Segmentation of Cloudy Regions

Baddou Othmane
Student at CentraleSupelec
othmane.baddou@student-cs.fr

Benaija Ismail
Student at CentraleSupelec
ismail.benaija@student-cs.fr

Bakkoury Ayoub
Student at CentraleSupelec
ayoub.bakkoury@student-cs.fr

Abstract

Climate change is a major current issue for which the construction of more efficient forecasting models would make it possible to better anticipate natural disasters and better understand the mechanisms of climate variations. This is particularly true for anticipating the formation of cloudy precipitation, which is what we will focus on in this project. We have trained various segmentation models of satellite images containing cloud formations. In order to densify our dataset and make it usable, we explored image generation techniques, set up recognition masks, as well as image processing.

1. Introduction

This project is inspired by a Kaggle competition initiated by the Max Planck Institute of Meteorology. The goal of the competition is to identify cloud formations on satellite images. It is proposed to identify 4 different classes of cloud formations :

- Sugar
- Flower
- Fish
- Gravel

These cloud formations play a determining role on the climate and are difficult to understand and to implement in climate models. By classifying these cloud formations, researchers hope to better understand them and improve existing models.

1.1. Data Sources

We used the open-source dataset from the Kaggle competition "Understanding clouds from satellite images". This dataset consists of a directory of 5546 labelled images, 3698 unlabelled images, and a .csv file detailing the location of the masks (in run-length encoding -RLE- format) as well as the type of formation on each image. There are four formations (fish, flower, gravel and sugar). Each image can contain from one to four different formations (several classes per image).

The images are from the NASA Worldview site and represent views of three different regions of the globe. They are colour photos that are compositions from two polar orbiting satellites (TERRA and AQUA). As these images are created by assembling several views, they have black bands on the majority of the images, as well as stronger brightness bands (one per image).

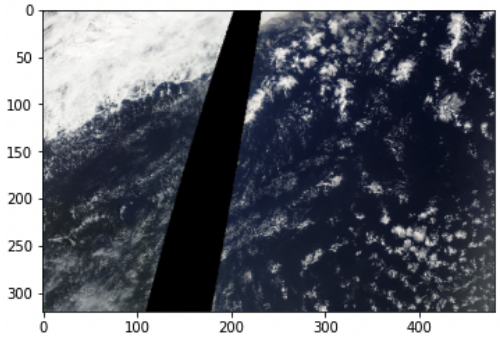

Figure 1. Example of an image from the dataset, in which different cloud formations can be seen. .

After observing the different types of formations, we noticed that some were more likely to be problematic to detect than others. The 'sugar' formation is very dim, the bright pixels are very scattered. It is therefore not very distinguishable from the background. The 'fish' formation is also very bright but has a rather long shape. It is therefore not always well defined when the 'line' is diagonal to the frame. This is partly due to the rectangular annotations. One of the partic-

ularities of this project is that a pixel can belong to different classes.

1.2. Related Work

As this project is a Kaggle competition, we have mainly fed it with contributions from other participants and their discussions. The challenge being to do semantic segmentation, we first did a literature review based on [1] and [3] to better understand how our problem works and what is at stake. These papers go in depth into the architecture of segmentation models and the encoding/decoding principle. This method is more suitable than other more elementary processes for our project. Indeed, the latter consisted in processing pixels one by one for segmentation, which is incompatible with the fact that a single pixel can correspond to different cloud classes in our project.

In [2] a state of the art of pre-trained models that can be used as a backbone is provided, which allowed us to make a better choice of models later on. In particular, this paper describes the characteristics and performance of the AlexNet, VGG-Net and ResNet encoders. It also explores the Unet, AD-LinkNet and DeepLab segmentation architectures. The performance of these models is calculated from the IoU function (which stands for Intersection over Union), which is, according to [2], the best performance indication for semantic models. The presented architectures all outperform the ImageNet structure, which is less appropriate in these cases. However, it seems that the majority of the participants in the cloud segmentation competition opted for different variants of the Resnet network as a backbone, while mainly using R-CNN, F-CNN, UNet or SegNet as a decoder.

We drew the most inspiration from [4] and [5], which represent well what has been tested in general by competitors. These two entries approach the problem in similar ways. [4] proposes a Mask R-CNN model based on a ResNet18 backbone, while [5] combines two different models, one with Unet architecture and Resnet18 backbone, the other with Linknet architecture and 'efficient Net B1' backbone. In addition, both projects propose various techniques for image augmentation and pre- and post-processing of images to optimise the results. We have been inspired by some of these techniques in our work.

