one by one.

Atlantis for ArTificiaL And Natural waTer-bodIes dataSet [14], is a dataset made by the University of South Carolina in Columbia, SC about water bodies. This dataset have 56 semantic classes over 5,195 images of different sizes. It is powerful in its ability to learn the concept of aquatic objects. The dataset is not only about maritime environment but provides more general images which depicts water elements on it. This dataset will be the base dataset for all my work. It also has an ignoring class for unlabeled regions.

MaSTr1325 [15] is a 4 classes dataset in the maritime environment. The 1325 images are taken on a small USV boat and depict the coast border vision. All the data has the same width and height. It is covering a range of realistic conditions encountered in a coastal surveillance task. The dataset also has a specific class to ignore unlabeled regions.

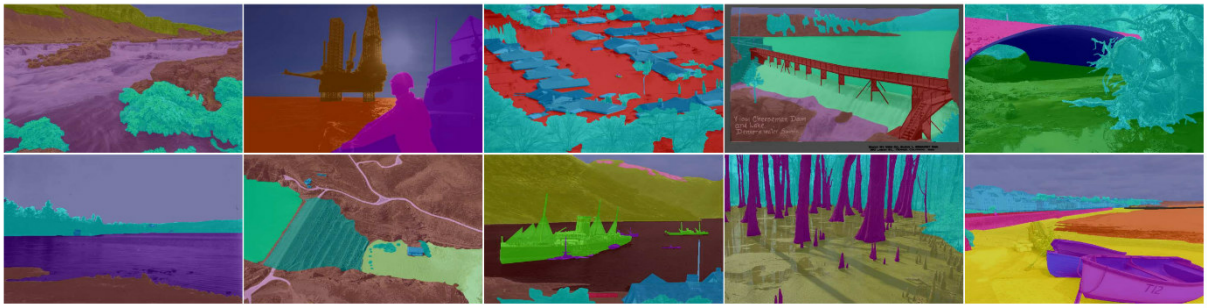

Figure 7: Sample of Atlantis dataset

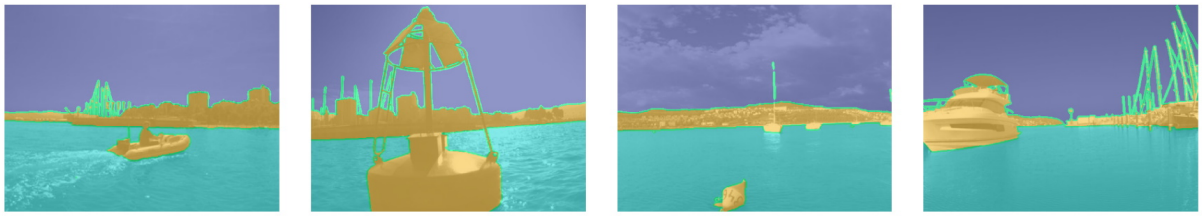

Figure 8: Sample of MaSTr1325 dataset

MarSyn [16] is a synthetic binary classes dataset in the maritime environment and aims for the instance segmentation task. It is made up of 25 various Blender-produced photorealistic video sequences, each with 1000 frames. Data sets users have the choice between HD resolution (1280x720) or a lowered and squared one (550×550). Synthetic dataset can be interesting but we need to be aware of the ability of our algorithm to overfit the water and sky texture. In the provided data, these textures are redundant.

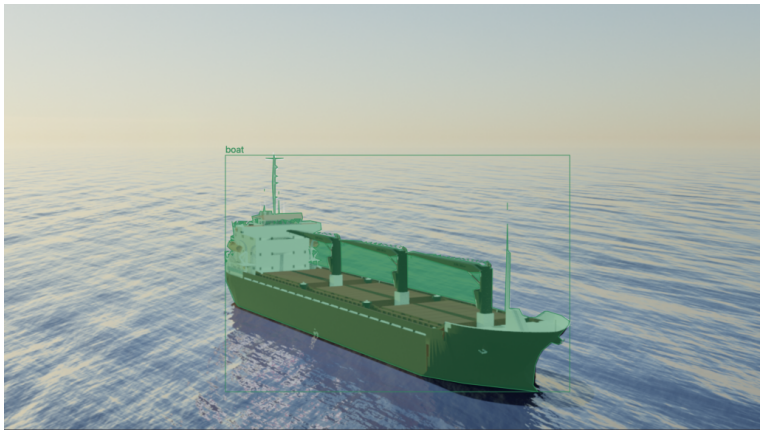

Figure 9: Sample of MarSyn dataset

4.3. COCO Stuff

The COCO Stuff dataset [17] is an extension of the popular COCO dataset that focuses on image segmentation tasks. It consists of over 164k images, which are annotated with pixel-level labels for more than 91 stuff classes such as sky, grass, road, and buildings. COCO-Stuff has 172 classes: 80 things, 91 stuff, and 1 unnamed class. The 80 things classes are identical to those in COCO. The dataset is very general and cross over multiple kind of photos. Images sizes differ from image to image. We will use this dataset, not to focus on the maritime environment but to aim on more general tasks. By doing this, we want the algorithm to avoid over-fitting on the maritime environment and simply be able to segment general regions.

Figure 10: Sample of COCO Stuff dataset

4.4. Customized : filtered and merged

Image semantic segmentation is a data-hungry task. In order to feed correctly the algorithm during the training phase, we will create a custom dataset which is a filtered merge of the four previous ones. To create this new customized dataset, we will first include the whole Atlantis dataset. The 56 semantic labels will be manually grouped into the 5 labels basis : sky, water, vehicles, land and others. We keep the ignore label which is a special label for unlabeled regions. During the semantic labels grouping, if any label does not fit in the sky, water, vehicles or land labels, it will be defaulted to the “others“ class. We apply exactly the same strategy to the whole MaSTr1325 dataset. For MarSyn, as the dataset is composed by multiple images timeseries, we need to regularly sample the timeserie in order

not to take the 25 000 samples. We will take 1 image for every 15 photographs, totaling 1666 more images in the custom dataset. As the images are frequently zoomed out, the ships appear to be sometimes incredibly little, and the region masks as well. A centered crop will be applied to the MarSyn observations, in order to zoom in on the segmentation mask . For the dataset COCO Stuff, we manually categorize the 172 classes into 5 labels, which is a very time-consuming job. COCO Stuff presents an abundance of samples, we must filter the quantity of photos in order to retain only those that we are interested in. To achieve this, we will compute the classes distribution for each image and apply the following defined criteria to filter them. This criteria is that the proportion of "others" labels (after the labels grouping) must be below 20% for the current analyzed image.

