Study on the detection of slow satellite tracks

Alexandre DI PIAZZA
270884

Submitted in partial satisfaction of the requirements for the degree of
MSc Data Science

Laboratory of Computer Vision
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
June,2021

Abstract

This report studies the detection of slow satellite tracks on single epoch images produced by OMEGACAM on the VLT Survey Telescope. We first generate synthetic data on which we train a Neural Network, and then fine-tune our network by freezing some layers and training again on real images of slow satellite tracks for a better generalization to true images. It is in continuation with another semester project on the same subject [1].

Acknowledgements

I would like to express my special thanks of gratitude to Prof. Mathieu Salzmann for his help and availability during the semester, guiding me through the project. I also express my sense of gratitude to Dr. Cameron Lemon and his expertise in astrophysics, which greatly helped me in analysing the images and detecting satellite tracks.

Contents

1 Introduction

The number of debris and satellites orbiting the earth keeps increasing. According to the Union of Concerned Scientist, as of mid 2020 there were 2666 satellites in Space [2]. The risks of collision between satellites and debris, or between satellites themselves, is real. Major collisions have occurred, as wrote Michael Dominguez, former senior Defense Department official [2]. It is thus crucial to detect and keep track of objects orbiting the earth in order to prevent such accidents. US Department of Defense maintains a highly accurate satellite catalog on objects in Earth orbit. NASA and the Department of Defense cooperate and share responsibilities for characterizing those objects and their environment.
The Space Situational Awareness (SSA) EPFL Team also has the goal of setting up a catalog highlighting important information on each detectable object in the Earth orbit. In this project, we study the detection of slow satellite tracks. It is in continuation with a previous project [1] that studied the detection of both slow and long satellite tracks. In the first part of the report, we focus on how to create data that can be used to train a Neural Network (NN) capable of detection slow satellite tracks. In the second part of the project, we use the created data set to train a NN and use the technique of fine-tuning, using real slow satellite streaks in order to achieve better generalization results.

2 Generating synthetic Data

The major problem in training a Neural Network capable of identifying slow satellite streaks is that there aren't real images with labels identifying the streaks. In a previous work [1], Yann Bouquet started to create synthetic images with known labels that would allow to train a NN and hopefully generalize to new unseen data. However, real images were too different from the synthetic ones in most cases, and the NN didn't succeed in finding the features of real satellite streaks. We therefore decided to first use a Generative Adversarial Network (GAN) to generate new data that would look more like real ones. We then decided to modify the previous synthetic images in order to make them more realistic.

2.1 More realistic images using a GAN

2.1.1 Description of CycleGan

The first idea was to create more realistic synthetic images by using a Generative Adversarial Network (GAN). We decided to use an already implemented GAN that was efficient for multiple tasks: CycleGAN (Cycle-Consistent Adversarial Networks [3]) . Given images from a set X and images from a set Y, the goal is to learn a mapping $G : X \longrightarrow Y$ so that the the images created by $G(x)$, $x \in X$, are indistinguishable from images y, $y \in Y$. Similarly, it also learns a mapping $F : Y \longrightarrow X$ for the other direction. It is important to mention that the images in X and Y don't need to be paired: for an element $x \in X$ there doesn't need to be the corresponding $y \in Y$. One important property of CycleGan is also that the transformation should be "cycle-consistent", meaning that if we transform $x_1 \longrightarrow y_1$ and then $y_1 \longrightarrow x_2$, we should have $x_1 \approx x_2$. In other words, $F(G(x)) \approx x$ and $G(F(y)) \approx y$
Below is an example of such a transformation, where in X contains images of horses, and Y contains images of zebras. The GAN learns to transform a horse into a zebra, and inversely.

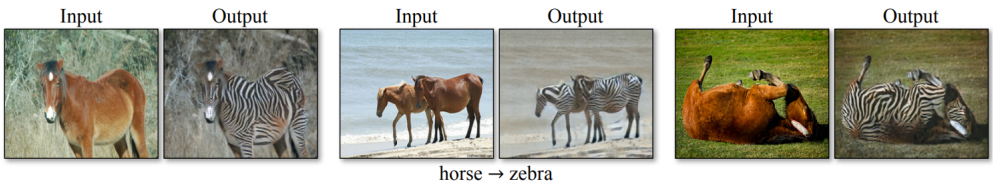


Figure 1: Examples of transformations. [3]

2.1.2 Implementation of CycleGan to our case

In our case, the goal was to have in X the synthetic images containing satellites streaks as they were created by the previous project [1]. The only difference was that the previous synthetic images created were 64x64 grayscale, which we changed to 256x256 RGB images in order to match the input of CycleGAN. To go from grayscale to RGB, we simply used R=G=B=gray value. To go from 64 to 256, we simply took a larger image before creating the synthetic track. The synthetic track itself was the same size. In Y, we had random real images. X contained 2000 synthetic images and Y contained 2000 real images, randomly chosen from 10 .fits file, each file containing a mosaique of 32 images of size 4040x2035. The Network of CycleGAN being very deep and the training thus being computationally expensive, we trained on the Izar cluster, from SCITAS [4].We trained with the same hyper-parameters and optimization algorithm that were used in the paper [3].
We hoped that the Cycle-Consistent Adversarial Networks could transform the synthetic images containing streaks in order to make them more real. However, we found that, as Y were random images that almost never contained satellite streaks, the transformation $G(x),\ x \in X$, actually learnt to erase the streaks through the learning, or transform the streaks into objects that look like stars.
Figures 2 and 3 show the evolution of the mapping for a sample and for different epochs, when we trained CycleGan on our DataSet. The first image of the figures is the original synthetic image x containing a synthetic track. Then the other images show the evolution of $G(x)$ through the epochs. The second image is the output after 5 epochs, then after 20 epochs, 80 epochs and finally 100 epochs.