2. Background

2.1. Semantic Image Segmentation

The project falls within the framework of image segmentation. The goal of these techniques is to find and identify objects in an image. Basically, the task is nothing more than classifying pixels. There are two types of segmentation techniques :

- Semantic segmentation
- Instance segmentation

The major difference being these two is that in instance segmentation, we try to locate objects in the image but also to determine and locate each instance of the class we are looking for. In other term, if we are trying to locate cats in an image, each cat of the image is going to be labeled differently as its own instance. However in semantic segmentation, we do not make a difference in labeling objects from the same class. Therefore, our problem is a semantic segmentation.

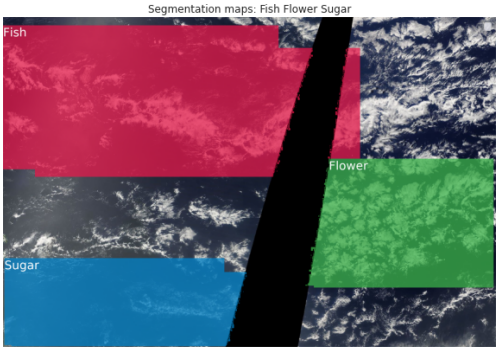

Figure 2. Cloud Segmentation, image taken from the Kaggle competition

To perform such a task, the most effective way is to use a deep neural network. A lot of architectures are suited and can be applied to our problem. Before discussing the models, let's talk about the evaluation metrics and the loss function.

2.2. Evaluation Metrics

Let's discuss the two most common evaluation metrics for segmentation problems :

- The IoU
- The Dice Coefficient

2.2.1 The IoU

The intersection over union (or also called Jaccard Index) is one of the most commonly used evaluation metric for segmentation problems. The figure 2 represents a visually explanation of how the metric is calculated. As we can see, the IoU is the area of overlap between the prediction and the truth divided by the area of union of the prediction and the truth. For a multi class problem the IoU is simply the average of each class' IoU.

figure[h!]

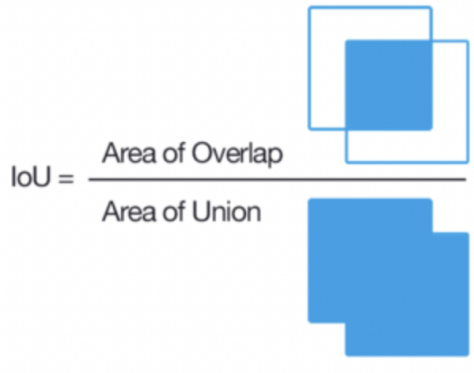

Figure 3. Visual representation of IoU metric

2.2.2 The Dice Coefficient

The dice coefficient is nothing more than the F1 Score. The way it's calculated is straight forward :

$$dice = 2 * \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

The IoU and dice coefficient are very similar. When used to determine which model is best, both metrics will always point out to the same one and give very similar metrics.

2.3. Loss function

2.3.1 Classical losses

The loss function is used to guide the optimization process of a neural network while training it to determine it's parameters. For our problem we decided to use the two most common loss functions for training semantic segmentation models :

- The Binary Cross Entropy Loss. This function can be used simply as the average of cross-entropy classification loss for every pixel in the image. However this method is very sensitive to class imbalances. Some models such as UNet are designed to offset such a problem. In that case, a problem with $M > 2$ classes, we calculate a separate loss for each class label per observation and sum the result.

  For each pixel i :

$$bce_i = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2}$$

- The Dice Loss. The name of this loss function explains well what this is. Obviously, it is based on optimizing the F1 score. We note that we can also use direct IoU score to run optimization in a similar way.

2.3.2 Custom Loss

We noticed in the course of our work that the classical loss functions known and proposed by the challenge were not necessarily the most suitable for the models we were defining. Some of our models gave much more weight to the background (i.e. the sky) than to the cloud formations to be predicted. We therefore computed a weighted binary cross entropy loss function. This function gives more weight to the non-detection of one of the four cloud classes than to the detection of a background. Indeed, we will see later in the paper that we have reduced the classification problem into four sub-problems of binary classification formation (label 1) vs. background (label 0). The custom loss was calculated as follows:

The bce is calculated for each pixel and each label in the batch.

Let the weight ratio be: $r_i = w_1/w_0$. We then calculate the weighted bce of the pixel from this ratio, and calculate the sum of the classification errors.

