GSati: Gaussian Splatting for remote sensing Super-Resolution

Adrián García Tapia

June 15, 2025

Resumen– Este artículo trata la superresolución (SR) de imágenes de satélite de Sentinel-2 mediante Gaussian Splatting (GS), una técnica novedosa en SR que aún no ha sido aplicada a imágenes multiespectrales. El objetivo principal de este trabajo es implementar el modelo GSASR, adaptarlo para tratar imágenes multiespectrales y comparar el rendimiento de la versión implementada en RGB y en multiespectral. Se describe cada bloque que conforma la arquitectura del modelo, que consiste en un *Encoder*, un *Decoder* y *Rasterizador 2D*. El trabajo busca evaluar la mejora que supone el uso de información multiespectral frente a la RGB para la SR con GS en datos Sentinel-2.

Palabras clave– Superresolución, Escala Arbitraria, Gaussian Splatting, Sentinel-2.

Abstract– This paper is about super-resolution (SR) of Sentinel-2 satellite images using Gaussian Splatting (GS), a new technique in the field of SR that is yet to be applied to multispectral images. The main goal of this work is to implement the GSASR model, adapt it to multispectral images, and compare the performance of both versions. Each block that composes the architecture is described, which consists in an *Encoder*, a *Decoder* and a *2D Rasterizer*. This work evaluates the improvement of using multispectral information versus just RGB for SR with GS in Sentinel-2 data.

Keywords– Superresolution, Arbitrary-Scale, Gaussian Splatting, Sentinel-2.

1 INTRODUCTION

IMAGE super-resolution (SR) is a technique that enhances the original resolution of an image. It is applied to different situations such as regular images taken with our phones or cameras, videogames to increase their performance without losing graphics quality, and to generate additional data from images. The quantity and quality of the generated data can improve the accuracy of algorithms for various tasks such as detection, segmentation, or classification tasks.

However, the 3D reconstruction landscape has seen a lot of advancement in the past few years. New novel view synthesis techniques like NeRF [1] and more recently Gaussian Splatting (GS) [2] can represent 3D scenes or 3D objects from images taken from different points of view. Although these techniques are meant for 3D tasks, they can be extrapolated to 2D tasks as well, since they have great representational capabilities.

In recent times, both of these 3D techniques have been used for the SR task, first with Implicit Neural Representations (INR) from which NeRFs come from, and more recently with GS. These techniques allow for arbitrary scale super-resolution (ASSR), which means they can work with even decimal scales such as $\times 2.5$. Nowadays, there are only three papers on the GS for SR topic: Hu et al. 2024 [3], Chen et al. 2025 [4] and Peng et al. 2025 [5]. This project implements the architecture proposed by Peng et al. 2025, and applies it to multispectral (MS) images.

After the next subsection, the document is structured as follows. Section 2 reviews the State of the Art (SotA) of the different SR techniques, both in RGB and MS images. Section 3 explains the development process of this project and the tools used for that. Section 4 focuses on the training procedure followed for this algorithm and the data preparation used for each version of the model. Section 5 presents and discusses the final results obtained. Finally, Section 6 reviews the results of this paper and proposes ideas for future work.

1.1 Objectives, Plan and Methodology

The target of this project is to apply these novel SR techniques using GS to MS images, which has not been explored yet. In particular, we focus on satellite images with multiple bands, such as Sentinel-2 [6], a constellation of 3 satellites from the Copernicus Project that monitors Earth through 13 different bands of the electromagnetic spectrum.

To achieve this goal, I did a Gantt diagram to plan each task and used Kanban as the methodology through the entire project.

• E-mail de contacte: adrian.garciat@autonoma.cat
• Menció realitzada: Computació
• Treball tutoritzat per: Felipe Lumbreras Ruiz (Ciències de la Computació)
• Curs 2024/25

The tasks and planning can be found in the dossier.

2 State of the Art

2.1 Super-Resolution

SR is divided in fixed-scale super-resolution (FSSR) and arbitrary-scale super-resolution (ASSR), and in single-image super-resolution (SISR) and multiple-image super-resolution (MISR). For this project, we focus on ASSR and SISR.

ASSR is mostly done with a single image, so it also fits in the SISR cathegory. This consists on training a single model to increase the resolution of an image in any scaling factor, like $\times 2$, $\times 3$, $\times 4$ and so on, including any in-between factor like $\times 2.5$. Meta-SR [7] is a method based on interpolation operators and was the first deep learning method capable of doing ASSR. However, the State of the Art (SotA) has been populated by methods based on Implicit Neural Representations (INR), inspired by the success they had in 3D reconstruction tasks such as NeRFs [8]. LIIF [9] was the first to adapt INR methods to the ASSR task, and others followed, like ITSRN [10], SRNO [11] and CiaoSR [12], being the last one the current SotA of INR methods for ASSR.