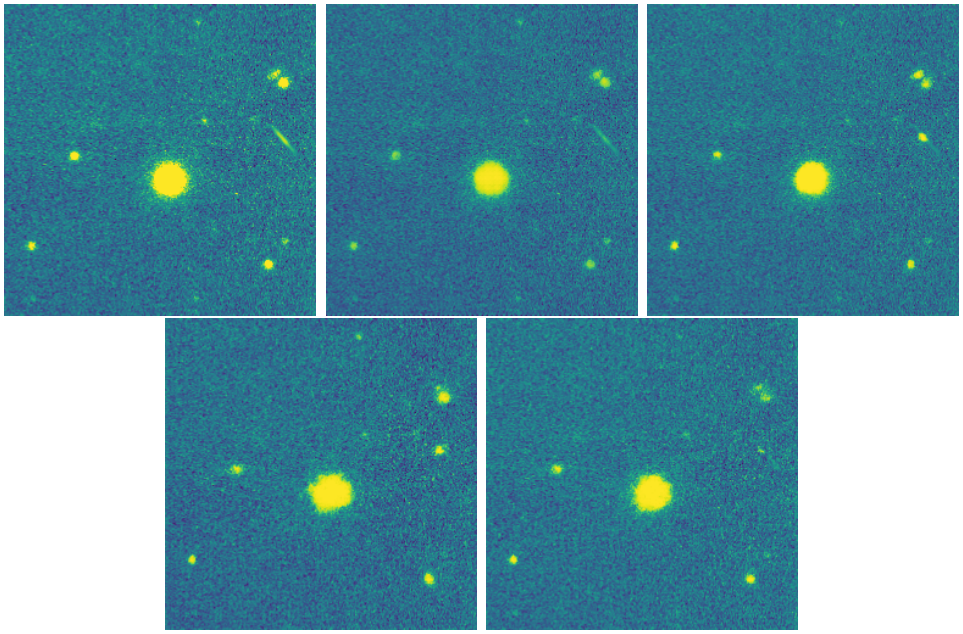


Figure 2: Evolution of the transformation via CycleGan for one sample

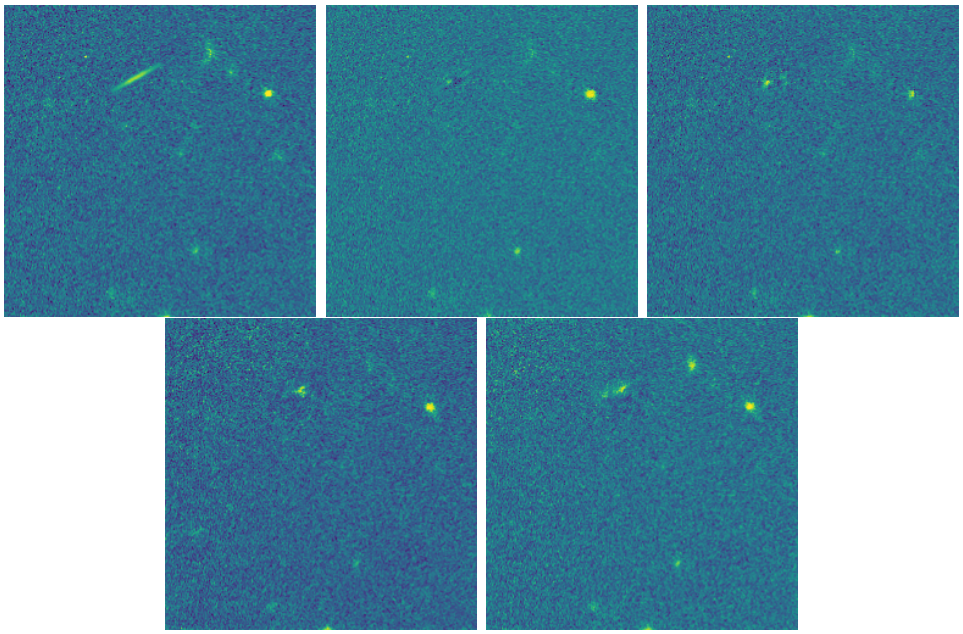


Figure 3: Evolution of the transformation via CycleGan for one sample

On both figures, we can see that the transformations try to delete the track, and even seem to transform it into a star. For example on the 3rd and 4th

image of Figure 2, the synthetic satellite track seems to have transformed into a star. Therefore, we thought that in order for the technique to work and the transformation to be efficient, we needed to have lots of real satellites streaks in the Y domain, so that the network learns a better mapping. We therefore came back to the same original problem that we have for training the Neural Network for segmenting the satellites streaks, which is that we don't have enough real slow satellites streaks in order to do that.
On the other side $Y \longrightarrow X$, the results were also mostly not satisfactory. Sometimes the transformation did not create any streak. When it created one they were often bent, and did not look like real ones. In addition, using directly CycleGan, the mask corresponding to the created streak was not created and therefore it was not possible to train the network directly on the good results. The following figures show the results of the transformations $Y \longrightarrow X$ after 100 epochs. We illustrate the 3 types of transformation that seem to happen.

