INFO8010: Deep Learning Project:
Downscaling of weather prediction

Samy Mokeddem, **Nicolas Giourgas** and **Andréas Coco**

May 2024

1 Introduction

The conventional approach to forecasting weather in a region involves utilizing a Global Climate Model (GCM), which generates forecast grids typically at a resolution ranging between 25 and 100 kilometers. However, this resolution is insufficient for achieving accurate regional forecasts. Consequently, various empirical methods have been developed to downscale these predictions to higher resolutions. These empirical models can either be based on physical or statistical principles. While physical models tend to offer greater accuracy, they also demand higher computational resources compared to statistical models. Machine learning, particularly Deep Learning (DL) and diffusion models, present novel approaches to address this downscaling challenge. This report details our implementation of a diffusion model specifically designed for downscaling weather predictions in Belgium, with a focus on enhancing wind speed forecasts.

We begin with a review of related work in this field. Next, we introduce the dataset utilized throughout our project. Following this, we outline our methodology, including a detailed description of our model architecture, its training process, and tuning procedures. Subsequently, we assess the model's performance and provide some recommendations for further improvement. Finally, we conclude the report, summarizing our findings and their implications.

2 Related works

High-resolution downscaling with DL is an active area of research. High-resolution downscaling models can be viewed as a subset of super-resolution models. Super-resolution models enhance image resolution beyond the original quality, often using DL techniques. They learn mappings from low-resolution inputs to high-resolution outputs, with principal applications in image enhancement and medical imaging.

The main difference between high-resolution downscaling models and super-resolution models lies in their fields of application. High-resolution downscaling is typically used in geological applications, such as weather forecasting, whereas super-resolution models are applied more broadly across various fields.

As highlighted in [5], super-resolution tasks have evolved significantly with advancements in deep learning. Initially, these tasks were accomplished using Convolutional Neural Networks (CNNs), which provided an end-to-end mapping from low-resolution to high-resolution images. For instance, the paper [6] presents various CNN models designed to downscale precipitation observations at the regional scale, specifically focusing on New Zealand.

Following CNNs, Generative Adversarial Networks (GANs) became the most widely used tools in SR tasks. GANs use a generator-discriminator framework to produce high-resolution images that appear sharper and more detailed. Despite their success, GANs are known to exhibit major drawbacks like mode collapse, high computational demands, and convergence issues. Consequently, they require complex stabilization methods during training.

Normalizing flows also contributed to SR by enabling invertible transformations and providing exact likelihood estimation, thus enhancing image quality. However, their impact went somewhat unnoticed because of the emergence of diffusion models.

As of the writing of this report, the state-of-the-art models in image upscaling are diffusion models. Diffusion models represent a disruptive advancement in SR tasks. Their superior image quality and easier training process make them a preferable choice for generating high-quality images compared to GANs. However, diffusion models come with their own set of challenges which include high computational demands and color shifts.

The paper [1] introduces a library for the general task of empirical downscaling using deep learning, with an example focusing on European atmospheric NO2 concentration as shown in Figure 1.

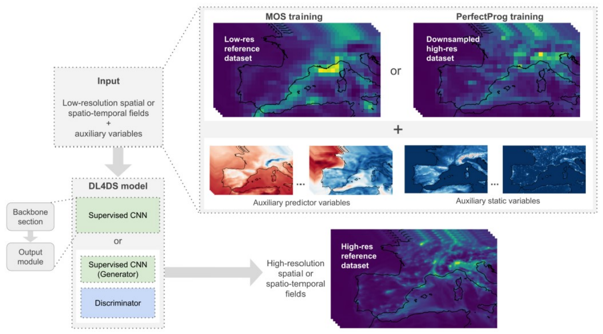

MOS training
Low-res reference dataset
or
PerfectProg training
Downsampled high-res dataset
Input
Low-resolution spatial or spatio-temporal fields + auxiliary variables
+
Auxiliary predictor variables
Auxiliary static variables
DL4DS model
Backbone section
Output module
Supervised CNN
or
Supervised CNN (Generator)
Discriminator
High-resolution spatial or spatio-temporal fields
High-res reference dataset

Figure 1: The general architecture of DL4DS [1]. A low-resolution gridded dataset can be downscaled, with the help of auxiliary predictor and static variables, and a high-resolution reference dataset. The mapping between the low- and high-resolution data is learned with either a supervised or a conditional generative adversarial DL model.