2.2 Gaussian Splatting Super-Resolution

Once more inspired by 3D reconstruction tasks, GS [2] has started to be explored for ASSR tasks given its success against the NeRF methods.

GaussianSR [3] was the first to take this technique to the ASSR task. Despite being surpassed by CiaoSR, it outperforms LIIF and ITSRN both in image quality and inference speed. GSASR [4] was the next advance in this area, proposing a different architecture which includes a 2D rasterizer implemented in C++/CUDA for faster inference. It performs better than CiaoSR in final image quality and inference time. Finally, the most recent of all is ContinuousSR [5], which outperforms all previous methods in image quality and has an surprisingly fast inference time, beating other methods by orders of magnitude depending on the dataset.

2.3 Sentinel-2 Super-Resolution

In remote sensing (RS) SR methods vary depending on the target satellite. In this specific case, we focus on the Sentinel-2 satellite, a constellation of 2 satellites from the Copernicus Program of the European Space Agency (ESA) that capture images from 13 different bands of the electromagnetic spectrum ranging from the 443 nm to 2190 nm. These bands are captured in different resolutions: 10 m/px, 20 m/px, 60 m/px [6].

Some SR methods focus on taking all bands to 10 m/px as the target resolution. HFN [13] is a convolutional method that uses information from all channels to take them to 10 m/px in a hierarchical way. The transformer proposed by Sharifi et. al. 2025 [14] is a recent method that also achieves the same goal.

On the other hand, there are algorithms to increase the resolution beyond those 10 m/px, though they usually focus more on the RGB bands rather than fusing information from all bands, leaving aside those channels that are not originally in 10 m/px. SENX4 [15] applies a $\times 4$ resolution increase to all 10 m/px bands using a convolutional neural network. RS-ESRGAN [16] applies a $\times 5$ upscaling to the 10 m/px bands. As the ESRGAN [17] it is based off, it uses a Generative Adversarial Network (GAN) to do the SR.

If we go to the hyperspectral field, DMGASR [18] is a diffusion technique that allows to use a lot of channels from an image to do the SR, which could be applied to Sentinel-2 images.

3 Development

3.1 GSASR

Since the goal of this project uses the GSASR [4] algorithm for Sentinel-2 MS images and at the beginning of this project there was no code available, only the instructions from their paper, we had to implement all of it from scratch, implementing each part of the algorithm based on their text and similar projects. We also had to prepare the necessary datasets and train the model with them.

To implement the algorithm, we used PyTorch 2.5.1 for Python 3.10.14. Since the rasterizer is implemented as a C++/CUDA extension for PyTorch, we used the PyTorch version compiled with the CUDA version 12.4, which is the same CUDA version used to compile the rasterizer.

The structure of the algorithm is shown in Figure 1, and is explained in more detail in the original paper [4]. The only change to the final architecture lies in the 2D Rasterizer. Instead of doing the algorithm they propose, we switched the order of the for-loops so each CUDA thread represents a gaussian and iterates through all the pixels in the image and added the number of channels of the image as input, as seen in Algorithm 1.

Equations (1) and (2) are the same gaussian equations as the original. μ and σ are the mean and standard deviation of the gaussian, the x and y axes respectively. ρ is the correlation of x and y, which describes the shape and orientation of the gaussian along with σ. Δx and Δy are the distances in both axes from the evaluated point to the gaussian center μ. Finally, α is the opacity of the gaussian and c is a vector describing the color of the gaussian in each channel of the image. Next, we have both equations:

$$f(x,y) = \left(2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}\right)^{-1} \exp\left[-\frac{1}{2(1-\rho^2)} \times \left(\frac{\Delta x^2}{\sigma_x^2} - \frac{2\rho\Delta x\Delta y}{\sigma_x\sigma_y} + \frac{\Delta y^2}{\sigma_y^2}\right)\right], \tag{1}$$

$$G(x,y) = \alpha c f(x,y) \ . \tag{2}$$

Algorithm 1 2D Rasterizer algorithm

Input: N (2D) Gaussians $\{G_1, G_2, ..., G_N\}$; LR image of size (H, W); scale factor s; raster ratio r; number of channels C.
Output: Renderized image I_{SR}.

```
 1: Initialize $I_{SR}$ as a zero array of dimensions $(sH, sW, C)$.
 2: for each $G_i$ in $\{G_1, G_2, ..., G_N\}$ do
 3:     Initialize $\alpha, \mu_x, \mu_y, \sigma_x, \sigma_y, \rho, c$ from $G_i$.
 4:     for each pixel $(x, y)$ in $I_{SR}$ do
 5:         if $|x/s - \mu_x| < rH$ and $|y/s - \mu_y| < rW$ then
 6:             Obtain $f(x/s, y/s)$ using Eq. (1);
 7:             Obtain $G_i(x/s, y/s)$ using Eq. (2);
 8:             $I_{SR}(x, y; s) += G_i(x/s, y/s)$
 9:         end if
10:     end for
11: end for
```