The custom dataset called "ImSeg 5C" is now mostly composed by maritime/water pictures and it contains a few very general images. We have a total of 22698 images. The pixels classes distribution is in the following figure. As planned, the classes land, sky, water

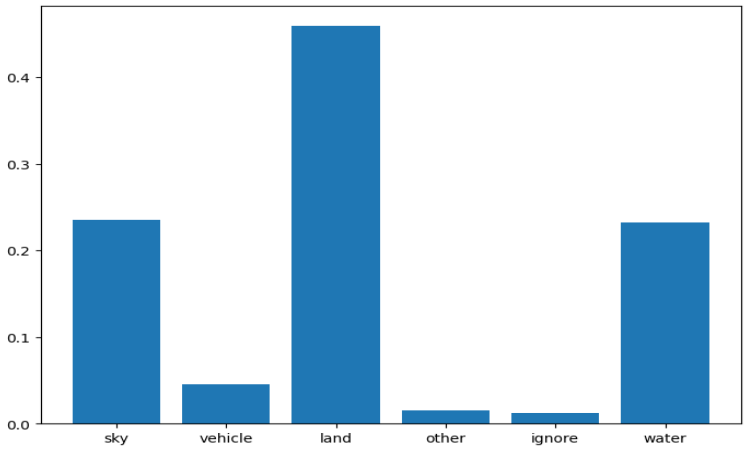

Figure 11: ImSeg 5C classes distribution

are the prevalent labels. We can observe a few proportions of ignored and others, which was the targeted amount. The problem is an unbalanced one but the explanations of the used correction are in the section 5.3. The need to standardize images forces us to apply a general image processing pipeline using the library Albumentations [18]. The first layer of the image processing takes the longest side of our image and resizes it to 448 pixels while maintaining the original image ratio. Then, a padding to 448x448 is applied to the image. Logically, the shortest side of the image will be padded to 448 pixels using the default border mode of OpenCV called "BORDER_REFLECT_101". This avoids having a default ignored value for the padded part, but instead, adds a reflection or mirror of the original image. The processed images are the same size thank to this previously described pipeline.

5. Training phase

5.1. GPU Station

The EO team at Thales Netherlands B.V. has a specialized computer with a strong GPU within it to do the most computationally intensive operations, such as training an algorithm. The computer is accessible in the local network using SSH and the host name of the machine. The local DNS server will resolve the host domain name and redirect it to the machine's actual local IP address. We don't need to know the local IP address, which is useful if the DHCP server updates it frequently. There are two ways to utilize SSH: one is to open a session with its remote interface and begin writing the appropriate commands, and the other is to immediately ask SSH to run a command without the need to open a long live session. We will use the second method because it is faster to build Python scripts and run them on the GPU station. We have a SFTP server with dedicated network bandwidth that is incredibly fast to share datasets between the local machine where I develop and the GPU station. To avoid having to launch a SFTP session every time I develop on my PC, I decided to keep a local copy of all datasets on my personal and local disk. To distinguish between the GPU station and the working computer when running a training script, I investigated the environment variables and looked for a distinction between a development computer and the GPU station. A variable called HOST was keeping the computer's hostname so that I could determine whether the script was running on the GPU station or not. Depending on that, I was able to immediately adjust the location of the datasets in Python. If the script is running on the GPU station, utilize the datasets served by the SFTP server. Otherwise use the local location which has slower and shared bandwidth but is preferable for development purpose.

The GPU station has a professional NVIDIA A5000 with 24 GB of GDDR6 VRAM. This kind of GPU approaches the 3,000€ and seems to be suited for the tasks we do. As we are two interns using this GPU, with Joshua, we had to use the half of the 24 GB of VRAM in order not to run out of memory allocation. For more people training algorithms, more GPU are required. On the GPU station, the CPU is an Intel Xeon W-2245 that goes up to 3.90Ghz. This processor has 8 cores and 16 threads and works with DDR4 RAM. The GPU station has 64GB of this DDR4 memory RAM. We will use CUDA on this machine to increase the computational power and optimize it.

5.2. Real-time metrics and logging

To follow the training phase of our algorithm that started on the GPU station we will use multiple techniques and tools. The first tool is TensorBoard, a tool created for TensorFlow and by its authors. It can be used in PyTorch and other frameworks. TensorBoard will allow us to follow in real-time the metrics of our model. It encodes the metrics in its own binary file format and provides a web server that serves the interface to load and visualize these encoded metrics. The TensorBoard is launched automatically, we need only to configure the proper callback during training loop. One other tool is the "nvidia-smi" command because it shows a summary of the available resources on the Nvidia GPU. To retrieve the failed steps of our training and all the logs, we will use the following commands :

```
$ nohup python3 train.py > ./logs/foreground.logs &

$ tail -f ./logs/foreground.logs
```

The nohup program is made to remove the signal SIGHUP which can be sent to all of our process when we disconnect from our Citrix session. The '>' is used to redirect the standard output in the file './logs/foreground.logs' where the current dot folder should be a shared folder between the GPU station and the working computer. We add a '&' at the end of the line to run the whole command in the foreground. Then, we can tail the log files in real time using the option -f. All the standard errors and outputs should appear in the 'foreground.logs' file.

Our main metrics are the accuracy and the mean intersection over union. We will remind what's the formula of both of them. We can denote for a binary class problem :

$$Acc = \frac{TP}{TP + FN + TN + FP} \quad IoU = \frac{TP}{TP + FP + FN}$$

Where TP, FP, TN, FN is respectively the number of true positive, false positive, true negative and false negative. Thus, we have for a binary class problem $Acc \leq IoU$ and we can consider that the IoU ignores the true negatives count in the total number of elements taken in account. The accurary metric is simply the proportion of predicted pixels classes equal to ground truth pixels classes on the same spatial basis. We can also see it as the trace of the confusion matrix over the sum of all coefficients of the same confusion matrix. The mean intersection over union is the mean over each class (i.e., considered as binary class) of the previously described IoU formula.

5.3. Data pipeline

A long research have been made to find the data pipeline that best fits to the data we use. Partly because of some hyper-parameters can be defined thought this pipe. In this section, we will summarize the final data pipeline that I used during all my training. First, to optimize the computation of the weights update, we need to ensure our image loading system do not load the whole dataset at once but more in the way of lazy iterators which have the advantage of not consuming all the RAM. The original dataset is a string dataset containing images paths. We map each string line of this dataset to a tuple of string which are the paths of the input image and the mask image. With each sample of the 2-tuple string, we are able to load both the training data and the ground-truth regions. Once we load the images, an operation transforming the 2-tuple strings to a 2-tuple arrays, we can apply our data augmentation to both images. Data augmentation is a pipeline which apply random transformation to our training images in order to provide more varied images and to avoid over-fitting. For a simple image, multiple transforms are possible such as the horizontal and vertical flip, image rotation, translation, color correction, blur, noise addition, random crop, etc. We chose a basic pipeline which can be decomposed in a base transform and a training transform pipelines. The base transform will be first applied to train, validation and test sets and the train transform will only be applied in a second time to the training dataset. The base transform will contain a simple 224x224 resizing layer whereas the training transform will contain a random shift, scale and rotation followed by a random horizontal flip. Then, we pad if needed the image and do a random 224x224 crop. Thanks to the library Albumentations to make this step really fast and easy. After the data is augmented, we can compute the classes weights based on the ground-truth masks. The classes weights are used to correct the unbalance issue of the classes distribution. There are many way to do it but we chose to use the most classical method based on the inverse of the appearing classes frequencies. Defining the classes weights in the following :

$$\hat{\mathbf{w}} = (f_k w_k)_{k \in C} = \left(f_k \frac{\sum_{i \in C} n_i}{n_k N_c} \right)_{k \in C} \tag{16}$$

Where N_c is the total number of classes, f_k an adjustment factor and n_k is the number of pixels belonging to the class $k \in C$. To get the values n_k, we need to iterate once through the whole data set, count the number of times each class appears in a single image and sum this for all images. To obtain the sample weighting map, we apply in a vectored fashion to

the ground-truth mask the following function :

$$\begin{aligned} \mathrm{f}\colon C &\to \mathbb{R} \\ c &\mapsto f_c w_c \end{aligned} \tag{17}$$

By default, all adjustment factors are equal to one. This factor will allow users to manually increase or decrease the computed weights.