The paper that most inspired the approach adopted in our project is [4]. In this paper, a diffusion model is proposed as a super-resolution model to downscale low-resolution wind-speed images to high-resolution images of a region encompassing Italy. In comparison with [4], the results we achieve over the region of Belgium are inferior to those they obtain for the region of Italy. The probable causes of this discrepancy are explored later in the report.

3 Dataset

Regarding the dataset, as in [6] and [4], we use the ERA5 global reanalysis dataset, presented in [2], as the low-resolution (LR) input. This dataset is in a grid format and covers the entire planet, with temporal coverage from 1940 to the present and an hourly temporal resolution. Moreover, it gets updated every day. For the high-resolution (HR) data, which is the output of our model, we use the Copernicus European Regional Reanalysis (CERRA) dataset. The CERRA dataset provides spatially and temporally consistent historical reconstructions of meteorological variables in the atmosphere and at the surface. Its temporal coverage is from 1984 to the present with a temporal resolution of 3 hours. Both datasets notably contain wind speed measurements, which will be our variable of interest. In ERA5, these measurements have a 0.25° x 0.25° resolution (approximately 27.75 km x 27.75 km) while in CERRA they have a 5.5 km x 5.5 km resolution.

Filtering
Both datasets have been filtered to retain a spatial area centered on Belgium. Only data points within a geographical rectangle defined by the coordinates (-2, 52) and (7, 49) were retained. The first coordinate represents longitude and the second represents latitude.

Normalization
Both datasets have been normalized using min-max normalization, defined as follows:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{1}$$

Here, X represents the original dataset, X' is the dataset normalized between 0 and 1, max is a function that retrieves the maximum value of the dataset, and min is a function that retrieves the minimum value of the dataset. The data can thus be denormalized using the following equation:

$$x = x'(\max(X) - \min(X)) + \min(X) \tag{2}$$

In this equation, x is the denormalized data, x' is the normalized data, and X is the original dataset. To denormalize the images generated by our model, we retain the maximum and minimum values of both the LR and HR datasets used for training.

Splitting
To train and evaluate our model, we divide the datasets into three subsets: training, validation, and testing. The data is split in a temporally stratified manner to preserve the temporal characteristics and dependencies within the datasets. Specifically, the training set includes data from the earliest years up to 2018, followed by the validation set in 2019, and the testing set in 2020. This division ensures that our model is trained on a substantial portion of the data, while being validated and tested on separate subsets to evaluate its performance and generalization capabilities. Moreover, this chronological division helps simulate a real-world scenario where the model is trained on past data and evaluated on future, unseen data.

4 Method

In this section, the proposed downscaling method is described through multiple subsections. Firstly, we formalize the problem, delve into the mathematics behind our models, and discuss the metrics used to evaluate them. Finally, the training procedure and the selection of hyperparameters is explained.

4.1 Problem Statement

The problem that our proposed method tackles is weather variable (WV) downscaling. The objective of this problem is to predict the regional high resolution data (HRD) of a target WV at time t based on the low resolution data (LRD) of multiple WVs at time t (often coming from global climate models). This problem can be formalized as follows:

$$y = m(x^1, \ldots,\ x^n,\ \theta) \tag{3}$$

where y is the HRD of the target WV, x^i is the LRW of the i^{th} weather input variable, m is the downscaling model, and θ is the parameter of the model.

Additionally, since the LRD of the input WV preceding the HRD that we want to downscale is unavailable, the problem can be extended as follows:

$$y_t = m(x_t,\ x^1_{t-1}, \dots,\ x^1_{t-n},\ x^2_t,\ x^2_{t-1},\ \dots,\ x^n_{t-n},\ \theta) \tag{4}$$

where y_t is the HRD of the target WV at time step t, x^i_t is the LRW of the i^{th} weather input variable at time step t.

Data representation
Both the LRD and the HRD are represented as grids of real values. This grid can be formalized as three matrices, where each element of one matrix is the value of the concerned weather variable in an area. Each element of the other two matrices represents respectively the longitude and the latitude of the area. Additionally, the latitude (and longitude) of the area increases with the corresponding row (and column). The size of the matrices for the low-resolution data (LRD) is 13×21, while the size for the high-resolution data (HRD) is 64×64.