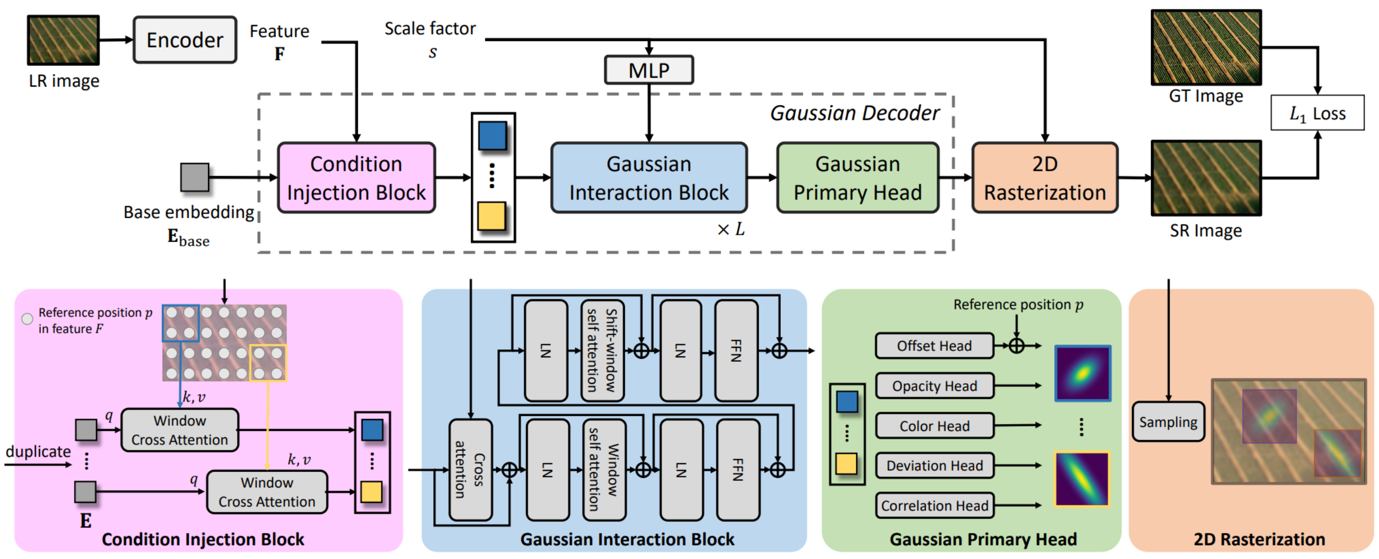


Fig. 1: GSASR architecture, extracted from the original paper. It follows an Encoder-Decoder architecture with a Rasterizer at the end. The Encoder is a backbone and extracts features from the input image. The Decoder generates Gaussian Embeddings and fuses the information from these embeddings and the input image features with various attention mechanisms, to generate the final parameters of each gaussian that represents the image. (*Source: Generalized and efficient 2d gaussian splatting for arbitrary-scale super-resolution, 2025*)

3.2 HFN

Each band from a Sentinel-2 image is saved in a different raster file, and they cannot be directly combined as each can have a different resolution like 10 m/px, 20 m/px and 60 m/px. Since our model needs the input to be a single tensor with C channels, every band of the image had to be combined into a single MS image. So instead of downsampling them to the lowest resolution of 60 m/px, we used our own implementation of the HFN model [13] as there is no code publicly available as well. However, we did some changes to the final implementation. The original model upsamples via pansharpening the 20 m/px to 10 m/px using the original 10 m/px as a help, and then does the same for the 60 m/px with the original 10 m/px. In our implementation, when pansharpening the 60 m/px bands, we also use the information of the upscaled 20 m/px bands along with the original 10 m/px bands, since it is extra information that can help the model refine the final output.

4 Experiments

We performed three different experiments with this model: one with the DIV2K dataset [19], another with only the Sentinel-2 RGB bands, and the third with all available Sentinel-2 bands-2. To monitor these experiments, we used the Peak Signal-to-Noise Ratio (PSNR) as the main metric, and also Structural Similarity Index Measure (SSIM), Learned Perceptual Image Patch Similarity (LPIPS) and Deep Image Structure and Texture Similarity (DISTS) as secondary metrics.