Finally, we return the average per pixel loss.

$$customedbce = 1/n * \sum_{i=1}^{n} l_i \tag{3}$$

with

- n the number of pixels
- $l_i = \sum_{c=1}^{M} bce_i * r_i$

2.4. Model structures

2.4.1 UNET family Models

To be able to accomplish a task like image segmentation, some changes to regular convolutional neural networks (CNN) can be applied. In a regular image classification task, as we can see in figure 4, the layers tend to get smaller and smaller until the prediction "y".

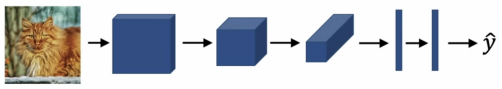

Figure 4. Visual representation of a CNN architecture

However, for the UNET model architecture, as we can see in figure 5, we gradually increase the layers' sizes so we can end up with a full size image.

In order to gradually increase the size of the layers, the Unet model uses what we call a "transpose convolution". In a regular CNN architecture, a normal convolution gives an output that is smaller than the input based on a filter size and some parameters. The transpose convolution is just the

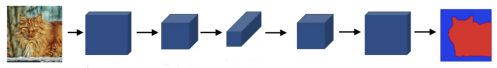

Figure 5. Visual representation of a UNET architecture

opposite operation : the output is bigger than the input based on a filter size and some parameters.

To get into more details about how the architecture of this neural network works, we included figure 6, a visual representation by Andrew Ng from his deep learning class at Stanford [7].

We decided to focus on this family of models for our study. This was indeed the structure favoured by the competitors of the challenge. We first tried to build a structure "from scratch", a Vanilla UNet model. We then compared it to so-called backbone structures. This allows us to leverage the performance of pre-trained structures such as VGG-16 or Resnet.

2.4.2 ResNet Model

The ResNet architecture has been very promising in improving performances of very deep neural networks. The idea from the paper [7] is to add skip connections to the layers. As we can see on figure 6, the difference between the plain neural network and ResNet is the black arrows representing these skip connections. Basically these connections add information from the previous layers to the feed forward mechanism. For a plain neural network we have :

$$a[l] = g(z[l]) \tag{4}$$

With the skip connection the equation becomes :

$$a[l] = g(z[l] + a[l-1]) \tag{5}$$

With g(z) being the activation function, and

$$z[l] = w[l] * a[l] * b[l] \tag{6}$$

with w[l] and b[l] being the weight and bias parameters.

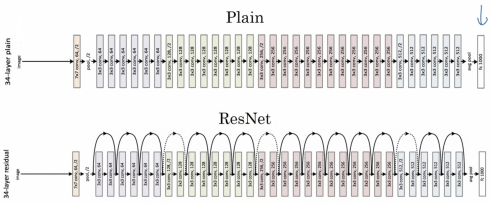

Figure 6. Difference between plain and Resnet NN

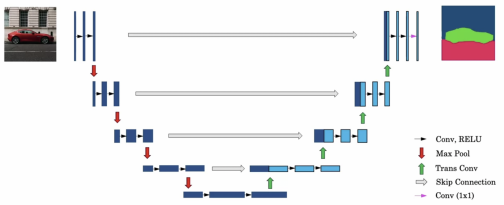

Figure 7. Andrew Ng representation of UNET architecture

The left part of the U-shaped architecture consists of standard convolution, fully connected layers and max pooling. The right part consists of transpose convolutions and fully connected layers. However the interesting part of this architecture is that we add skip connections between the left and right side to keep information regarding the overall image context. These are represented by the big grey arrows in figure 7 connecting the layers of the left side to the ones in the right side. Basically these connections feed information about the image structure to the layers expanding in size. This is done by using the output of a layer as the input of another. As we can imagine, this type of model have a big amount of hyper parameters to tune.

3. Experiments

3.1. Approach

Our approach was as follows:

- Import of libraries and packages
- Pre-processing of data and creation of masks
- Creation of a data generator (data augmentation)
- Definition of metrics, loss functions (native to Keras or created to fit the challenge) and callbacks
- Creation (or loading), compilation and training of a model
- Post-processing the data to visualise it correctly
- Visualisation of results and performance analysis
- Fine tuning
- Eventually submitting the results to measure the performance of the model on Kaggle.