4.2 Model

In this section, the deep learning model used is described. Firstly the general diffusion model and the conditional diffusion model are defined. Then the general architecture used is presented. Finally, some of the characteristics of our architecture are explained in more depth.

Diffusion model
In machine learning, diffusion models are one of the most popular generative models for synthetic images. The goal of this diffusion model is to learn a diffusion process. A diffusion process is defined by two key subprocesses, namely the forward process and the reverse process.

The forward process consists in starting with an input x and noising it by adding Gaussian noise iteratively to converge to a normal distribution with a zero mean and a variance I. This forward process is defined as follows:

$$x_s = \sqrt{1-\beta_s}x_{s-1} + \sqrt{\beta_s}z_s \quad \text{and} \quad \lim_{s\to\infty} x_s \to \mathcal{N}(0, I) \tag{5}$$

where x_s is the noisy input at noise step s, z_s are IID samples from $\mathcal{N}(0, I)$ and β_s is the noise ratio constant at noise step s.

The reverse process consists in starting with a pure noise sample from $\mathcal{N}(0, I)$ and iteratively denoising it to generate an image. To denoise the image, the diffusion model has to predict the noise that was added at each time step. This process is formalized by:

$$p(x_{0:S}) = p(x_s)\prod_{s=1}^{S} p_\theta(x_{s-1}|x_s) \tag{6}$$

$$p(x_s) = \mathcal{N}(x_s; 0, I) \tag{7}$$

$$p_\theta(x_{s-1}|x_s) = \mathcal{N}(x_{s-1}; \mu_\theta(x_s, s), \sigma^2_\theta(x_s, s)I) \tag{8}$$

$$x_{s-1} = \mu_\theta(x_s, s) + \sigma_\theta(x_s, s)z \tag{9}$$

with $z \sim \mathcal{N}(0, I)$

Conditional diffusion model
Conditional diffusion models have the same forward process but during the reverse process some additional information, called conditional information, is provided at each time step. Hence, the reverse process can be formalized as follows:

$$p(x_{0:S}|y) = p(x_S)\prod_{t=1}^{S} p_\theta(x_{s-1}|x_s, y) \tag{10}$$

where y is the conditional information.

The model used in our project is a conditional diffusion model where the input is the HDR of the target WV and the conditional information is the LRD of the weather input variables.

Architecture
The architecture used in our methods is a U-Net framework adapted for diffusion models. [1] As shown in Figure 2, the U-Net gets its name from its distinctive U-shaped design. Each input channel is a low-resolution image at time step t of one of the selected weather variables (WVs). The low-resolution data is upscaled to dimensions 64×64 before being concatenated to the high-resolution images, which are also 64×64. On the other hand, the output is the predicted noise applied to each pixel at the noise step s.

The left side of the architecture constitutes the contracting path, which captures information through convolutional and pooling layers. The right side forms the expansive path, which consists of upsampling and convolutional operations.

The contracting path gradually reduces the spatial resolution of the input image while increasing the number of feature channels, thereby facilitating the extraction of high-level semantic features. Conversely, the expansive path restores the spatial resolution via upsampling layers while preserving the learned features, although it reduces the number of channels. Crucially, it incorporates skip connections from the contracting path to facilitate training and improve accuracy.

[1]Our architecture was inspired by a video that can be found at `https://www.youtube.com/watch?v=a4Yfz2FxXiY&t`