4.1 DIV2K Training

For this experiment, we used the DIV2K dataset [19], a widely used dataset for the SR task since it was introduced in 2017. To prepare the data, we followed the same procedure described in the original GSASR paper [4], which is the standard procedure for ASSR trainings with this dataset as seen in [12] and [9] too.

We downloaded the training set composed of 800 high quality images and the validation set made of 100 high quality images, they are the DIV2K800 and DIV2K100 respectively. To preprocess the data on demand, we randomly crop patches of $48s \times 48s$ pixels, where s is the scale factor randomly chosen between $Scales = \{1, 2, 3, 4\}$ at each iteration. Then, the generated patch is downsampled using bicubic interpolation to the size 48×48 and fed to the neural network along with the chosen scale factor s.

The neural network is configured in the same way with embeddings of size 180, a window size fixed to 12, 4 attention heads where it does attention, 6 stacked gaussian interaction blocks, a rasterization ratio of 0.1 and a density of gaussians m of 16. For the inner MLPs, we also used their MLP decrease ratio of 4. All experiments were conducted using ESDR [20] with an inner feature size of 64 and 16 residual blocks as the backbone.

We tried to use the same training hyperparameters from the original paper [4], however, since our implementation is less memory efficient, we had to change some of them accordingly. We decided to set the batch size to 9, the learning rate to $7.5e-5$, 2000 warm-up iterations followed by a scheduler with a decay factor of 0.5 at iterations 500000, 800000, 900000 and 950000. In the last decay step, early stopping kicks in, using the PSNR to decide when to stop training. We also used mixed precision for a more efficient training, and gradient clipping to ensure gradient stability in the whole process.

Each training was done in an ADA 6000 GPU with 48 GB of VRAM for seven days.

4.2 Sentinel-2 Training

The second model was trained on the RGB bands of Sentinel-2 images. Since we couldn't find any SR dataset for all Sentinel-2 bands at the same time and with all of them at the same initial resolution, we built it ourselves. We downloaded 5 different Sentinel-2 images using the Copernicus Browser [21]. These images are from:

- Around Pyhäjärvi, Finland 29/06/2022 (S2B_MSIL2A_20220629T100029_N0510_R122 _T35VML_20240630T200504);
- Catalonia, Spain 27/11/2024

(S2B_MSIL2A_20241127T104309_N0511_R008 _T31TDG_20241127T131852);

- Aragon, Spain 30/11/2024 (S2B_MSIL2A_20241130T105329_N0511_R051 _T30TXK_20241130T132639);
- Spanish-France border 30/11/2024 (S2B_MSIL2A_20241130T105329_N0511_R051 _T31TCH_20241130T132639);
- Northern Australia 16/05/2025 (S2C_MSIL2A_20250516T014601_N0511_R074 _T52KBG_20250516T045613).

We used three of these images to create the training dataset, one for the validation set and another for the test set. We cropped these images to feed them to our implementation of the HFN [13] model to take all bands to the 10 m/px resolution and use these images as train, validation and test sets. For the GSASR training we did the same thing as with the DIV2K dataset, we crop the new images in patches of $48s \times 48s$ pixels, where s is the scale factor randomly chosen between $Scales = \{1, 2, 3, 4\}$ at each iteration. Then, the generated patch is donwsampled using bicubic interpolation to the size 48×48 and fed to the neural network along with the chosen scale factor. For the RGB version of the model, we only passed it the RGB bands of the image, and for the MS variation, we used all bands.

These two neural networks have the same configuration as the DIV2K version. The only change lies in the MS model, which is set to take 12 channels instead of 3 as input. The EDSR [20] backbone is trained with the model instead of starting from a pre-trained version of it, as there are no weights for 12 channels for the EDSR.

We used the same training hyperparameters for these two models. We fixed a batch size of 9 due to GPU memory constraints, a learning rate of $7.5e-5$, 2000 warm-up iterations, a decay factor of 0.5 which activated at iterations 500000, 800000, 900000 and 950000. After all the decay milestones have been reached, early stopping kicks in to stop the training when it considers, focusing in the PSNR from the validation set to decide when the model has stopped learning.

The RGB model was trained on an L40S GPU with 48 GB of VRAM and the MS model was trained on an ADA 6000 GPU with 48 GB of VRAM for seven days.

4.3 Original training

A few weeks before turning in the final report of this project, the code of the original paper [4] was released, so we decided to train it as well to compare results. However, there are a few things to clarify. For this training, we used the original training code as is, without changing any hyperparameter. Due to a lack of available GPUs, this training has been done with a single ADA 6000 with 48 GB of VRAM, so the total batch size is 14 instead of the original 64.