3.2. First iteration

After importing all the necessary libraries and loading the data, the first task was to pre-process the available data and apply masks to the training images. In the training .csv file (see data), each image was labelled on average by 3 NASA scientists. The masks were defined by combining the

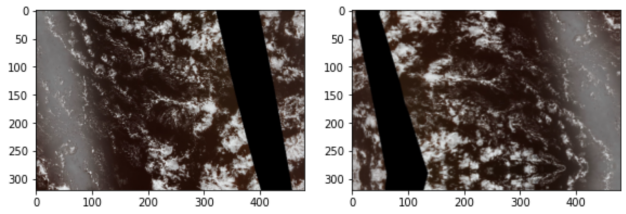

Figure 8. Example of data augmentation : Small crop of the image and horizontal flip .

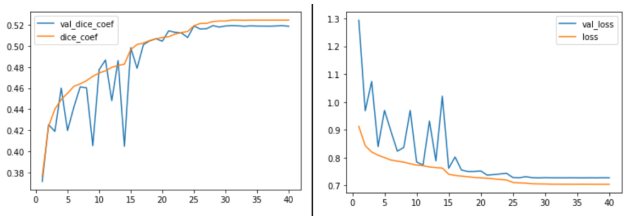

Figure 9. Evolution of the dice coefficient and the loss function of the Vanilla model

areas marked by the scientists on each image, omitting from these masks the black bands from different shots. This document is composed of 4 lines per image. Each line designates the mask of one of the formations (fish, flower, gravel or sugar) in run-length encoding format. If the formation does not exist in the image, the column "Encoded Pixel" is set to NaN. We therefore pre-processed the .csv file so that we could create masks from this data. We then created an annotation folder with four files for each image (one mask for each formation type). Each annotation is of binary type with the pixels defined as 0 (background) or 1 (presence of a formation). In addition, we reformatted the training file in the same way as in [5], so that for each image, we have the delimitations of the mask to apply. Thus, the absence of one of the four cloud formations in the image is characterised by an empty box (NaN).

In order to diversify the dataset, we have defined a Python class of data generator. This also makes the model more generalizable and therefore reduces overfitting. We performed this image augmentation using the Albumentations library. Our data augmentor performs simple geometric transformations. We have chosen not to make it too complex at first so as not to weigh down our dataset and keep the focus on our study. It performs rotations within 20 degrees, vertical and horizontal mirror flips with a probability of 0.5, a random crop of the image at maximum resolution, and occasional distortions.

After documenting and comparing the different models in [1] [2] and [3], we decided to focus on the Unet architecture as a first step. We first compared two kind of structures : One with a VGG-16 as a backbone, and Vanilla UNet with 3M parameters. The difficulty in building the models layed in how we posed the problem. The segmentation here is multi-label. After reflection, it seemed inconsistent to use 15 or 16 classes for this problem (1 pixel can belong to several classes). Indeed, in the majority of cases, the pixels belonged to only one class (or to the background), and thus the border pixels would be underrepresented in the training database. We thus considered four binary sub-problems "Cloud vs Sky (background)". To consider the multi-class boundaries in these sub-problems, we constructed four-dimensional masks, where the last dimension corresponds to the pixel value.

For the first model, we used an SGD optimizer with a learning rate of 0.01, which was the only one in this case to be compatible with the implemented callbacks (Adam optimizer caused the fit to fail). These callbacks are :

- Model checkpoint, which enables us to save the model and it's weights to "save" the training from time to time and to avoid losing all of our history after a training.
- Learning rate reduction on plateau. Indeed, we wanted to reduce the learning rate once it stagnates. If no improvement is seen for a 'patience', it this case a number of 5 epochs, the learning rate is reduced by a factor 10.

For the Vanilla U-Net model, we used an Adam optimizer with a learning rate of 0.001. The U-Net + VGG-16 encoder model was trained on 10 epochs, whereas the Vanilla one on 40 epochs. The first model gave us very poor performance. The visualisations we observed were always poor. The model seemed to spot only one class, and to generate more or less random results depending on each prediction. The Vanilla U-Net model, however, trained on a larger number of epochs (and therefore for a longer period of time), allowed us to obtain more satisfactory results. For this model, we defined a custom binary cross entropy function that weights the different classes. We were able to visualise the evolution of the loss function and the dice coefficient over the epochs, as well as the predictions of the model in the form of masks.