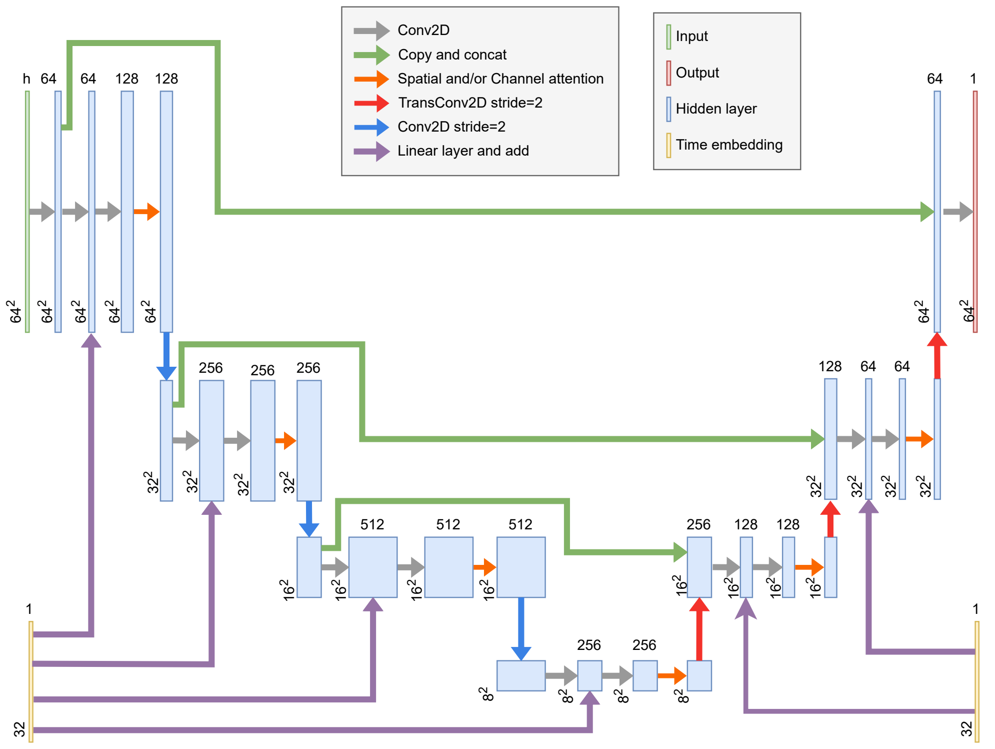


Figure 2: Diagram of the U-Net architecture of the proposed model

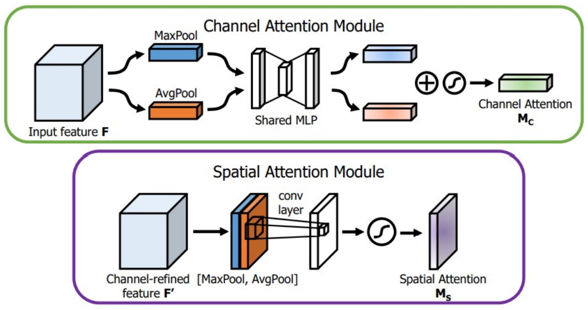


Figure 3: Functional diagram from [9] of the channel attention and spatial attention module. As illustrated, the channel sub-module utilizes both max-pooling outputs and average-pooling outputs with a shared network; the spatial sub-module utilizes similar two outputs that are pooled along the channel axis and forward them to a convolution layer.

Channel and Spatial Attention

In our U-Net architecture, we have integrated channel attention and spatial attention modules at each block of our network. The principles of these modules are presented in Figure 3. The channel attention module is defined by the following equation:

$$x' = M_c(x) \otimes x \tag{11}$$

Here, $x' \in \mathbb{R}^{C\times H\times W}$ represents the output, $x \in \mathbb{R}^{C\times H\times W}$ denotes the input, and $M_c \in \mathbb{R}^{C\times 1\times 1}$ corresponds to the 1D channel attention map. Furthermore, M_c is computed as follows:

$$M_c(F) = \sigma(MLP(Avg(x)) + MLP(Max(x))) \tag{12}$$

where σ denotes the sigmoid function, MLP represents a multi-layer perceptron, Avg signifies an average pooling operation, and Max indicates a max pooling operation.

The spatial attention module is defined as follows:

$$x' = M_s(x) \otimes x \tag{13}$$

Where $x' \in \mathbb{R}^{C\times H\times W}$ denotes the output, $x \in \mathbb{R}^{C\times H\times W}$ represents the input, and $M_s \in \mathbb{R}^{1\times H\times W}$ signifies the 2D spatial attention map. Moreover, M_s is computed as follows:

$$M_s(x) = \sigma(f^{7\times 7}([Avg(x); Max(x)])) \tag{14}$$

where σ represents the sigmoid function, $f^{7\times 7}$ denotes a convolution operation with a filter size of 7 by 7, Avg denotes an average pooling operation, and Max denotes a max pooling operation.

Time Embedding

As shown in Figure 2, a vector representation of the time step in the denoising process is provided at each block. This representation is computed in two stages. First, the integer value of the time step is converted into a vector of fixed size through a process called sinusoidal positional embedding.