6. Abstract representations

6.1. Data set and data loader

To make the update of architectures easier, we created an abstract representation of the whole deep-learning pipeline in order to be able to modify and rearrange those quickly. In these sections, we will describe the functionalities of the different abstract representations. The data set class is a wrapper around the TensorFlow data set which allows a string path generator to be transformed into an augmented TensorFlow data set. A tuple of three tensor of the same size will be outputted for each observation : input image, ground-truth mask and sample weight. The transformation is made using an abstract class from the standard library of Python called "abc". A class child from the data set abstract parent class must have the following methods implemented.

Method name	**Argument(s)**
image_path_generator	split
mask_path_concat	path
split_folders	None
labels	None
labels_ignore	None
images_labels_map	im, mask
augmentation_compose_base	None
augmentation_compose_augmented_train	None
augmentation_factor	None
cardinality	None
output_shapes	None
labels_vocabulary_map	None

Table 1: Data set abstract methods

We can create the data sets class by implementing the above methods. With these methods, we defined here the paths data set, the data augmentation transformations, the classes and data mapping, the ignore classes and the tensors output shape. This can be contained in a maximum of 40 lines and will facilitate the way we use the data set such as a plug-and-play tool.

```python
class ImSeg_5C(ImSegDataset):
    def __image_path_generator__(self, split):
        basis_dir = "/ImSegDatasets/ImSeg_5C/" if EnvUtils.isGPU_HOST() else "./datasets/ImSeg_5C/"
        paths = list(pathlib.Path(f"{basis_dir}{split}/Images/").glob('*.jpg'))
        random.seed(0); random.shuffle(paths)
        return paths.__iter__()

    def __mask_path_concat__(self, path):
        res = str(path)
        return res, res.replace("Images", "Masks").replace(".jpg", ".png")

    def __split_folders__(self): return ["train", "test", "val"]

    def __images_labels_map__(self, x, y): return x, y

    def __augmentation_compose_augmented_train__(self):
        return A.Compose([
            A.ShiftScaleRotate(scale_limit=0, shift_limit=0.01, rotate_limit=10, p=0.9,
    border_mode=cv2.BORDER_CONSTANT, value=0, mask_value=self.labels_ignore),
            A.HorizontalFlip(p=0.5),
            A.PadIfNeeded(min_height=SHAPE,min_width=SHAPE, always_apply=True,
mask_value=self.labels_ignore),
            A.RandomCrop(SHAPE,SHAPE, always_apply=True),
        ])
    def __augmentation_compose_base__(self):return A.Compose([A.Resize(SHAPE, SHAPE)])

    def __augmentation_factor__(self): return 3

    def __labels__(self): return [{'name': 'sky', 'id': 0, 'color': [129,236,236]}, {'name':
'water', 'id': 1, 'color': [9,132,227]}, {'name': 'vehicle', 'id': 2, 'color': [20, 250,
20]},{'name': 'land', 'id': 3, 'color': [240, 147, 43]}, {'name': 'other', 'id': 4, 'color':
3*[50]}]

    def __labels_ignore__(self): return 255

    def __cardinality__(self): return {'train': 20428, 'test': 1135, 'val': 1135}

    def __output_shapes__(self): return (SHAPE,SHAPE)

    def __labels_vocabulary_map__(self): return {}
```

Figure 12: ImSeg 5C data set abstract implementation

6.2. Model architecture

A model architecture abstraction is the concatenation of pre-processing layers, a neural network model and post-processing layers. There is few required attributes such as the number of classes, the batch size and the input shape which are required to merge the three layers. The methods are noted in the below table [2]. The called model are Model instances from the Keras module in TensorFlow. We use this prebuilt instance in order to avoid to rewrite the wheel and to use the complete TensorFlow compatible functionalities.

Method name	Argument(s)
get_preprocessing	None
get_model	None
get_labels_shaper	None

Table 2: Model architecture abstract methods

6.3. Model trainer

The model trainer will be the final abstract object explained. It is used to build the entire pipeline, begin training and save the model's signature. The signature enables us to recreate the model if it is lost or rewritten. The model trainer has important responsibilities, we can configure all of the training loop callbacks, the number of epochs, the loss functions and metrics used, the weights sampling method (per batch or on the entire data set), the pre-trained weights loading, the learning rate schedule and which layers in our model architecture are trainable or not. The data set and model architecture are the two main components that are fed into the trainer. These objects can be changed on the fly before a training loop begins in order to tailor or fine-tune the training loop. The built model trainer may identify whether or not the training is being conducted on the GPU station and, if so, alter some aspects to take advantage of the GPU optimization and the SFTP server speed.

7. Results and validation

7.1. UNET

This subsection will spread all the obtained results from the different model architectures. We will validate each trained model on the configuration described in the corresponding section. The first tried architecture is UNET with a pretrained backbone.

We will only train the UNET model on the Atlantis data set and its 56 classes. Each image will be augmented 16 times on each epoch, which corresponds to 1682 steps per epoch because of the batch size of 32, and the training will long 200 epochs. The initial learning rate is 5e-4 and the final learning rate will be 2e-5 using the classical exponential decay schedule. The model architecture uses 11,4M parameters where 2M comes from the MobileNetV2 pretrained backbone that is considered as our features encoder. The 9M remaining weights comes from the decoder part which is the decoder of the UNET. The decoder consists in 2D up sampling layers with a concatenation of the same size skip connection. We use a composite loss function that is defined in the following equation :

$$l = C_e - log \circ M_{IoU} \tag{18}$$

Where C_e is the cross entropy loss function and M_{IoU} is the mean, along the pixels classes, intersection over union loss function. The $-log$ is used because the mean intersection over union belongs for each pixel to the set $]0, 1]$. Finally, the input shape is 224x224x3.