5 Results

In this section we report the results obtained from the various experiments we conducted.

5.1 DIV2K

TODO: Write this subsection as soon as the models finish training.

Fig. 2: Visual comparison of the original GSASR and our implementation.

5.2 Sentinel-2

Due to the MS nature of these images, we decided to test the results in two different ways. The first of them are shown in Table 2, where only the RGB bands are used to calculate every metric. As we can see, the same model with the same hyperparameter training gets a higher or similar result in all metrics except PSNR. On the other hand, Table 2 shows the same results, but with all channels available in each case. Since LPIPS and DISTS are metrics coming from a neural network and they are not pre-trained with multispectral data, we limited these results to the PSNR and SSIM metrics.

In Figure 3 is a visual comparison of both models and we can see that there is some blurriness in the results, although it being a $\times 12$ scale factor. This may be because we deviated from the original training hyperparameters for these models. We also think that the low range of the pixel values from these images may be affecting in a slower training and the loss function may need to be modified.

5.3 Metrics Analysis

Results shown in Tables 2 and 3 differ a lot from those in Table 1. The numbers obtained from the models trained on Sentinel-2 images are better than those trained in DIV2K800 while the visual quality we can observe in Figures 2 and 3 are quite similar. We suspect a reason for that behaviour lies in the way PSNR and SSIM are calculated and the range of values from Sentinel-2 images.

Images taken from Sentinel-2 are encoded in a raster with int12 values and are normalized dividing by 10000 as stated by the official documentation [6]. This results in images in floating point, commonly represented with values from 0 to 1, but that in this case can even go up to 2 instead of 1. This is not an issue since values higher than 1 are usually clouds and can be clamped to to the range 0 to 1. However, those high values are outliers and most of the image is between the range 0 to 0.3 or 0.5. This just happens in RGB bands, so the variability in value of MS bands are more widespread in that 0 to 1 range.

If we take a look at how PSNR is calculated according to Equation (4), it relies on MSE, and the lower the MSE value is, the higher the PSNR goes. In the case of Sentinel-2 images, once the model learns to predict between the most common range of values, the PSNR grows as the MSE as shown in Equation (3) is lower than in a natural image. It could explain why in both Sentinel-2 models results are higher than in the DIV2K800 variants and also why it has even higher values when using only the RGB bands of the image. Here are the Equations for computing PSNR and MSE, where n is the number of images, y_i is the observed value and p_i is the predicted value for y_i and MAX_I is the maximum value of the image, which in this case is always 1:

$$\text{MSE} = \frac{1}{n}\sum (y_i - p_i)^2 \ , \tag{3}$$

$$\text{PSNR} = 10 \times \log_{10}\left(\frac{\text{MAX}_I^2}{\text{MSE}}\right) \ . \tag{4}$$

SSIM is a similar case. In Equation (5), we can see that to get a high result, we need to decrease the variance of the pixels, which grow faster in the denominator than in the nominator. This

	PSNR ↑		SSIM ↑		LPIPS ↓		DISTS ↓	
	GSASR-O	GSASR-C	GSASR-O	GSASR-C	GSASR-O	GSASR-C	GSASR-O	GSASR-C
×2	0	0	0	0	0	0	0	0
×3	0	0	0	0	0	0	0	0
×4	0	0	0	0	0	0	0	0
×6	0	0	0	0	0	0	0	0
×8	0	0	0	0	0	0	0	0
×12	0	0	0	0	0	0	0	0
×16	0	0	0	0	0	0	0	0
×18	0	0	0	0	0	0	0	0
×24	0	0	0	0	0	0	0	0
×30	0	0	0	0	0	0	0	0
Best	∞		1		0		0	

Table 1: Performance comparison of GSASR-O and GSASR-C at different scaling factors.

	PSNR ↑		SSIM ↑		LPIPS ↓		DISTS ↓	
	GSASR-RGB	GSASR-MS	GSASR-RGB	GSASR-MS	GSASR-RGB	GSASR-MS	GSASR-RGB	GSASR-MS
×2	**44.06**	43.60	0.9721	**0.9726**	0.1081	**0.0977**	0.0479	**0.0406**
×3	**41.21**	40.86	**0.9503**	**0.9503**	0.1798	**0.1611**	0.1398	**0.1281**
×4	**39.72**	39.53	0.9305	**0.9314**	0.2449	**0.2155**	0.2291	**0.2016**
×6	**37.34**	37.28	0.9027	**0.9040**	0.3288	**0.2969**	0.3890	**0.3276**
×8	**36.54**	36.51	0.8880	**0.8887**	0.3864	**0.3571**	0.5220	**0.4573**
×12	**35.11**	34.93	**0.8788**	**0.8788**	0.4642	**0.4527**	0.6708	**0.6426**
Best	∞		1		0		0	

Table 2: Performance comparison between GSASR-RGB and GSASR-MS at different scaling factors. These results are calculated over the RGB bands only, as LPIPS and DISTS are not trained on multiespectral data. The best results in each metric are highlighted in **bold**.