This process can be defined by the following equations:

$$t'[2k] = \sin\left(\frac{t}{1000^{2k/d_{\text{model}}}}\right) \tag{15}$$

$$t'[2k+1] = \cos\left(\frac{t}{1000^{2k/d_{\text{model}}}}\right) \tag{16}$$

where $t'[2k]$ denotes the value of the embedding at the $2k$-th position, t is the integer value of the time step, and d_{model} represents the dimension of the embedding, which is 32 in this case.

This time-embedded vector is then fed into a dedicated linear layer at each block of the U-Net. Each block has its own dedicated linear layer that recomputes a new time representation in the form of a vector. This vector, sized to match the number of channels, is then added to the corresponding block's feature map.

4.3 Evaluation

In this section, we define the metrics used to assess the performance of our model and provide the rationale behind their selection.

The metrics used to evaluate the inference performance of our method are the Mean Squared Error (MSE) and the Structural Similarity Index Measure (SSIM). MSE has been selected because it is a classic metric for evaluating the error between a target and a prediction, providing a good intuition about the quality of the prediction. However, as highlighted in [8], two generated images with the same MSE can have significantly different perceived quality by human evaluators. Thus, to complement MSE and provide a stronger assessment of the quality of the generated images, we also use SSIM.

By using both MSE and SSIM, we can be confident that a decrease in MSE and increase in SSIM indicate an improvement in the model's performance.

MSE is computed using the `PyTorch` library, and SSIM is computed using the `Torchmetric` library with default parameters.

4.4 Training procedure

The goal of the training procedure is to refine a diffusion model capable of predicting and denoising high-resolution wind-speed images from their low-resolution counterparts. This process involves learning the optimal parameters of the described U-Net based model using a dataset of paired low and high-resolution images. The low resolution image are conditional information and high resolution image are the target output.

Noise scheduler configuration
To begin with, the noise scheduler manages the progressive addition of noise in diffusion models. It is modeled by the `NoiseScheduler` class in our implementation. This class defines a linear noise pattern, pre-calculates the terms required for the diffusion steps, and provides a method for adding noise to an image according to a specified noise step.

The noise scheduler is initialized with a total number of noise steps (S), and a range for the β_s values. This scheduler dynamically adjusts the noise level added to the images throughout the training process based on a linear progression from minimum to maximum noise. The scheduler calculates a beta schedule linearly interpolating between the minimum and maximum noise levels. This schedule dictates how much noise is added at each noise step of the diffusion process, enabling controlled deterioration of the image quality for subsequent reconstruction training.

Training set-up
The model is specifically configured to operate on CUDA-enabled devices, utilizing GPU acceleration to significantly enhance processing efficiency. For optimization, the Adam optimizer is employed, known for its proficiency with sparse gradients and its adaptability to various deep learning tasks, especially those involving images.

The loss function we minimize is the Mean Squared Error (MSE). It measures the effectiveness of the U-Net in predicted the noise added to the original high resolution images, conditionally on their low resolution counterparts.

Batch processing & data handling
Each training batch involves pairs of low and high-resolution images. For each high-resolution image in the batch, the `add_noise` function of the `NoiseScheduler` is called with a randomly chosen noise step. This function computes the noised image by combining the original image with Gaussian noise scaled according to the computed betas from the noise schedule, see Eq 5. As discussed in the next section, we found the optimal size to train our model to be 64.

Training dynamics
During training, for each batch, the model attempts to predict the noise that has been added to the high-resolution images. It uses the current noised image, the actual LRD images and possibly the LRD images from previous timesteps as input. The MSE loss between this predicted noise and the actual noise is calculated. Backpropagation is used to update the weights of the model, minimizing the MSE loss, with the goal of improving the model's ability to accurately predict and counteract applied noise. To avoid gradient explosion, a gradient clipping technique is applied, clipping gradients to a maximum norm of 1.0.

At the end of each epoch, the model's performance is validated using a separate validation dataset. This helps monitor overfitting and the model's generalization.

The pseudocode for the training algorithm is given below [3].