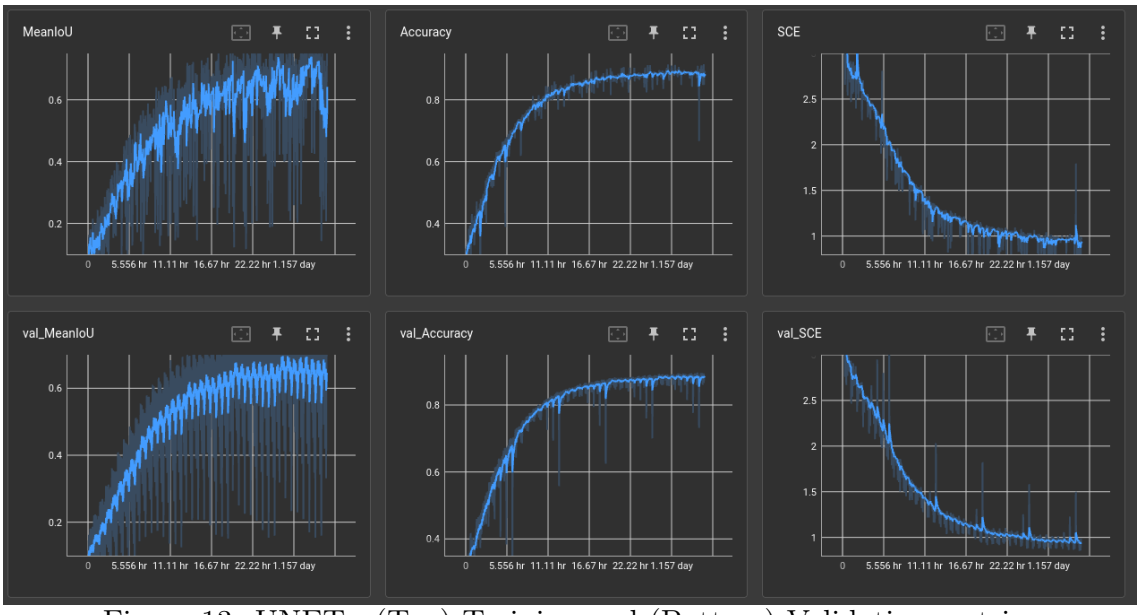

Figure 13: UNET : (Top) Training and (Bottom) Validation metrics

We obtained the metrics showed in figure [13]. The mean intersection over union reaches a smoothed value of 0.63 on both the training set and validation set. The accuracy is 0.88 and 0.87 respectively on the training and validation set. We can observe that the not smoothed metrics are really noisy which can degrade the interpretation of these results. We will predict the masks on few validation sample in order to visualize the accuracy. The

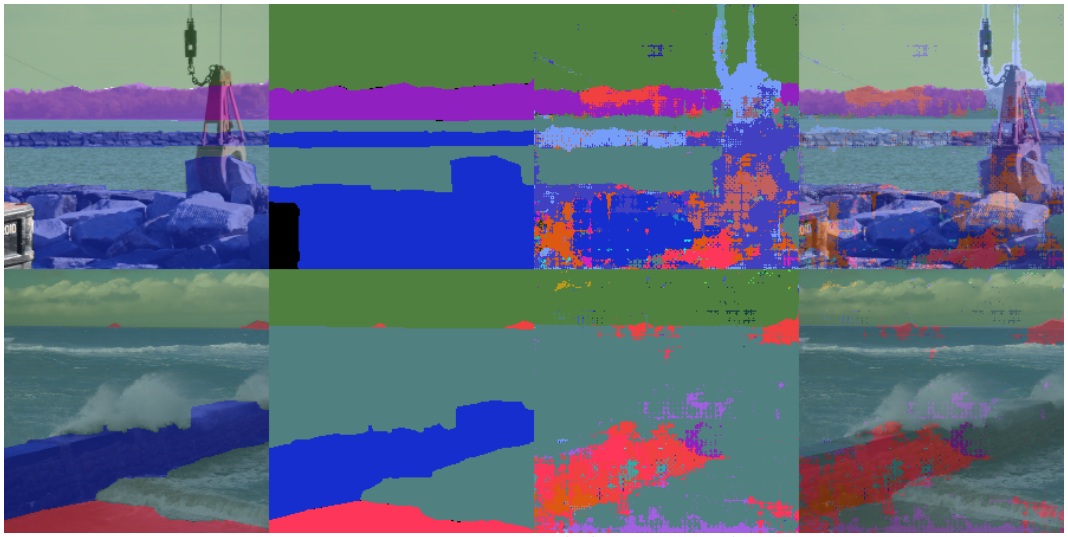

Figure 14: UNET : Validation prediction

Column 1 - Image superposed with the ground-truth mask. Column 2 - Ground-truth mask. Column 3 - Predicted mask. Column 4 - Image superposed with the predicted mask

results are noisy, regions are not convex. We have to improve the outcomes. The current hypothesis for a failure model are that there are too many classes (56), too few examples, too simplistic architecture or in batch samples that are too similar due to augmentation. Investigating these difficulties with several training on different configurations is required and was completed throughout the internship.

7.2. DDRNet

The next tested architecture is the DDRNet one. To remind why this architecture has been invented by researchers, we must remember that the UNET base predictions miss high-resolution details. Researchers added a new branch which is not down sampled like the main branch in order to keep small size details. The configuration is quite similar to the previous one, only the neural network architecture changed. This will allow us to understand if the DDRnet is acting better for this specific task than the UNET as established by the state-of-art papers which have compared both. We observe the given metrics in the figure [15]. These are above the previous architecture because we gained +4% of accuracy and

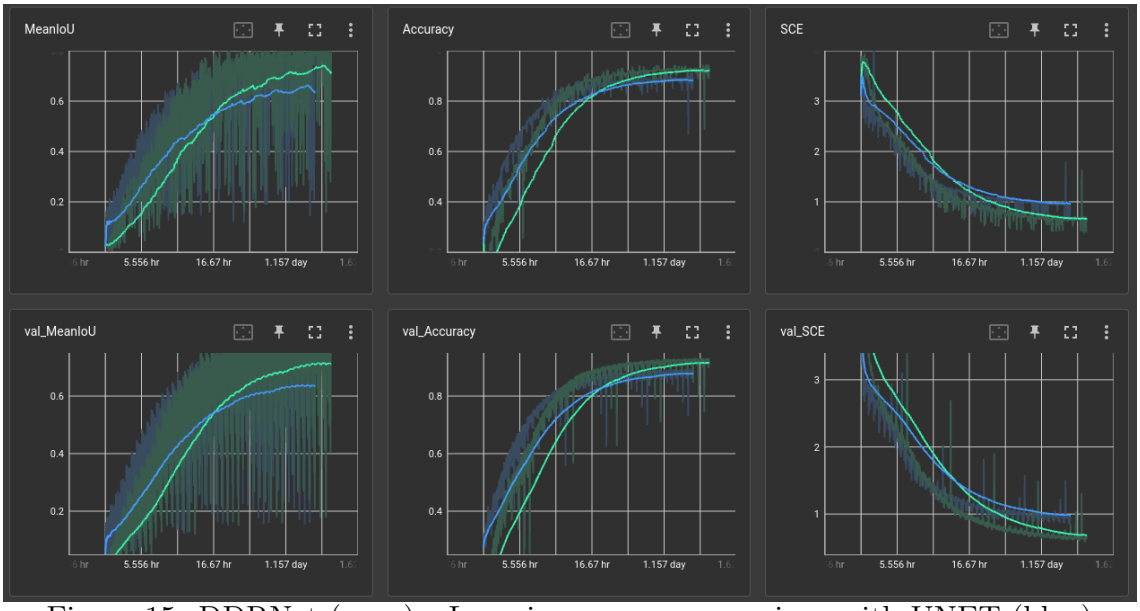

Figure 15: DDRNet (cyan) : Learning curves comparison with UNET (blue)

+8% of mean intersection over union. The following figure [16] is showing how the noise is reduced in comparison with the UNET results. The shapes are more convex but we can still see for regions not of water and sky that the predictions are completely wrong. We will explain how we resolved this issue in the next section.