	PSNR ↑		SSIM ↑	
	GSASR-RGB	GSASR-MS	GSASR-RGB	GSASR-MS
×2	**44.06**	40.33	**0.9721**	0.9697
×3	**41.21**	36.20	**0.9503**	0.9277
×4	**39.72**	34.10	**0.9305**	0.8850
×6	**37.34**	31.38	**0.9027**	0.8152
×8	**36.54**	30.14	**0.8880**	0.7749
×12	**35.11**	28.63	**0.8788**	0.7443
Best	∞		1	

Table 3: Performance comparison between GSASR-RGB and GSASR-MS at different scaling factors. These results are calculated over the RGB bands in the GSASR-RGB model and over the all bands for the GSASR-MS. The best result in each metric is highlighted in **bold**.

produces the same effect than it did in the calculation of PSNR. Here we have the equation where x and y are the 2 samples to compare, μ_x is the pixel sample mean of x, μ_y is the pixel sample mean of y, σ_x^2 is the sample variance of x, σ_x^2 is the sample variance of y, σ_{xy} is the sample covariance of x and y, and c_1 and c_2 are 2 variable to stabilize the division with weak denominator:

$$\mathrm{SSIM} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_x^2 + c_2)} . \tag{5}$$

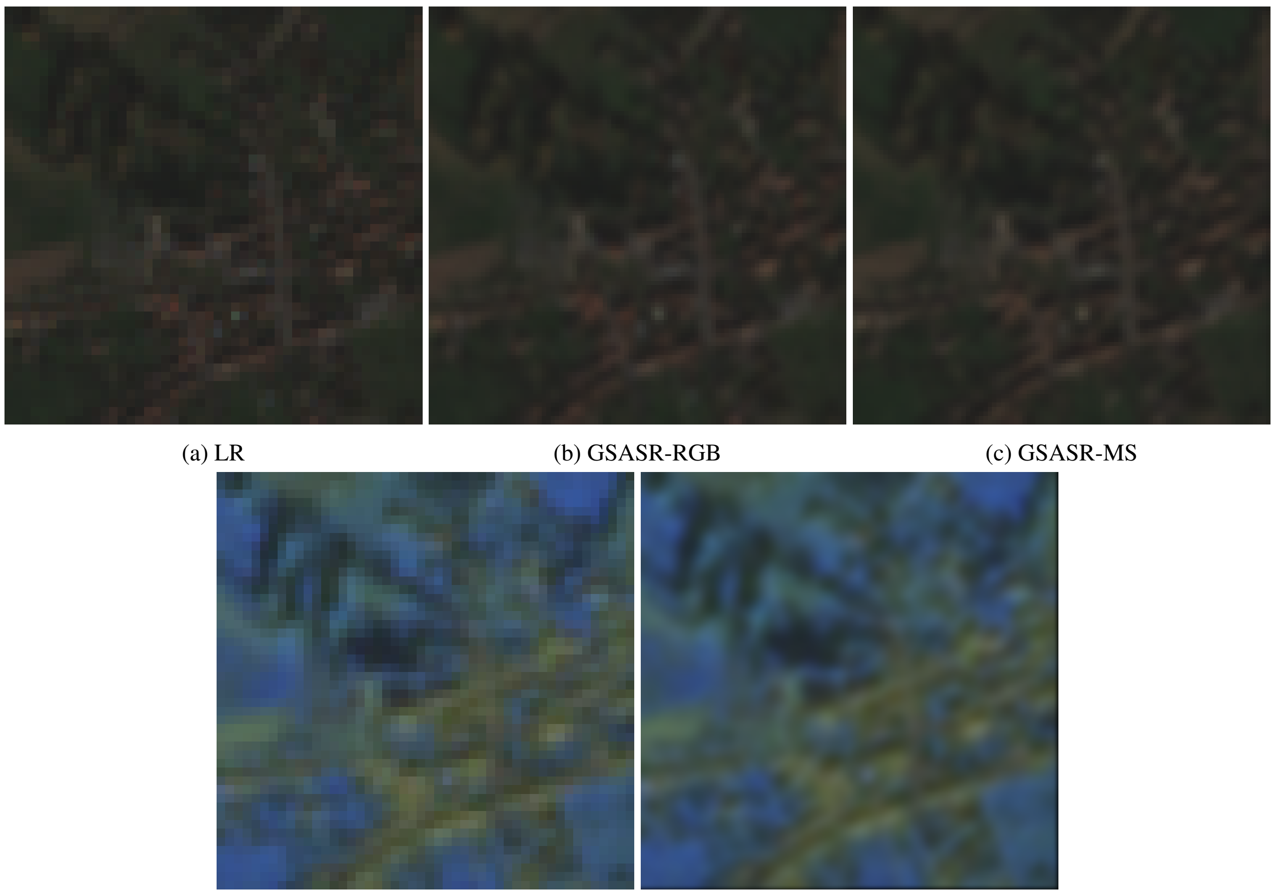
6 Conclusion

We presented our implementation of GSASR [4], a neural network that adapts GS to ASSR. Our implementation uses the same architecture as the original despite being less memory efficient. GS has shown a lot of representative potential in 3D scenes and has now made its way into 2D images, both RGB and MS.

6.1 Future Work

As for future work, this model and some others are quite limited in the format of the input data they receive. For example, in this architecture we need all bands from Sentinel-2 to be of the same size, so we have to rely in another model to take all of them to a common resolution. However, because GS is so good at representing the world, it should be possible to use different channels from an input image at different resolutions to refine the GS and improve the quality of the overall image.

This idea goes beyond that and it may be possible to mix in-


(a) LR (b) GSASR-RGB (c) GSASR-MS

(d) LR (e) GSASR-MS

Fig. 3: Visual comparison of GSASR-RGB and GSASR-MS on a $\times 12$ scale factor. Top row is the RGB image, available in both models. Bottom row is the image in 3 random MS channels. In both cases, the left image is the LR image and the other images are the output of each model.

formation from different sensors and codify them into the same gaussian space, turning it into a continuous gaussian space where all information could be represented in the same way no matter its origin.

7 Acknowledgements

Thanks to the Computer Vision Center (CVC) for providing me the necessary resources to do this project, as well as for offering a comfortable working space with a nice environment.