- the transformation does not create a streak (fig 4)
- the transformation creates a streak that is bent (fig5)
- the transformation creates a satisfying streak (fig6)

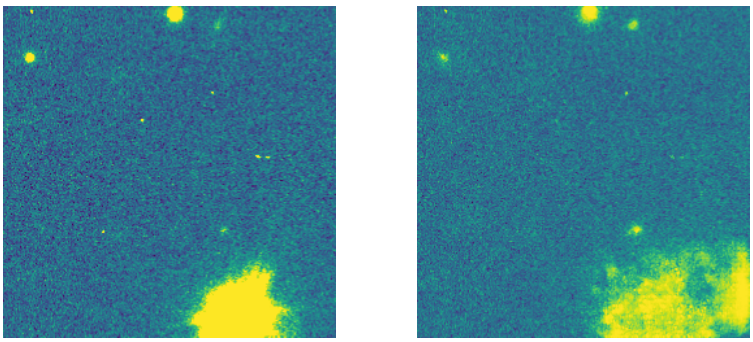


Figure 4: Example of a transformation $y \longrightarrow x$

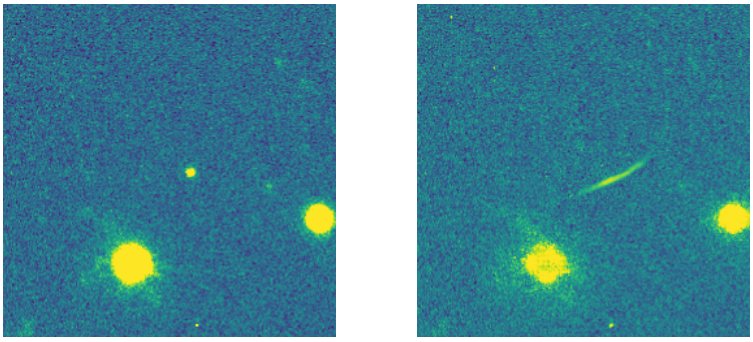


Figure 5: Example of a transformation $y \longrightarrow x$

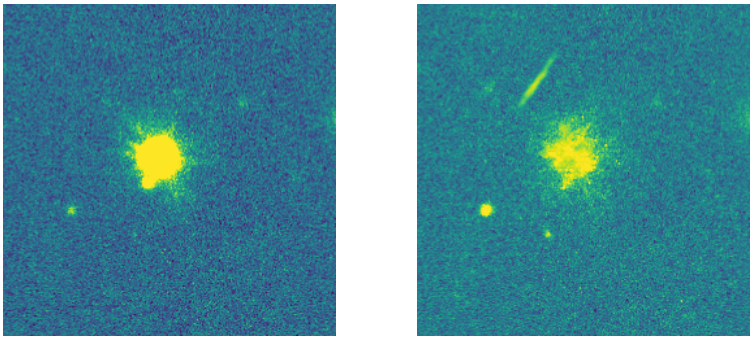


Figure 6: Example of a transformation $y \longrightarrow x$

Both 3 cases seem to be as likely to happen. Seeing that only a fraction of the results was satisfactory, and that some additional work was needed in order to create the corresponding mask when it created the streak, we decided to change our method and to stop using CycleGan. We later talk in the Conclusion about how the technique could be improved in future works, as some of the results from $Y \longrightarrow X$ (Figure 6) were promising and could need some deeper analysis.

2.2 Modification of the synthetic images

Understanding the GAN would not give satisfying results, we decided to modify the synthetic images created by previous the project [1].The dimensions of the previous images were 64x64. After training the NN on these images, the goal is then to go through one single epoch images, and to classify each segments of the image. One single epoch image being 4040x2035 (there are 32 of them in the whole mosaique), we decided that 256x256 images were more appropriate in order to reduce the chance of having a streak across borders. In addition, with the help of Cameron, we realized that real streaks had sometimes a more blurry effect than the synthetic ones. We thus decided to create a new type of streaks, for which the luminosity would decrease from a certain distance away from the center of the streak. Those streaks have a constant luminosity at its core, and then from a certain distant it starts decreasing. The distance from which it starts decreasing is random, which helps generalization. We thus have two types of synthetic streaks, one which is similar to the previous ones, and one with a decrease in luminosity. For the ones that were similar to the previous one, we changed the minimum thickness, as we observed that some streaks were too narrow compared to real ones. For each new synthetic image, each type is chosen with a probability of 0.5, and we thus have around 50 % of both types in the final data set. Fig 7 shows an example of previous images before modification. Figure 8 and 9 show the two types of new synthetic images. Note that the previous images were 64x64 and therefore the streak seems bigger.