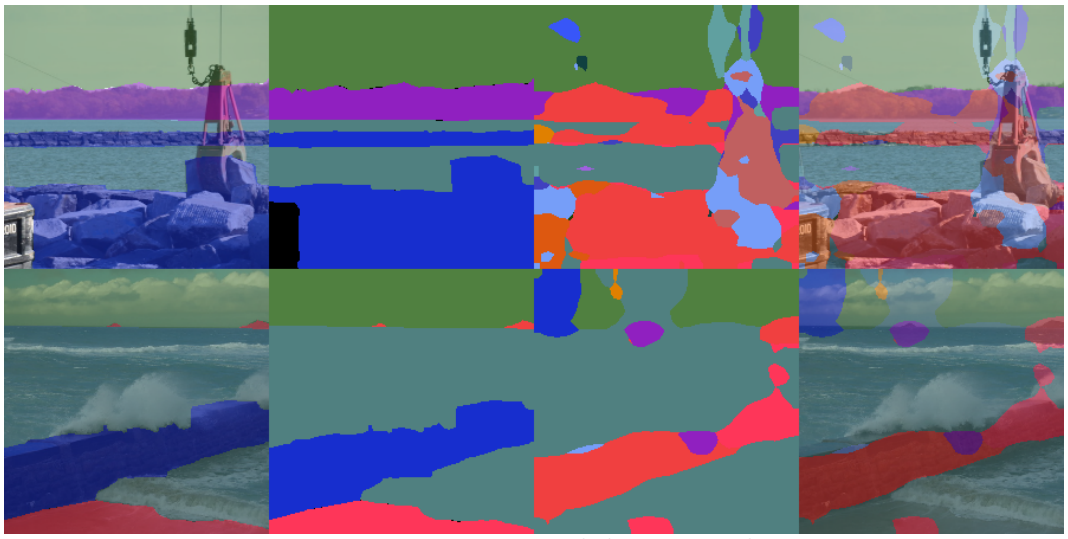

Figure 16: DDRNet : Validation prediction

Column 1 - Image superposed with the ground-truth mask. Column 2 - Ground-truth mask. Column 3 - Predicted mask. Column 4 - Image superposed with the predicted mask

7.3. PIDNet

The PIDNet architecture is the one I gave half of the duration of my internship because it was the most promising. Most architecture and training configuration improvement have been made on this neural network model. Unfortunately, I won't have the space to explain all the paths I took to arrive to the next results but I will express the major improvements and how they were important.

First issue with the previous algorithms was the noise present in the learning curves. This could leads to wrong interpretation of model performance. After multiple search on why this issue occurs, I could find the problem. With the Atlantis data set, images are grouped in folders of the same nature. For example, the two firsts folders are "bridges" and "breakwater" and we could find multiple corresponding images inside it. In parallel, the string path generator for the images was reading the image paths in the order of the folders. This means that when creating the images batches and augmenting their numbers, we could have a sequence of 10 to 20 batches of similar images. Weights updating will occur 10 to 20 times for the same kind of image, with low standard deviation. My intuitive hypothesis is that this behaviour is equivalent to increase the learning rate and prevent the algorithm to finely learns. Other way to see this issue is to consider the algorithm was briefly over-fitting each 10-20 batches, so that next batches sequence was killing the previous over-fit for a new one. I discovered this issue because we could see a cycle with constant frequency on the validation prediction masks where the main predicted class was changing in the same order throughout the training loop. A really simple solution who solved this issue was to collect then shuffle the paths generator in order to read paths in a random order.

Second issue was the too high number of classes for a too few number of sample. Atlantis gave 56 classes for approximately five thousands samples which may not be enough and miss some generalization because of the water bodies specific data. We solved this issue by creating the ImSeg_5C data set described in the section [4.4.]. The number of classes is now reduced to five and the number of samples has been increased to reach the twenty-five thousand samples.

Third issue was the zero edge-awareness nature of the previous architectures. As we used a composite loss function of the cross entropy and the mean intersection over union, we are just optimizing the weights on a pixel-wise base. By introducing a loss function that is edges-aware or boundary-aware, the weights won't be optimized on this pixel-wise basis anymore. Following the reference [19], an example of this kind of loss function can be the following :

$$l = \frac{1}{N_c N_p} \sum_{(c,p)\in C\times P} |\frac{\partial Y_{p,c}}{\partial x} - \frac{\partial Y_{pred,p,c}}{\partial x}| + |\frac{\partial Y_{p,c}}{\partial y} - \frac{\partial Y_{pred,p,c}}{\partial y}| \tag{19}$$

Where $Y_{p,c}$ is the ground-truth binary mask for the class c and pixel p and $Y_{pred,p,c}$ is the predicted binary mask for the class c and the pixel p. This loss function measures the pixel-wise difference of the gradients for each direction. Depending how the gradient is computed, we should now consider our loss function is edges-aware. The PIDNet architecture is solving this issue using a weighted summation of four loss functions described in the associated theoretical study and which is also edges-aware. Finally, after resolving the previous issues, we could improve the results to obtain the following metrics.

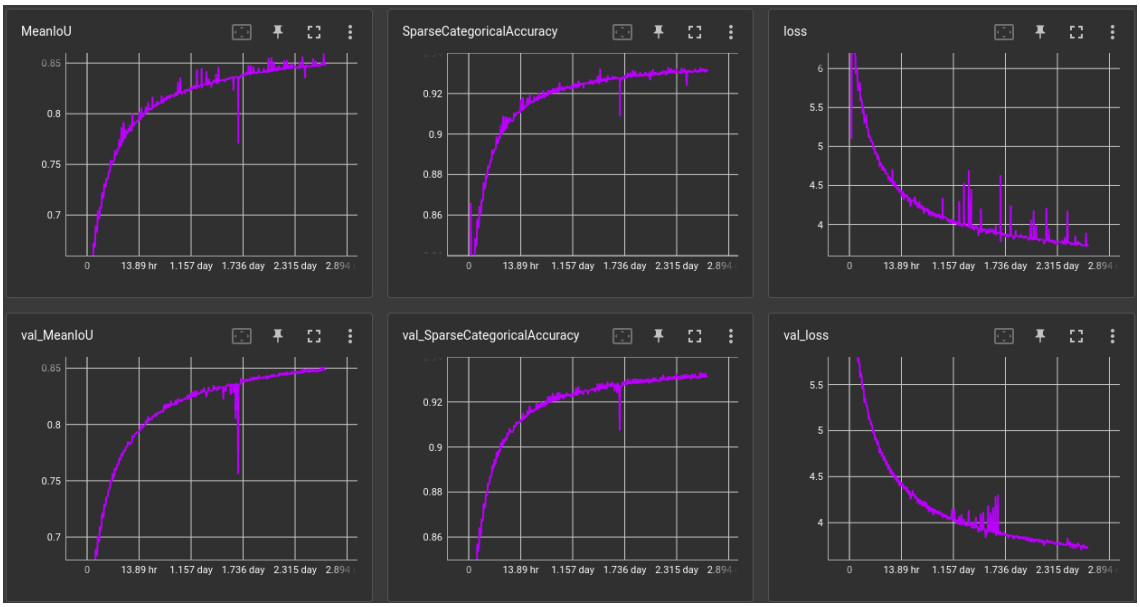

Figure 17: PIDNet: Learning curves

Because of the paths generator shuffle, the learning curves are less noisy than in previous trainings. Figure [18] shows that, despite the relatively near metrics, the PIDNet model trained on the ImSeg 5C performs far better than earlier models. This appears to be logical because the volume and diversity of consumed data is substantially larger. As the variety of the validation set also increases, the metrics for our most recent model become more harsh and must be compared with caution to metrics from earlier models. The boundaries and high resolution details are some of the present model's limitations. The 224x224x3 picture size is down sampled in the architecture to a 3x3xC with $C >> 3$, resulting in a reduction rate of maximum 75.