3.3. Second Iteration

We were quite happy with the performance of the Vanilla U-Net model predictions, but we felt that we could go further, especially by trying other models, or by going further in the optimisation. We decided to focus on U-Net structures, but this time we changed the backbone to a ResNet34. The backbone weights are those pre-trained on the ImageNet dataset, and are imported thanks to the segmentation models project [6]. The model contains about 24 million parameters, but we froze 21 million for a first attempt, in order to reduce the GPU memory required, as well as the computation time. The model was trained with an Adam

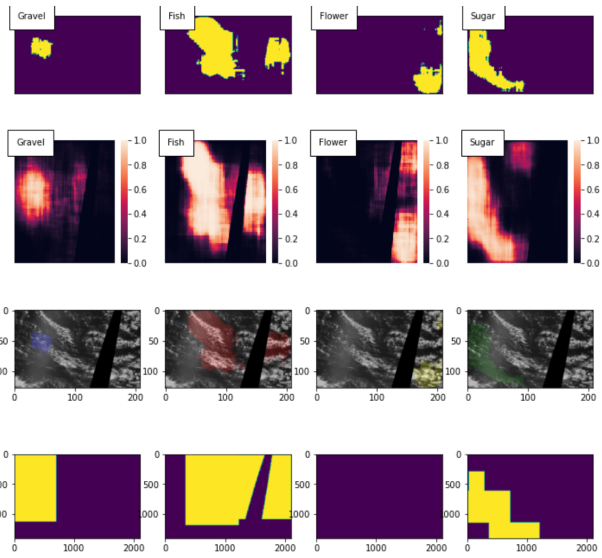

Figure 10. Visualisation of the vanilla model predictions. Top: the real mask, bottom: the prediction.

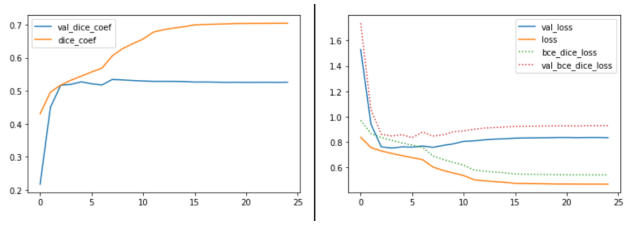

Figure 11. Training graphs for the Unet model - Resnet34.

optimiser set up in the same way as for the vanilla model, and with the same callbacks. This time the model overfits but learns very quickly, with a better metric at dice = 0.53. The training was started with an initial learning rate of 1e-3. The dotted lines on the 2nd graph show the curves of a custom binary cross entropy. According to figure 8, we will reduce the number of epochs to 10/12 on the following iterations, especially those aiming at improving our performance by fine-tuning.

The network seems to better capture the areas of interest, indeed outside the annotated area we observe on the heatmap almost zero predictions. Even if we do not seem to gain on the dice metric, we gain in precision in the determination of the shapes, as noticed on figure 7. The model also discards the noise due to the rectangular shape of the masks and only captures the important parts of the formation. However, when we set a threshold for the prediction, it can happen that a formation is selected, even if it is small and absent on the original mask. An improvement step would be to post-process the prediction after setting thresholds in order to keep only the predictions that represent a

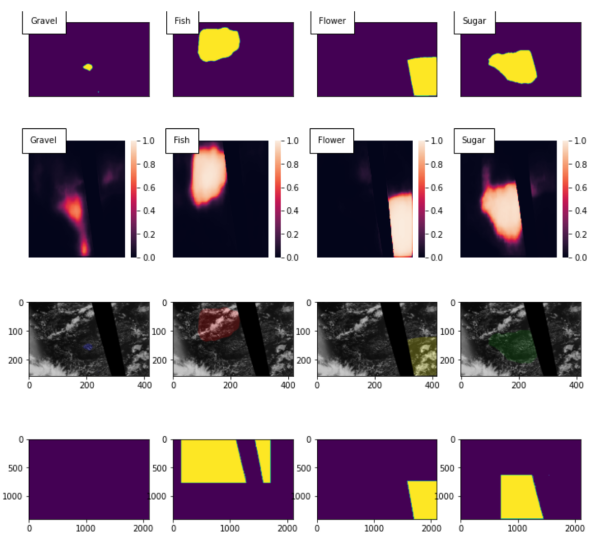

Figure 12. Predictions for the Unet model - Resnet34.

significant area, as the competitors did in [5].