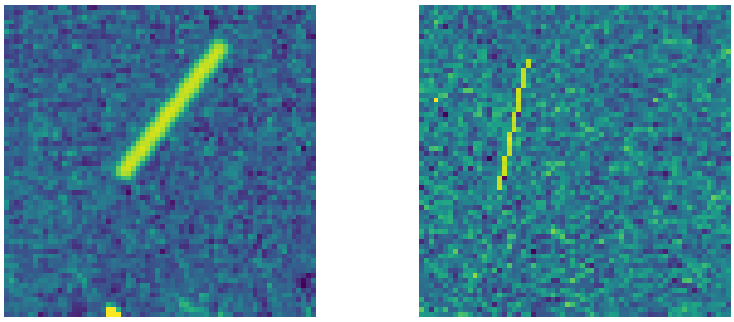


Figure 7: Previous synthetic images

‘

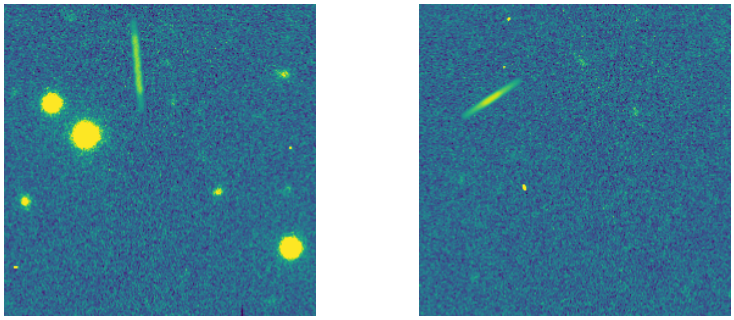


Figure 8: New synthetic images : decreasing luminosity

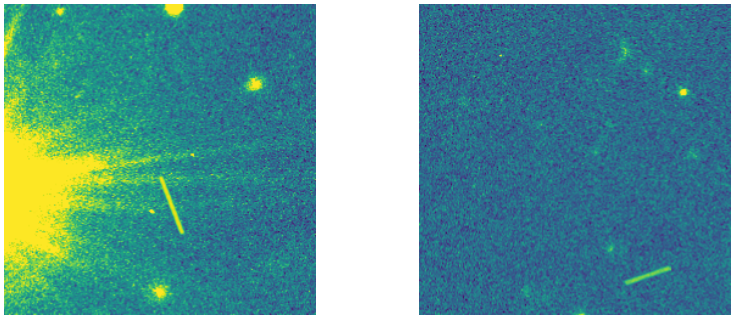


Figure 9: New synthetic images : constant luminosity

2.3 Real Images

We dedicated a lot of time of this project in finding real slow streaks on real images. We inspected slowly around 20 files (each containing 32 images of 4040x2035), writing down the coordinates of what seemed like real satellite streaks. Dr. Cameron Lemon then confirmed us the real ones. We present in Appendix A all the coordinates with the corresponding .fits file so that they are directly available for next projects.
We eventually found a total of 19 images of what seemed like real satellite streaks. For each image, we also created manually a corresponding mask, with 1 when the pixel is part of the streak, and 0 everywhere else. We then performed some data augmentation in order to have more samples of real streaks. We did the following transformations:

- Vertical flip with a probability of 0.8
- Horizontal flip with a probability of 0.5
- Shift-Scale-Rotate with a probability of 0.6

For each sample, we uniformly did one or two transformations. (around 50 % of the time we have one transformation of a sample, and around 50 % of

the time we have two transformations). We finally had a set of 48 256x256 images containing slow satellite streaks (19 real ones and 29 transformed ones).

3 Image Segmentation

The new synthetic images created seem more appropriate to train a Neural Network and generalize to new real unseen data. However, we found out that training only on those new synthetic images is still not enough to generalize to real unseen data. We therefore decided to first train a model on the synthetic images, and then fine-tune the model by training again with real images.

3.1 Model

The model used is the same as the one used in the previous project [1]. It is used for image segmentation in the biomedical domain and for road segmentation in satellite images [5]. The model architecture is the following:

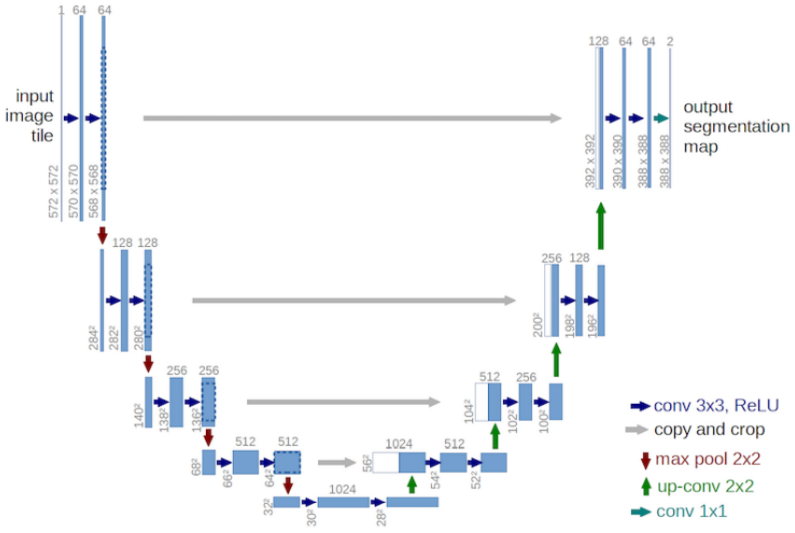


Figure 10: Model Architecture [1]

We kept the modifications that were made by Yann Bouquet in [1] (reduction of the number of channels in the convolutional layers, and only one convolution layer before each max-pooling instead of 2). We only modified the dimensions of the input and the output from 64x64 to 256x256, matching the size of the new synthetic images.

3.2 Training on the synthetic images

We trained on 2000 synthetic images (with around 50% of them containing synthetic satellite streaks) using Adam optimizer. We tested different parameters (learning rate, epochs and the number of epochs to wait before early stopping) but all the resulting models generalized poorly to new real data. The final model was trained on 70 epochs, a learning rate of 0.01 and a dropout value of 0.2. The goal being to obtain the best results after fine-tuning and not directly with the resulting model after training on synthetic images, we did not try to optimize the parameters when we understood that it couldn't generalize to new images. We quantify later in the report the results on real images by comparing the model before and after fine-tuning.