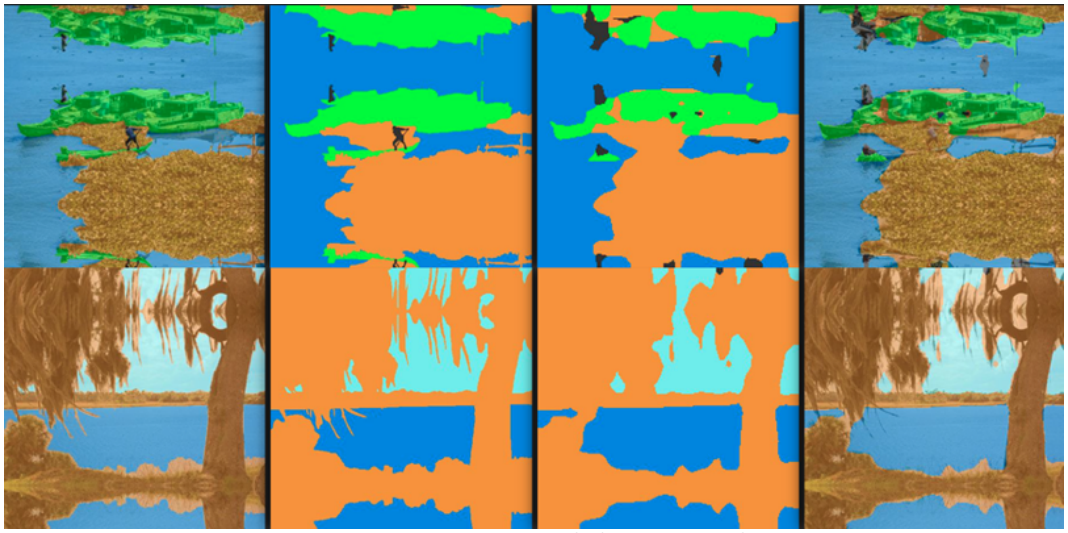

Figure 18: PIDNet : Validation prediction

Column 1 - Image superposed with the ground-truth mask. Column 2 - Ground-truth mask. Column 3 - Predicted mask. Column 4 - Image superposed with the predicted mask

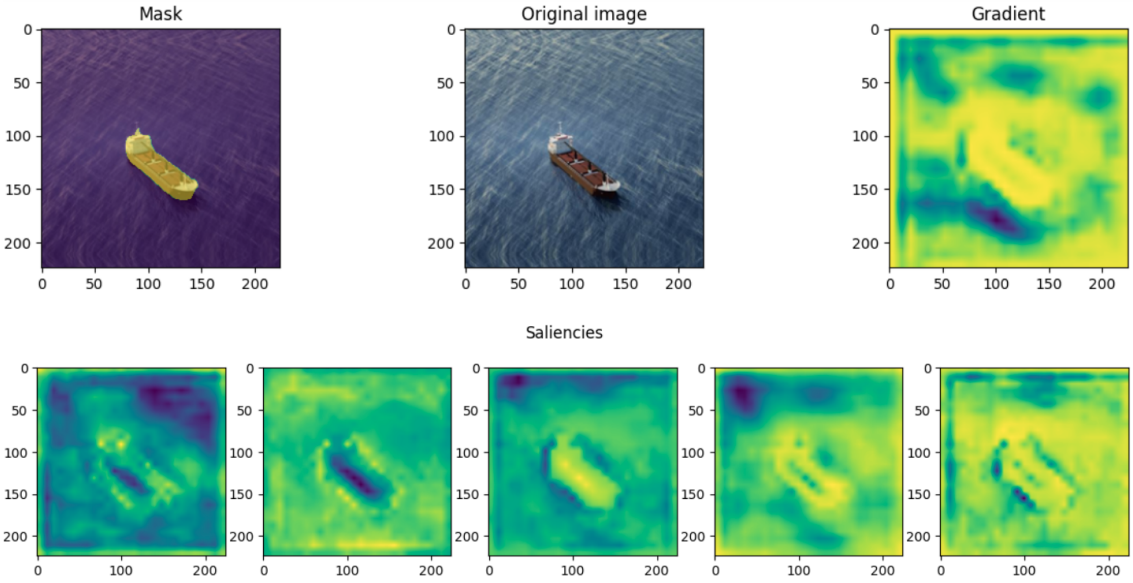

Figure 19: PIDNet: Output example

7.4. Summary

In this part, we will summarize the metrics of our tested models in a table. In actuality, we conducted a lot more training because the GPU was almost operating every day for 6 months. We picked the top models for each architecture and several data sets and summarized them in the table below.

Architecture	**Dataset**	**Val. Accuracy**	**Val. MIoU**
U-Net	Atlantis	0.8787	0.63
DDRNet	Atlantis	0.93	0.71
PIDNet	Atlantis	0.94	0.75
PIDNet	ImSeg 5C	0.94	0.85

Table 3: Trained models resulted metrics

8. Post-processing : Saliency enhancement using edges

The goal of the saliency enhancement is to improve the neural network output utilizing deterministic methodologies. A neural network's raw output in an image segmentation challenge can be C saliencies maps, each with a value in $\mathbb{R}$. We can use the softmax function to convert it to a probability density throughout the channel depth so that the sum over the last channel for each pixel is equal to one and all values are in the range $[0, 1]$. We now have our saliency map for each class, and every time we feed the neural network an input image, we get the C saliencies map. The efficacy of the saliencies map is the key to image semantic segmentation. In order to include the gradient domain information of our input image, we will review few techniques that allows the C saliencies maps to be modified to achieve the goal of better classification results.

8.1. Two passes recursive 1D salient filtering

The reference [20] is giving a special two pass 1D filter to resolve this task. An

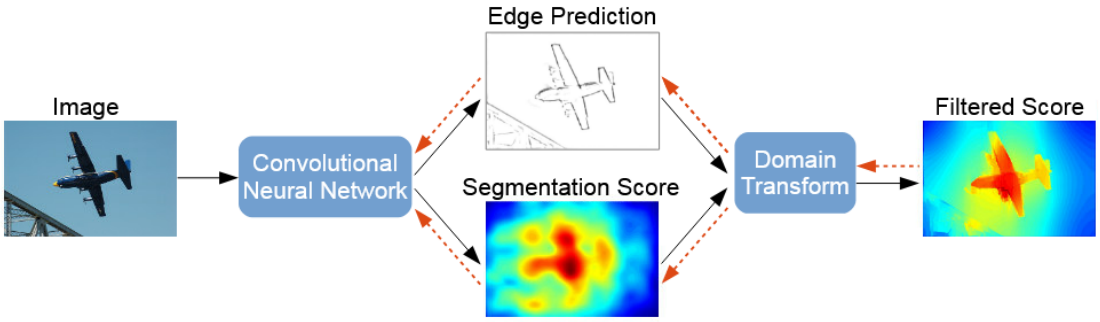

Figure 20: Summary of the idea behind filtering saliencies maps

input image is given as an input to the neural network. The latter will compute multiple saliencies map called in the next figure 'Segmentation score' but also, it will compute an 'Edge prediction' which is a binary or grey scale image that approximate in a robust way the gradient of our input image. The filter is the following :

$$y_i = (1 - w_i)x_i + w_i y_{i-1} \tag{20}$$

With x_i the i^{th} pixel of the input image in the applied direction. The weight is :

$$w_i = exp(\frac{-\sqrt{2}}{\sigma_s}(1 + g_i\frac{\sigma_s}{\sigma_r})) \tag{21}$$