I want to thank Arnau Marcos for his help in optimizing the C++/CUDA code of the model and to the rest of my CVC colleagues such as Jordi Ventosa, Paula Font and Gerard Asbert for discussing ideas with me and motivating me through the process of developing this project. They have also been of help when identifying some issues with the code that I did not catch.

Finally, thanks to Felipe Lumbreras for being my tutor in this project through all these months, helping me with doubts I had at specific points and being through the whole process since the start. Also I want to thank him for suggesting this novel idea as the topic of my final thesis, which I have enjoyed and suffered, but that has helped me learn a lot and grow my motivation in Computer Vision in general.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, *Nerf: Representing scenes as neural radiance fields for view synthesis*, 2020. arXiv: 2003.08934 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2003.08934.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, *3d gaussian splatting for real-time radiance field rendering*, 2023. arXiv: 2308.04079 [cs.GR]. [Online]. Available: https://arxiv.org/abs/2308.04079.

[3] J. Hu, B. Xia, B. Chen, W. Yang, and L. Zhang, *Gaussiansr: High fidelity 2d gaussian splatting for arbitrary-scale image super-resolution*, 2024. arXiv: 2407.18046 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2407.18046.

[4] D. Chen, L. Chen, Z. Zhang, and L. Zhang, *Generalized and efficient 2d gaussian splatting for arbitrary-scale super-resolution*, 2025. arXiv: 2501.06838 [eess.IV]. [Online]. Available: https://arxiv.org/abs/2501.06838.

[5] L. Peng, A. Wu, W. Li, *et al.*, *Pixel to gaussian: Ultra-fast continuous super-resolution with 2d gaussian modeling*, 2025. arXiv: 2503.06617 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2503.06617.

[6] E. S. Agency, *Sentinel-2 User Handbook*, 2015, pp. 51–54. [Online]. Available: https://sentinels.copernicus.eu/documents/247904/685211/Sentinel-2_User_Handbook.

[7] X. Hu, H. Mu, X. Zhang, Z. Wang, T. Tan, and J. Sun, *Meta-sr: A magnification-arbitrary network for super-resolution*, 2019. arXiv: 1903.00875 [cs.CV]. [On-

line]. Available: https://arxiv.org/abs/1903.00875.

[8] K. Gao, Y. Gao, H. He, D. Lu, L. Xu, and J. Li, *Nerf: Neural radiance field in 3d vision, a comprehensive review*, 2023. arXiv: 2210.00379 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2210.00379.

[9] Y. Chen, S. Liu, and X. Wang, *Learning continuous image representation with local implicit image function*, 2021. arXiv: 2012.09161 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2012.09161.

[10] J. Yang, S. Shen, H. Yue, and K. Li, *Implicit transformer network for screen content image continuous super-resolution*, 2021. arXiv: 2112.06174 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2112.06174.