3.3 Best parameters for Fine-Tuning

Fine-tuning is a powerful technique which makes small adjustments to a pre-trained model in order for it to perform better on a specific task. It requires to re-train the model on new images (often a small dataset) in order to learn new features and make better predictions on those new types of images. Lots of different techniques of fine-tuning exist. To avoid overfitting the new data, a common technique when fine-tuning a network is to freeze some layers. The weights of the layers that are frozen don't get updated during the new training. One essential parameter to choose when fine-tuning the network is thus how many layers would be frozen, and which ones. It is common practice to believe that the first layers of the network tend to see the general features of the images, and it is thus usually the last layers that are unfrozen to increase the performance of the model on new, but similar data, while avoiding overfitting. In our case, we decided to try 3 different ways to freeze layers:

- Freeze all the layers except the last convolutional layer
- Freeze all the layers except the last 2 convolutional layer
- Freeze no layers

In addition, in order to find the best model we also decided to try varying the following parameters when re-training the model on the real images :

- Epochs : 40,60,80
- Dropout : 0.1, 0.2
- Learning Rate : 0.01, 0.005, 0.001

To choose the best resulting model, we need to choose a metric in order to compare the output of our model with the true target. In the previous project [1] , the metric used to compare the results of the prediction and the real image was the Jaccard Distance. The Jaccard Distance between two sets A and B is defined as follows :

$$J(A, B) = \frac{|A \cup B|}{|A \cap B|}$$

In our case, instead of comparing directly the output of the network with the real image, we decided to compare the output of the network after tresholding each pixel. Let Y be the 256x256 output and $Y_{i,j}$ be the value of the pixel at the ith line and jth column. We have that $Y_{i,j} \in [0, 1]$. Let Y^{fin} be the final boolean prediction to detect the track, we simply have

$$Y_{i,j}^{fin} = \begin{cases} 1 & \text{if } Y_{i,j} > tr \\ 0 & \text{else} \end{cases}$$

We can thus directly compute the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). The real masks being mostly filled with 0 (around 99,99 % of 0s), and the major goal in our case being to be accurate on the TP, we thought that the most suitable metric was the Sørensen–Dice index. It is a common metric used in image segmentation. It is for example used for comparing algorithm output against reference masks in medical application [6]. When applied to boolean data, it is defined as follows (note that it doesn't take in account True Negative) :

$$DSC = \frac{2TP}{2TP + FP + FN}$$

It is still very similar to the Jaccard index, which is defined as follows and differs only by the factor of 2.

$$J = \frac{TP}{TP + FP + FN}$$

The treshold used in transforming the output of the model (Y) to the final boolean prediction (Y^{fin}) has lots of impact on the different metrics mentioned above. We thus decided to also try different treshold for the models:

- Treshold : 0.15,0.25,0.3

To summarize, we have thus 5 varying parameters (layers frozen, epochs, learning rate, dropout, treshold) for each model that we re-train on the new

data. We thus have in total $3 * 3 * 2 * 3 * 3 = 162$ different models that we have trained. As done for training the GAN, we trained on the Izar cluster, from SCITAS [4].
For each model, we used k-cross validation, with k = 4, to measure the final results after training. We thus trained each model 4 times on 75 % of the data, and computed Sørensen–Dice coefficient, accuracy, precision and recall on 25 % of the validation data. We then computed the average results of the k-folds. The data consisted of the 48 images of real track satellites (after data augmentation at explained in 2.3). It is important to note that the data only consisted of images containing satellites.
We chose the best model based on the highest value of the Dice coefficient, and Table 1 shows the parameters that achieved the best results.

Unfreezed layers	Epochs	Learning rate	Dropout	Treshold
Two last conv layers	60	0.01	0.2	0.15

Table 1: Parameters of the best model

It achieved a Dice Coefficient, Recall and Precision of respectively 0.58, 0.57 and 0.68. The average value of all the models was 0.53, 0.52 and 0.64.

3.4 Performance of Fine-Tuning

Having found appropriate parameters for the fine-tuning, we wanted to test how well they really performed. We took back the training set of 19 real images that we had. We separated it in a Train Set containing 14 images, and a Test Set containing 5 images. We again made the same geometric transformations (same type of transformations explained in 2.3, but slightly different as they are random) of the 14 images, making the train set 35 images. We fined-tuned the model on the 35 images using the above parameters (Table 1). In the following, model 1 refers to the model without fine-tuning, and model 2 refers to the model after fine-tuning.
To quantify our results, we compute the Dice index as defined in section 3.3. We also measure the Precision, which is defined as follows:

$$Pre = \frac{TP}{TP + FP}$$

It measures the fraction of true positives over all the pixels that were predicted as positive. In other words, how accurate is the model when predicting that a value is positive. We also use the Recall, which is defined as follows :

$$Rec = \frac{TP}{TP + FN}$$

It measures the fraction of positive that were classified as positive. Each metric lies between 0 and 1, 1 indicating a perfect score. We also computed the accuracy, but as around 99.9 % of the pixels values are 0 in the masks, the accuracy for both models is 99.9 %, and it is thus not a significant metric to estimate the performance of the models.
We present here in Table 2 the average of each metric on the train data (the 35 real images after data augmentation) after the fine-tuning (model2):

Model	Dice $\pm\sigma$	Precision $\pm\sigma$	Recall $\pm\sigma$
Model 2	0.85 ± 0.11	0.94 ± 0.09	0.79 ± 0.13

Table 2: Results on the Train Data of model 2

We also provide an example of the output of one train sample :