Algorithm 1 Training

```
1: repeat
2:     $x_0 \sim q(x_0)$
3:     $s \sim \text{Uniform}(1, \dots, S)$
4:     $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
5:     $x_s = \sqrt{\alpha_s} x_0 + \sqrt{1 - \alpha_s}\epsilon$
6:     Backpropagate on $\|\epsilon - \epsilon_\theta(x_s, \alpha_s)\|^2$
7: until converged
```

Inference and Evaluation

During and after training, the model undergoes inference tests using denoising diffusion probabilistic model (DDPM) inference methods. This evaluation assesses the model's effectiveness in generating accurate, high-quality subsampled images across epochs. As previously explained, the metrics used for evaluation are the MSE and SSIM calculated for objective evaluation.

The pseudocode for the sampling algorithm is given below.

Algorithm 2 Sampling (DDPM)

```
1: $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$
2: for $s = S, \dots, 1$ do
3:     if $s > 1$ then
4:         $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$
5:     else
6:         $\mathbf{z} = 0$
7:     end if
8:     $\mathbf{x}_{s-1} = \frac{1}{\sqrt{\alpha_s}}\left(\mathbf{x}_s - \frac{1-\alpha_s}{\sqrt{1-\bar{\alpha}_s}}\epsilon_\theta(\mathbf{x}_s, s)\right) + \sigma_s \mathbf{z}$
9: end for
10: return $\mathbf{x}_0$
```

where $\alpha_s = 1 - \beta_s$

4.5 Hyperparameter fine-tuning

In this section, we describe the choices we made to select the final values of our model's parameters. We identified 8 hyperparameters to adjust to improve our model's performance. Their impact and their optimal values are discussed. [2]

Learning rate

The learning rate is a commonly used value for diffusion networks: $lr = 0.0001$. Other values were tested but did not give significantly better results. Consequently, we decided to keep this value. Table 1 and Table 2 show the various validation and training losses after many epochs for each learning rate tested.

	Validation Loss	**Training Loss**
$lr = 0.0001$	0.1082	0.0075
$lr = 0.00025$	0.1146	0.0079
$lr = 5e - 5$	0.1132	0.0077

Table 1: Training and validation loss (MSE) at the end of 120 epochs for $S = 50$ and different learning rate values.

	Validation Loss	**Training Loss**
$lr = 0.0001$	0.0053	0.00415
$lr = 0.0005$	0.07	0.0041

Table 2: Training and validation loss (MSE) at the end of 100 epochs for $S = 200$ and different learning rate values.

Attention mechanism

Spatial and channel attention mechanisms did not improve the model's performance. This suggests that, for the current tasks and dataset, these attention mechanisms do not bring any additional benefits, although they slightly speed up the training process, as evidenced by Table 3). This inefficacy might stem from the small size of our images (64 x64).

	Attention	**No Attention**
Average training time	74.5	78.2

Table 3: Average training time per epoch in seconds with and without attention mechanism

Dropout

Implementing a dropout rate of 0.2 did not improve our model's performance. In fact, it led to worse results as shown in Figure 4. This could indicate that the model does not overfit the training data, or that drop-out is not an appropriate regularization technique in this context.

[2] It is essential to note that all the results presented in this section are obtained from evaluation on normalized images. We tuned our parameters without denormalizing the predictions before evaluating performances. We later realized that this was a mistake. Unfortunately, by the time we made this realization, we did not have enough time to restart the training process. This explains the discrepancy in evaluation measures between this section and the results sections, as the evaluation measures in the results section are based on unnormalized images.

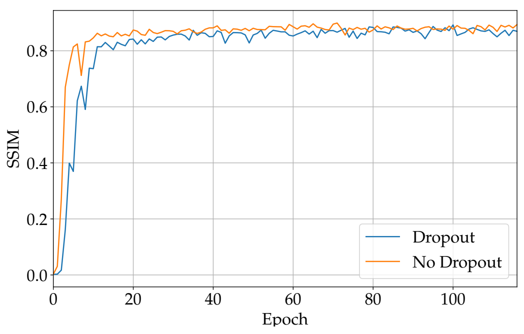


Figure 4: SSIM predictions at each epoch on the validation set with and without dropout

Ensemble method

To boost our model's predictions' accuracy, we implemented a method that produces several predictions and returns their average. This technique was inspired from [4]. As shown in Figure 5 and Table 4, averaging more than 2 predictions improves our results. Beyond this threshold, the marginal impact is no longer noticeable.