With g_i the i^{th} pixel of the edge map, and σ_s,σ_r are hyperparameters that allow to set the degree of smoothing. When the density of the edge map g_i is high, y_i approach y_{i-1} so the smoothing is done and on the other hand, when g_i is low, y_i approach x_i and nothing is filtered. This technique can be interesting but some stripping artefacts appears and can be considered as noise in the filtered saliency map. The fact that we need multiple two pass (horizontal, vertical, backward, forward) can be very computationally expensive. We obtain the following gaussian response for the given edge map :

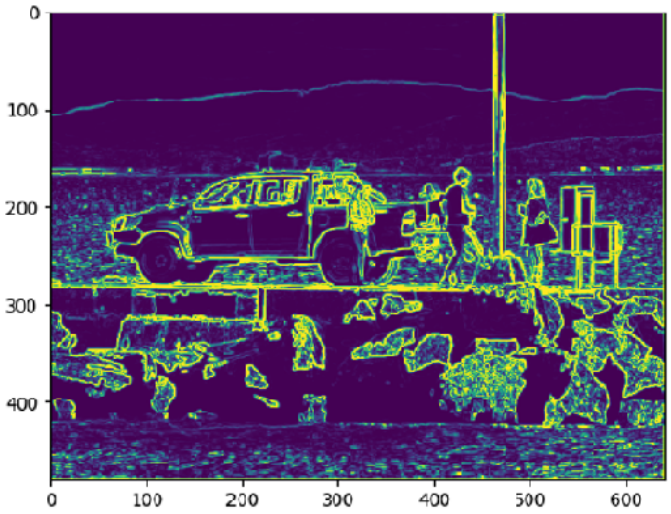

Figure 21: Filtering results : Edges used to compute the response

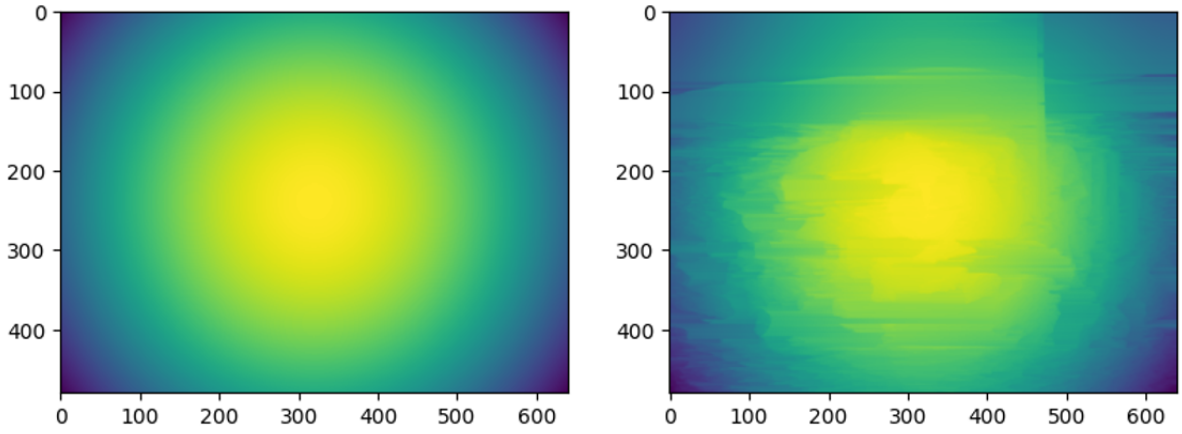

Figure 22: Filtering results : (Left) Gaussian input, (Middle) Response to gaussian input

8.2. Gradient domain merging with Green's function

Another solution presented by Dominique Beaini is gradient domain merging [21], which is based on the fact that we cannot easily combine an edge map with an RGB/Grayscale image due to their separate spatial domains. The saliency map will be in the same domain as the edge map because the technique will use the gradient domain to combine them. Once they're both in the same gradient domain, the purpose is to combine them with a complex norm and angle merger. Finally, in order to retrieve the combined image, we compute the Laplacian (and derive it once from the gradient domain). In the FFT domain, we use Green's function to resolve the Laplacian. This method is really valuable because it provided us with an inverse transform to the derivative for the images. With this technique, a picture can be derived twice to obtain its Laplacian, and then the original image can be recovered by resolving its Laplacian. It acts exactly like an integration. Of course, the recovering is not perfect as it needs colour correction as post-processing step but for saliency maps, it is not case. After having implemented the technique, I found the technique a bit too slow even with vectored computation. On a CPU, we reach 500ms per image of 224x224x3 because of the need to compute for each channel one 2D FFT, one Hadamard product and one 2D iFFT. Considering that we can have multiple channels for our saliency maps, ten for example, it is very computational expensive.

IV. Local machine learning framework

1. Requirements

Klaas Mussche recommended building a local framework to bring the EO team's machine learning activities together. The goal for all future interns and engineers working on machine learning for the EO team is to avoid anarchic organisation. Project abstraction is critical since research projects are typically constructed at random with limited system engineering design capabilities. The project's general structure should be independent of the language chosen. Matlab and Python, for example, must be framework-compatible, although this is not an exhaustive list. The system should allow users to simply retrain models, examine outcomes, and make predictions. The repositories must be easily available to following projects using Thales' existing version control system: a self-hosted version of Bitbucket, a Git online platform for hosting project repositories.

2. Work done

To avoid having to wait for the end of trainings, which can take a long time, I could work on my side-project. I did a couple things to help with this, which I'll go into in the next part. The work is not connected to data science, but rather to system engineering design.

The development of a Bitbucket monorepo was the first item achieved. A monorepo is a repository that has subfolders for all related projects. We decided to employ Git submodules to achieve monorepo. It denotes that the mother project maintains a large number of Bitbucket separate repositories in the projects folder. All framework-generated projects follow a preset template, have their own Bitbucket url, and are registered in the monorepo. To do so, we had to connect to the local Bitbucket API to receive repository information as well as conduct tasks like repository creation, deletion, and updates. For this work, the Atlassian Python package's Bitbucket client is utilized.

The second point to note is the command-line interface. Because the framework must be language-agnostic, we decided that a command line interface would be the best approach to connect different backends, such as Python or Matlab. We utilize the libraries Click and Rich for visual effects and to construct tables using Python straight in the terminal to create this command-line utility. This application must allow users to list, create, remove, and repair connected repositories controlled by the monorepo.

The third purpose is to automate the usage of Git commits, pulls, and pushes depending on query instructions from the command-line interface. To do this, we must provide

all of the fundamental functionalities since the framework must be designed with a high-level configuration in mind, making it easy for users to use the system without a thorough understanding of Git. We do this by employing the subprocess library of the Python standard library. This module enables us to perform Bash commands, hence automating Git commands.