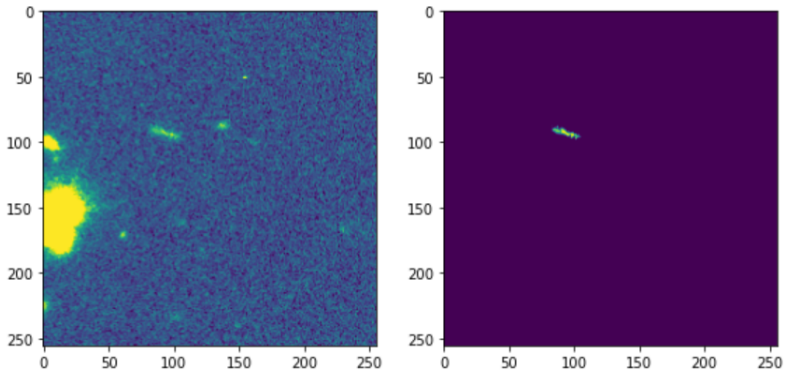
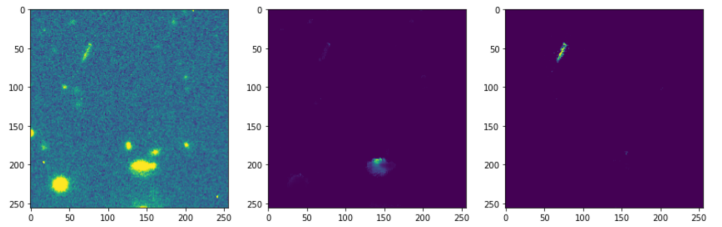


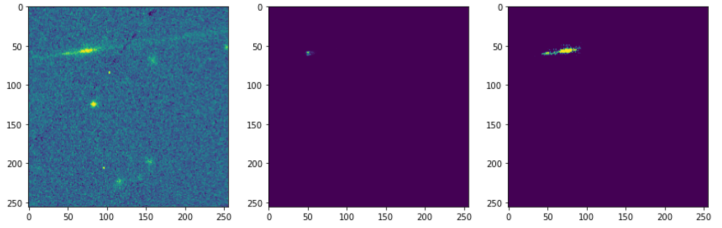
Figure 11: Prediction on a train data by model 2

The most important thing is that the network generalizes to completely new unseen data. We thus tested our model on the real test set containing real slow satellite streaks. The results are thus only on 5 images. To have more results, we thought of doing new transformations on the unseen test set in order to have more images on which to test. However, we preferred to quantify results only on 100 % real data, without any transformation. The resulting model after fine-tuning does perform very well on those 5 unseen

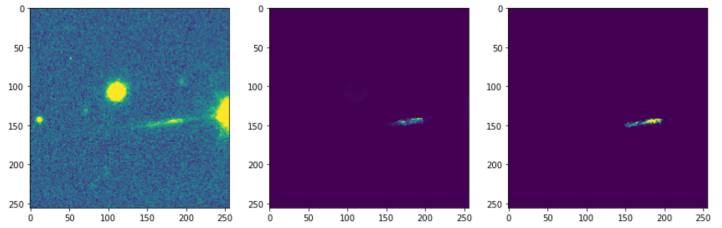
new images. Figure 12 shows the outputs of each models on the unseen test data. Figure 13 shows the output of model 2 on the test data after tresholding compared to the true target.



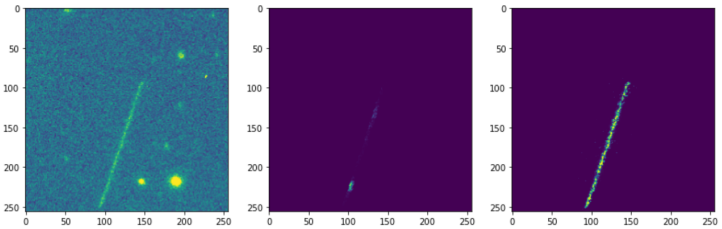
(a)



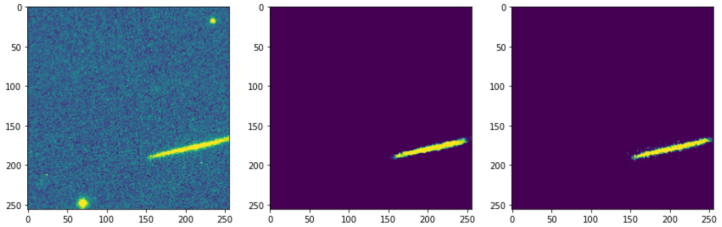
(b)



(c)

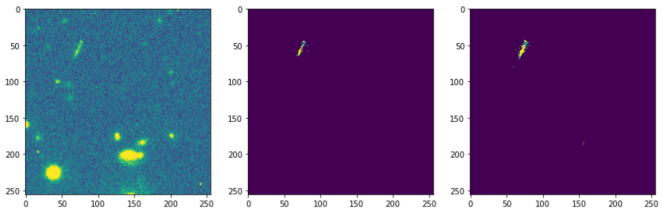


(d)

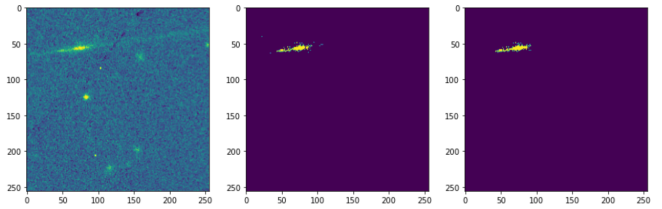


(e)

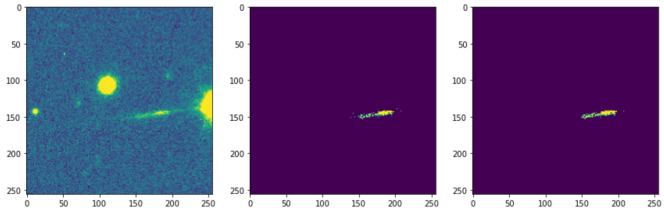
Figure 12: Prediction of both models on the test set. Model 1 in the middle and Model 2 on the right