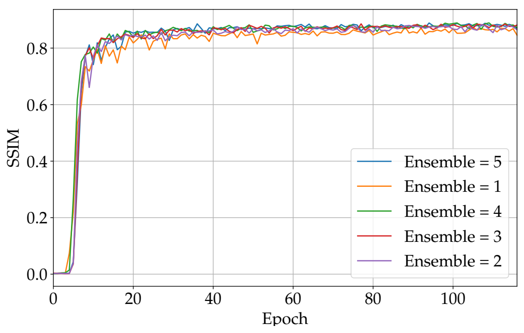


Figure 5: SSIM predictions at each epoch on the validation set for different prediction numbers used to average output. We used 50 scheduler noise steps and only current images as input.

	SSIM
E = 1	0.8818
E = 2	0.8892
E = 3	0.8978
E = 4	0.9045
E = 5	0.8985

Table 4: Final SSIM prediction on the validation set for different prediction numbers used to average output. We used 50 scheduler noise steps and only current images as input.

Number of previous images considered

To improve the model's ability to predict and denoise images, we enabled it to base its predictions not only on the low-resolution image from the time step being predicted but also on those from previous time steps. The number of time steps taken into account is managed by the parameter `pred_step`. By integrating information across these time steps, the model improves its predictive performance. Figure 6 and Table 5 illustrates the impact of conditioning on varying numbers of previous images on the model's outcomes. Optimal results are achieved when the model conditions its predictions on the current time step plus three preceding time steps.

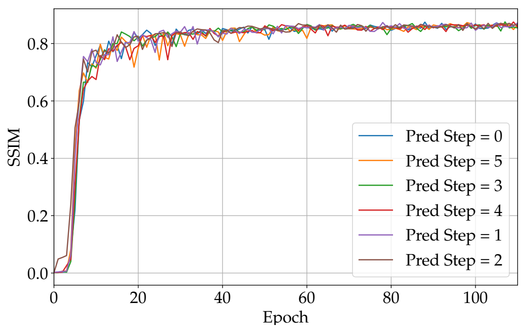


Figure 6: SSIM predictions at each epoch on the validation set for different number of previous images used as input and using 50 scheduler noise steps.

	SSIM
Pred Step = 0	0.8774
Pred Step = 1	0.8846
Pred Step = 2	0.8849
Pred Step = 3	0.8902
Pred Step = 4	0.8795
Pred Step = 5	0.8862

Table 5: Final SSIM prediction on the validation set for different number of previous images used as input and using 50 scheduler noise steps.

Scheduler noise steps

In regards to the scheduler noise steps, we tested different maximum number of steps S and considered two scenarios. In the first scenario, we only the current low-resolution image was in the input. In the second scenario, we also included the three previous images in addition to the current image. The results obtained for the various values of S differed between the two cases, as demonstrated by Figure 7 and Figure 8.

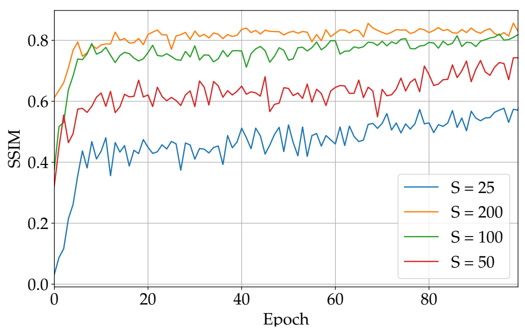


Figure 7: SSIM predictions at each epoch on the validation set for different time step values and considering only current input images.

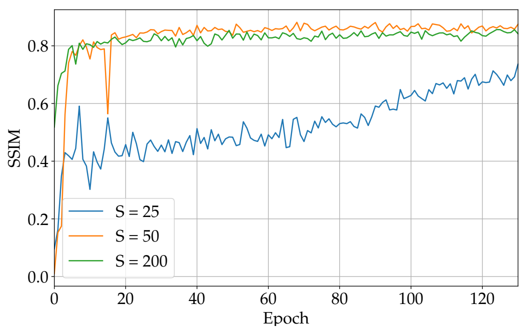


Figure 8: SSIM predictions at each epoch on the validation set for different time step values and also considering images of the 3 previous times as input.

$S = 200$ is identified as the best parameter value in the first scenario, suggesting that a higher number of (de)noising steps is beneficial under these conditions. However, $S = 50$ gives better results in the second scenario, where several previous images are added to the input.