[11] M. Wei and X. Zhang, *Super-resolution neural operator*, 2023. arXiv: 2303.02584 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2303.02584.

[12] J. Cao, Q. Wang, Y. Xian, *et al.*, *Ciaosr: Continuous implicit attention-in-attention network for arbitrary-scale image super-resolution*, 2023. arXiv: 2212.04362 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2212.04362.

[13] J. Wu, L. Lin, C. Zhang, T. Li, X. Cheng, and F. Nan, “Generating Sentinel-2 all-band 10-m data by sharpening 20/60-m bands: A hierarchical fusion network,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 196, pp. 16–31, 2023, ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2022.12.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271622003331.

[14] A. Sharifi and M. M. Safari, “Enhancing the spatial resolution of sentinel-2 images through super-resolution using transformer-based deep-learning models,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 18, pp. 4805–4820, 2025. DOI: 10.1109/JSTARS.2025.3526260.

[15] M. Galar, R. Sesma, C. Ayala, L. Albizua, and C. Aranda, “Super-resolution of sentinel-2 images using convolutional neural networks and real ground truth data,” *Remote Sensing*, vol. 12, no. 18, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12182941. [Online]. Available: https://www.mdpi.com/2072-4292/12/18/2941.

[16] L. Salgueiro Romero, J. Marcello, and V. Vilaplana, “Super-resolution of sentinel-2 imagery using generative adversarial networks,” *Remote Sensing*, vol. 12, no. 15, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12152424. [Online]. Available: https://www.mdpi.com/2072-4292/12/15/2424.

[17] X. Wang, K. Yu, S. Wu, *et al.*, *Esrgan: Enhanced super-resolution generative adversarial networks*, 2018. arXiv: 1809.00219 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1809.00219.

[18] Z. Wang, D. Li, M. Zhang, H. Luo, and M. Gong, “Enhancing hyperspectral images via diffusion model and group-autoencoder super-resolution network,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 6, pp. 5794–5804, Mar. 2024, ISSN: 2159-5399. DOI: 10.1609/aaai.v38i6.28392. [Online]. Available: http://dx.doi.org/10.1609/aaai.v38i6.28392.

[19] E. Agustsson and R. Timofte, “Ntire 2017 challenge on single image super-resolution: Dataset and study,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jul. 2017.

[20] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, *Enhanced deep residual networks for single image super-resolution*, 2017. arXiv: 1707.02921 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1707.02921.

[21] “Copernicus Browser,” Copernicus Browser. (), [Online]. Available: https://browser.dataspace.copernicus.eu/ (visited on 06/11/2025).

APÉNDICE

O1(T)	O2(F)	O3(P)
Implementar RGB y Multiespectral.	Implementar RGB.	Crear repositorio de GitHub.
		Entender Condition Injection Block.
		Entender Gaussian Interaction Block.
		Entender Gaussian Primary Head.
		Entender 2D Rasterization Block.
		Aprender a combinar PyTorch+cuda.
		Estudiar las implementaciones en cuda de GS.
		Implementar Condition Injection Block.
		Implementar Gaussian Interaction Block.
		Implementar Gaussian Primary Head.
		Implementar 2D Rasterization.
		Preparar EDSR feature extractor.
		Conectar todos los bloques para el modelo final.
		Crear bucle de entrenamiento.
		Preparar Dataset RGB.
		Entrenar modelo.
		Comparar.
	Implementar MS.	Adaptar Condition Injection Block a MS.
		Adaptar Gaussian Interaction Block a MS.
		Adaptar Gaussian Primary Head a MS.
		Adaptar 2D Rasterization a MS.
		Adaptar EDSR feature extractor a MS.
		Preparar Dataset MS para EDSR.
		Preparar Dataset MS para GSASR.
		Crear bucle de entrenamiento para EDSR.
		Entrenar EDSR.
		Entrenar GSASR.
		Comparar.
Comparar con SotA Multiespectral	Comparar CiaoSR.	Preparar CiaoSR para MS.
		Crear bucle de entrenamiento.
		Entrenar CiaoSR.
		Comparar.
	Comparar Transformer SotA.	Preparar Transformer para MS.
		Crear bucle de entrenamiento.
		Entrenar Transformer.
		Comparar.
	Comparar Difusión SotA.	Preparar modelo para MS.
		Crear bucle de entrenamiento.
		Entrenar modelo de Difusión.
		Comparar.
Combinar HFN con GSASR.	HFN + GSASR.	Preparar HFN.
		Crear bucle de entrenamiento.
		Entrenar HFN.
		Comparar.

Table 4: TABLA DE OBJETIVOS.