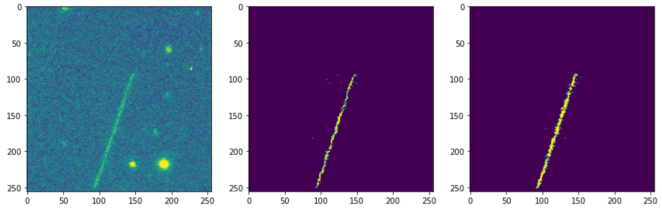
(a)



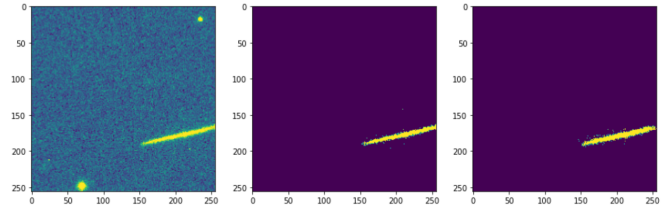
(b)



(c)



(d)



(e)

Figure 13: Final prediction on test set of model 2 after treshold. Test target in the middle, and final prediction on the right.

We can see visually that model 2 performs well on all the images, whereas the first model only performs well on the last image (e of Figure 12), where the track is very pronounced. We present in Table 3 the quantitative results obtained on the test data for both models. The results show the average Dice, Prediction and Recall for the 5 images of the test data. Having only 5 images, we also provide values for the standard deviation in order to quantify the viability of the results.

Models	Dice $\pm\sigma$	Precision $\pm\sigma$	Recall $\pm\sigma$
Model 1	0.21 ± 0.31	0.38 ± 0.32	0.20 ± 0.34
Model 2	0.82 ± 0.11	0.73 ± 0.16	0.94 ± 0.05

Table 3: Results on the test set for both models

We can see that the fine-tuning approach greatly helps to improve results, and achieves a good performance on the test set. We can see that the standard deviation is very high for model 1, which comes from the fact that model 1 performs very poorly on 4 of the 5 images with a Dice index almost 0, and performs only well on one image (e in Figure 12).

As mentioned before, fine-tuning was made using only new real images containing satellites. It is thus important to measure the performance of model 2 on real images that don't contain satellites to make sure it didn't affect the performance on negative samples. In this case, the target being only 0 as there are no tracks, TP = 0 and FN = 0. We thus have that recall, precision and dice will always be 0 for such a sample. In addition, we have

$$Acc = \frac{TN}{TN + FP}$$

and

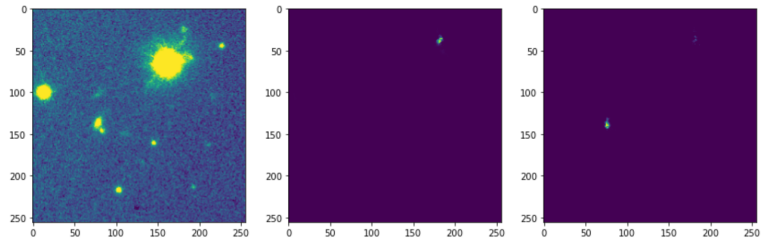
$$1 - Acc = \frac{FP}{TN + FP}$$

1-Acc thus measures the % of FP in the case of TP = 0 and FN = 0. We use those two metrics and obtain the results of Table 4, on 200 real random images. We don't provide measures for the standard deviations as there were on the order of 1e-4 and provided therefore no useful information.
Most of the labels are 0 and therefore the results are extremely good, but we can see in Table 4 that model 2 performs similarly to model 1 on real synthetic data that don't contain a satellite. We can see that fine-tuning didn't change much the predictions of images where there are no satellites,

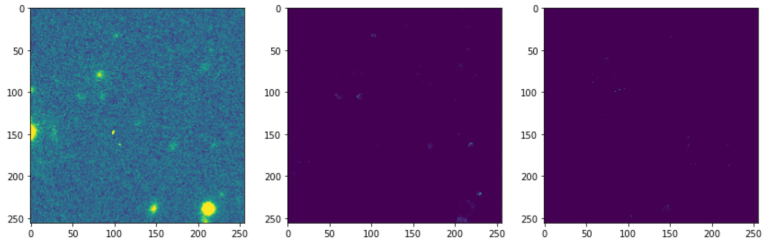
Models	Accuracy	% of FP
Model 1	0.999997	0.000041
Model 2P	0.999911	0.000585

Table 4: Results for the 4 architectures for 10 runs

and still predict very few Positive in that case.
Out of the 200 images, we plot in Figure 14 the predictions for both models on one random sample, and one sample that achieved the worst accuracy with model 2.



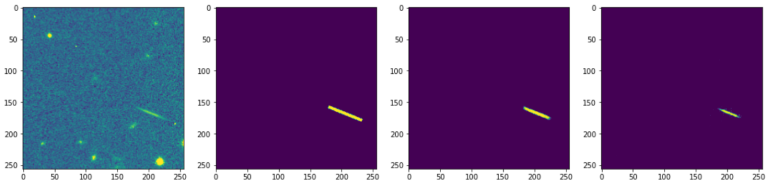
(a) Worst Accuracy



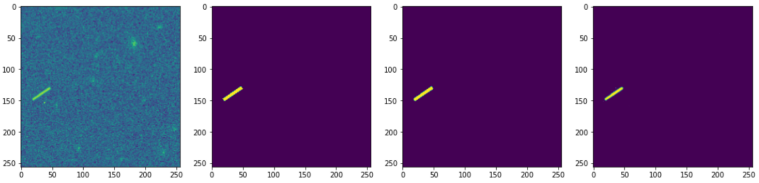
(b) Random image

Figure 14: Predictions of real images without satellite, model 1 in the middle, model 2 on the right

We have thus seen that the model 2 after fine-tuning greatly increases the performance on real data containing satellite tracks and still performs well on real data that don't contain satellite. We finally wanted to illustrate in figure 15 how both models perform on the synthetic images to illustrate how fine-tuning changed the outputs on the original task. The first image on the left of the figure is the synthetic image, then the corresponding mask, then the prediction by model 1 and finally the prediction by model 2. It is only for purpose of illustration and we don't provide quantitative results as the goal is not to have good predictions on the synthetic images



(a)



(b)

Figure 15: Prediction on synthetic images.