3.4. Optimisation of the learning rate

In order to optimize the model, we decided to study the influence of the learning rate on the performance of the model in terms of dice and binary cross entropy. In the first trainings, we started with a fairly high learning rate for the chosen Adam optimizer, i.e. around 10^{-2}, trained on 10 epochs. This allowed us to have a first model quickly, which, even with poor performance, allowed us to work in parallel to develop and improve the prediction visualisation functions. Following this, we reasoned as follows:

- Gradually decrease the learning rate over a total of 15 epochs, still with the freezed encoder. It was divided by 1000 in the end. This resulted in a performance gain of almost 15 percent. It was observed that from about 10 epochs onwards, the model started to overfit, and thus the improvement due to the reduction of the learning rate alone reached a plateau. The model was still beginning to capture good areas of interest, and the good prediction of classes was beginning to balance out well.

- We therefore decided to "progressively" increase the number of trainable parameters of the encoder. First, we unfreeze the last layers of the Resnet34 backbone (about 20 percent of its parameters), and decrease the learning rate to 10^{-6}. We reloaded the old weights and ran a fit on 5 epochs. Unfortunately, this had no effect on performance, but also on predictions, which remained the same as before.

- We finally unfroze the entire backbone parameters and trained the model on 5 epochs. This brings the total training to 35 epochs. This clearly improved the predictions and the capture of the areas of interest of the model. It is at this point that we observed the gain in accuracy on the null zones of the heatmap. We gained 10 percent more performance following this training, which overfitted after 2 epochs.

We kept exploring the influence on learning rate on the performance, but it seemed that setting it to 10^{-6} was the optimal value.

3.5. Computation Power

In order to be able to train and run our models, we quickly found the need to move from our personal computers to actual computing centers. In order to do that, we used Paris-Saclay mesocentre. The mesocentre brings together 3 major technical platforms for scientific computing/data processing from the partner institutions in the Paris-Saclay project. We were able to connect to the mesocentre VM and move our code there which allowed us to make use of the GPU nodes. The one we used is equipped with an Nvidia Tesla V100. With this GPU, training was about six times faster which made it possible to explore and to fine tune several parameters.

4. Conclusion

The particularity of this project lies in the fact that the only reference we had to go on were the open source contributions provided on Kaggle. This allowed us to abode the problem according to our own approach, while feeding off the discussions and errors encountered by competitors on the kaggle forum. It is for this competition and optimisation of results that we chose this project, which remains renowned for the number of participants and the excellence of the leaderboard.

We also chose to structure the code in the form of a streamlit file, like the project [5], although their file is much more extensive, for the purposes of a "showcase application". We found that this format, more or less original, allows to easily consult the project code and to execute only the necessary. We had a real satisfaction in developing this project as the issue of climate change is very close to our hearts. We thus realised that our skills as data scientists allow us to contribute to this, even on slightly more distant axes such as meteorology. Moreover, we noticed our own progress during the project despite the many obstacles we encountered (bad predictions, overfitting, loss functions not adapted or loss of model backup after a very long calculation time, mask functions not adapted, etc.).

Moreover, despite the results that are not excellent in terms of performance (dice), we note that the models still manage to make acceptable predictions and to identify good areas of interest. An additional step would be to post-process the predictions obtained as in [5] to gain even more precision on the delimitations and provide higher quality annotations. The data structure, although very large, included some biases that we identified, but which we were unable to deal with effectively to improve the performance of our predictions. These included the images corresponding to the sugar class, which was very dim and had very scattered bright pixels, and the sheet class, which was very bright. The image processing that we tried to do to remedy this problem did not affect the predictions. Another area of progress would be to diversify our data generator, and to push the data augmentation further with, for example, GAN styles, although this does not necessarily lend itself to this project.

5. References

[1] : Vijay Badrinarayanan et al.2020: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation .

[2] : Rongyu Zhang et al 2020 J. Phys.: Conf. Ser. 1544 012196 Comparison of Backbones for Semantic Segmentation Network

[3] : James Le 2021 : How to Do semantic segmentation using Deep Learning

[4] Mask R-CNN for Cloud Classification (https://www.kaggle.com/code/malipeddihanisha/maskrcnn-for-cloud-classification-keras)

[5] Yann BERNERY et al. Nebula Project (2021): (https://github.com/DataScientest-Studio/nebula)

[6] Segmentation models Documentation(https://segmentation-models.readthedocs.io/en/latest/tutorial.html)

[7] Deep residual networks for image recognition, He and al. (2015)

[8] Andrew Ng et al. Deep learning class (https://cs229.stanford.edu/materials/CS229-DeepLearning.pdf)