The last step will deal with downloading and uploading data sets. Users may download and upload data sets from and to a shared location that refers to the SFTP server, which is a vital functionality. The command-line interface user can see a list of remotely accessible data sets. He can choose one and save it to his project's local data sets folder. In contrast, a user may construct their own data sets locally and then remotely publish them to the SFTP server for other users to access.

V. Recommendations

Obviously, having specific annotated data is the first thing I would recommend to Thales Nederland B.V. for the task I did. The algorithm may lose its sense of generalization while remaining extremely efficient on the learned task. The maritime environment is simple, textures are generally similar and variation is extremely minimal in compared to autonomous driving. With a little effort, a neural network model can detect the sky and water with great accuracy. However, more energy will be required to discriminate with high precision the lands, vehicles, and people because the standard deviation of these clusters is far greater than that of the sky and water.

My second recommendation will be concerning the data's time dependence. The purpose, in reality, is to apply the image segmentation method to videos. We'll need a basic algorithm like the one I created to do this. Then, by wrapping the neural network base model in a new architecture that accepts the previously predicted mask and the new picture as input, we may upgrade it to consider the optical flow of the videos. Because optical flow is continuous, the segmented masks should not change drastically between t and t time steps. We could also use deterministic physic models in order to solve differentials equations on the AI predicted regions. The first equation that comes my mind to improve predictions in the temporal domain is the advection differential equation, some of the basic kinematic models but also the optical flow differential equation. This will allow the algorithm to focus on minor changes, which can be a pretty interesting path to pursue in my opinion. The first prediction should be accurate but can also be corrected by the future predictions.

The third guideline is to use Vision Transformers (Vit) only after a large amount of data has been annotated. These new models are promising, but they require a lot of data. They shatter all the greatest scores in several sectors, but a training from scratch is now reserved for Google, Amazon, Facebook and Microsoft because of the massive amounts of annotated data they possess. Today, in 2023, I believe that the most essential aspect of data value is when it is annotated.

The fourth and final suggestion is to hire a specialized data scientist with a Ph.D. or a Master's degree. Since 2015, the career of data scientist has been exposed to numerous developments, with many institutions and universities changing their courses to produce data scientists. A specialized data scientist will be able to explain what is and isn't doable based on the available data and will immediately understand the limits of what he needs to create. Being a data scientist, in my opinion, is a very time-consuming job, as the subject is vast and specific. This field has its own specificities, which are numerous and require a great deal of exercise.

VI. Conclusion

Finally, we created a whole deep-learning pipeline for the RGB image semantic segmentation challenge. We started by a deep analysis of the theoretical state-of-the-art of image semantic segmentation in order to gain the maximum amount of time not to go on fruitless directions. We chose to develop with the library TensorFlow three architectures from the state-of-art analysis. We created our own combined and filtered tailored data set where we used data augmentation to increase the amount of varied images and also sample weighting to reduce the classes unbalance problem. We tried classical loss functions, composite but also edges-aware ones where the latter seems to be the most efficient ones for my study.

We found encouraging results for the maritime environment, with 94% pixel wise accuracy and 85% mean intersection over union on the architecture called PIDNet which seemed to be the most efficient among the ones I chose. However, these results can be enhanced by annotating Thales sensitive data for semantic segmentation and making the algorithm really specific for the task it needs to complete. Finally, I learned a lot about how to refine and fine-tune the deep-learning pipeline for improved performance but also I learned a lot of culture on the way we design neural networks.

VII. Greetings

The internship enabled me to improve my English and discover a different European culture, outside and inside the work. I was able to witness France's strong influence, which is giving hope to my country, notably because it has been going through some very intense social unrest in recent months when I was abroad. At Thales, I met a lot of kind people who gives a lot of respect and confidence which put the employees in a comfortable work condition. In the Netherlands, I met a lot of Dutch with whom I had great experience especially the flying trip when my colleague Joshua was piloting the plane. I'd like to express my gratitude to Klaas Mussche for his unfailing belief in me. He was very impressive in computer science for his multitasking engineering talents. I'd also like to congratulate Hans de Groot on his tremendous career, which has been the most impressive I've witnessed thus far. He was supposed to retire in July 2023, but he chose to work as a contractor three days a week instead. I'd like to wish each and every one of them all the best for the future.

During my internship, I had to think hard about my future and what I desired. I had several interviews for a consulting position in Bordeaux, my hometown, to begin practicing and developing data science algorithms. But there is one thing I've learned throughout life. You don't count the working hours when you're passionate about applied science, mathematical theory, computer science and programming. This is the primary reason I chose my Ph.D deep learning topic, which I will work on for the next three years.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[3] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. *CoRR*, abs/1807.10165, 2018.

[4] Huimin Huang, Lanfen Lin, Ruofeng Tong, Hongjie Hu, Qiaowei Zhang, Yutaro Iwamoto, Xianhua Han, Yen-Wei Chen, and Jian Wu. Unet 3+: A full-scale connected unet for medical image segmentation, 2020.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[6] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *CoRR*, abs/1606.02147, 2016.

[7] Roland Gao. Rethink dilated convolution for real-time semantic segmentation. *CoRR*, abs/2111.09957, 2021.

[8] Rudra P. K. Poudel, Stephan Liwicki, and Roberto Cipolla. Fast-scnn: Fast semantic segmentation network. *CoRR*, abs/1902.04502, 2019.

[9] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. *CoRR*, abs/1808.00897, 2018.

[10] Yuanduo Hong, Huihui Pan, Weichao Sun, and Yisong Jia. Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *CoRR*, abs/2101.06085, 2021.

[11] Jiacong Xu, Zixiang Xiong, and Shankar P. Bhattacharyya. Pidnet: A real-time semantic segmentation network inspired by pid controllers, 2023.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[13] Yechan Kim, Younkwan Lee, and Moongu Jeon. Imbalanced image classification with complement cross entropy. *CoRR*, abs/2009.02189, 2020.

[14] Seyed Mohammad Hassan Erfani, Zhenyao Wu, Xinyi Wu, Song Wang, and Erfan Goharian. Atlantis: A benchmark for semantic segmentation of waterbody images, 2021.

[15] Borja Bovcon, Jon Muhovič, Janez Perš, and Matej Kristan. The mastr1325 dataset for training deep usv obstacle detection models. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[16] Miguel Ribeiro, Bruno Damas, and Alexandre Bernardino. Real-time ship segmentation in maritime surveillance videos using automatically annotated synthetic datasets. *Sensors*, 22(21), 2022.

[17] Holger Caesar, Jasper R. R. Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. *CoRR*, abs/1612.03716, 2016.

[18] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[19] Sandip Paul, Bhuvan Jhamb, Deepak Mishra, and M. Senthil Kumar. Edge loss functions for deep-learning depth-map. *Machine Learning with Applications*, 7:100218, 2022.

[20] Liang-Chieh Chen, Jonathan T. Barron, George Papandreou, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform. *CoRR*, abs/1511.03328, 2015.

[21] Dominique Beaini, Sofiane Achiche, Alexandre Duperré, and Maxime Raison. Deep green function convolution for improving saliency in convolutional neural networks. *CoRR*, abs/1908.08331, 2019.