4 Conclusion

The first part of the project was to create the best synthetic images in order to train a NN capable of detecting real slow satellite streaks. The use of CycleGAN unfortunately didn't provide satisfactory results that would be used for the rest of the project. However, pre-training the U-NET with new synthetic images and fine-tuning the network with a small data set of real images gave very promising results. Indeed, the model after fine-tuning successfully detects real streaks on new unseen images. However, the test set remains too small to be able to give very general results.
We can thus think of several improvements that can be made in order to possibly improve results and be more confident on the performance of the resulting model. One idea would be to search for more images of real streaks, in order to train the pre-trained model 1 on more data, and more importantly to have a bigger test set in order to analyse the results more confidently.
Another idea is to analyse more deeply the transformation of $Y \longrightarrow X$ of the CycleGAN. As discussed in section 2.2.1, some of the results where it creates a streak from an image which doesn't contain one were satisfactory. However, it was only a fraction of the results and some work would be necessary in order to create the corresponding mask.
Finally, based on the CyCADA [7], we thought of combining the GAN and the U-Net in order to create images that would already have good predictions, in order to generate better images.

References

[1] Yann BOUQUET, 'Delienating Satelite Tracks in Astronomical Images' (2020)

[2] Geospatial World : https://www.geospatialworld.net/blogs/how-many-satellites-orbit-earth-and-why-space-traffic-management-is-crucial/

[3] Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks https://arxiv.org/pdf/1703.10593.pdf

[4] SCITAS, Scientific Computing resources : https://www.epfl.ch/research/facilities/scitas/

[5] Arkadiusz Nowaczynski. 'Deep learning for satellite imagery via image segmentation'. In: (2017). url: https://blog.deepsense.ai/deep-learningfor-satellite-imagery-via-image-segmentation/

[6] Morphometric analysis of white matter lesions in MR images : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=363096

[7] CyCADA : Cycle-Consistent Adversarial Domain Adaptation : http://proceedings.mlr.press/v80/hoffman18a/hoffman18a.pdf

APPENDIX

A Title of Appendix

We provide in Table 5 the coordinates of the 19 real satellite streaks. For each satellite streaks, we provide the corresponding .fits file, as well at the block number as defined in figure 16, and the coordinates of the satellite on the corresponding block. The coordinates are those of the numpy array corresponding to the block, meaning it starts at index 0 and the syntax is (line, col). Let x and y be the line and col as given in table 5, the image going from [x:x+ 256, y:y+256] will contain the satellite streak.

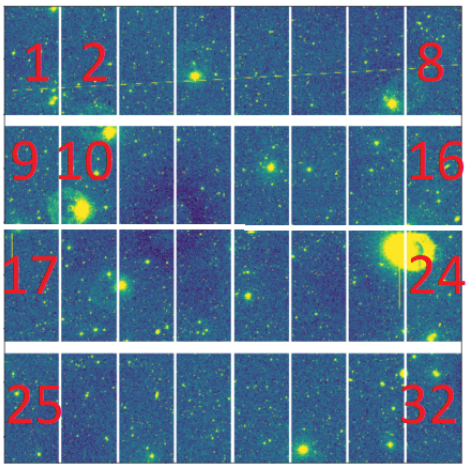


Figure 16: Numbering of the blocks on the mosaique to understand table 5

File Name (.fits)	Block Number	line	column
OMEGA.2020-01-28T03_17_31.123_fullfield_binned	1	300	1100
OMEGA.2020-01-28T03_17_31.123_fullfield_binned	2	200	400
OMEGA.2020-01-28T03_17_31.123_fullfield_binned	2	0	1750
OMEGA.2020-01-28T03_23_35.394_fullfield_binned	2	3300	1550
OMEGA.2020-01-28T03_23_35.394_fullfield_binned	30	50	1750
OMEGA.2020-01-28T03_23_35.394_fullfield_binned	29	550	700
OMEGA.2020-01-29T03_09_59.498_fullfield_binned	2	3500	1500
OMEGA.2020-01-29T03_09_59.498_fullfield_binned	26	2270	500
OMEGA.2020-01-29T03_09_59.498_fullfield_binned	26	1870	250
OMEGA.2020-01-31T03_21_17.513_fullfield_binned	3	2700	500
OMEGA.2020-01-31T03_21_17.513_fullfield_binned	20	3150	1100
OMEGA.2020-01-31T03_27_21.824_fullfield_binned	3	2700	400
OMEGA.2020-01-31T03_27_21.824_fullfield_binned	24	800	1200
OMEGA.2020-01-31T03_27_21.824_fullfield_binned	20	3100	940
OMEGA.2020-01-31T03_33_26.165_fullfield_binned	20	3050	1200
OMEGA.2020-01-31T03_33_26.165_fullfield_binned	28	3750	800
OMEGA.2020-03-18T00_34_04.673_fullfield_binned	9	2200	1700

Table 5: Coordinates of the real slow satellite streaks