In the end, $S = 50$ with 3 previous time steps is the setting that yields the best results. Hence, we can conclude that fewer time steps are needed if we take the previous images as input. In addition, a lower S implies fewer steps during inference which in turn implies a reduced computing time.

Batch size
Batch sizes of 16, 32, 64 and 128 were tested. No major differences were observed for any of these models in terms of results. Still, Table 6 reveals that a batch size of 64 accelerates our model's training.

	Average training time
Batch Size = 16	62.5
Batch Size = 32	59.8
Batch Size = 64	33.9
Batch Size = 128	56.8

Table 6: Average training time per epoch in seconds for different batch size

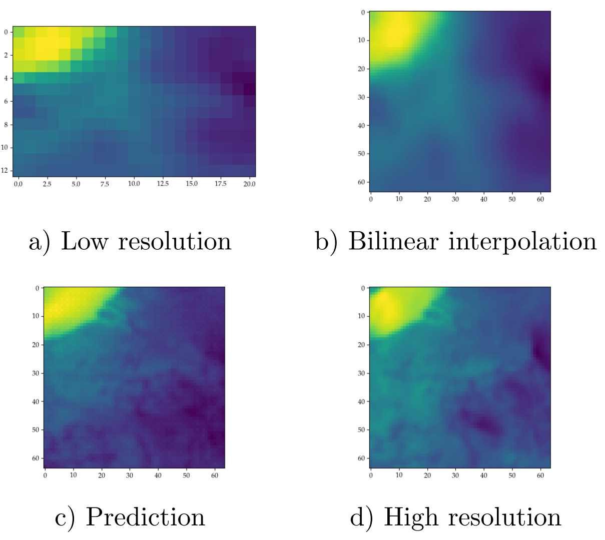
Noise scheduler weights
Other parameters that could have been tuned are β_0 and β_S, both from the noise scheduler. However, due to a lack of time, the tuning of these parameters was omitted. Instead, we used the values suggested in [4]. In other words, we used $\beta_0 = 0.05$ and $\beta_S = 0.95$.

Final parameters
The analysis of our results allowed us to identify optimal parameter values for our models. As these are the values used to evaluate our model's performance in the results section, they are summarized below.

- **Learning rate:** A learning rate of 0.0001 to ensure steady convergence without overshooting the minimal loss.
- **Attention mechanism:** Utilization of a simple model structure with no attention mechanisms.
- **Dropout:** No dropout is applied, maintaining all neurons active during training to maximize learning.
- **Ensemble method:** Averaging the outcomes of 3 different predictions to enhance the stability and accuracy of the output.
- **Previous images considered:** Three previous images alongside the current input image to enrich the model's contextual understanding.
- **Noise step S :** A fixed time step of 50 the best trade-off between inference time and accuracy.
- **Batch size:** A batch size of 64, for quicker training.
- **Noise scheduler weights:** Employing a noise scheduler with a $\beta_0 = 0.05$ and $\beta_S = 0.95$

5 Results

As a final step, we assess the performance of our optimal model on the test set. The resulting MSE and SSIM values are displayed in Table 7. It is apparent that our results do not match those reported for Italy in [4]. Specifically, our model's MSE is 0.709, whereas their best model achieved an MSE of 1.02e-03. Similarly, our model's SSIM is 0.538, compared to their best model's SSIM of 0.845.


a) Low resolution b) Bilinear interpolation

c) Prediction d) High resolution

Figure 9: Comparison of bilinear interpolation and our model's prediction for an example from the test set. The images clearly show that our model's prediction is better than what is obtained from interpolation.

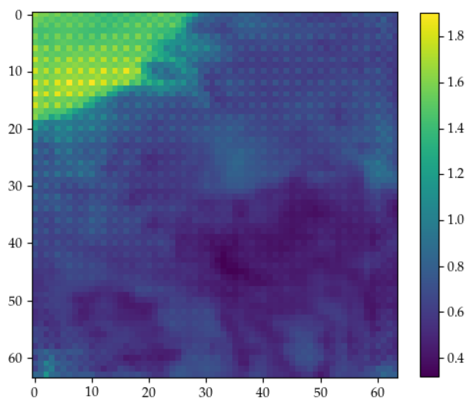
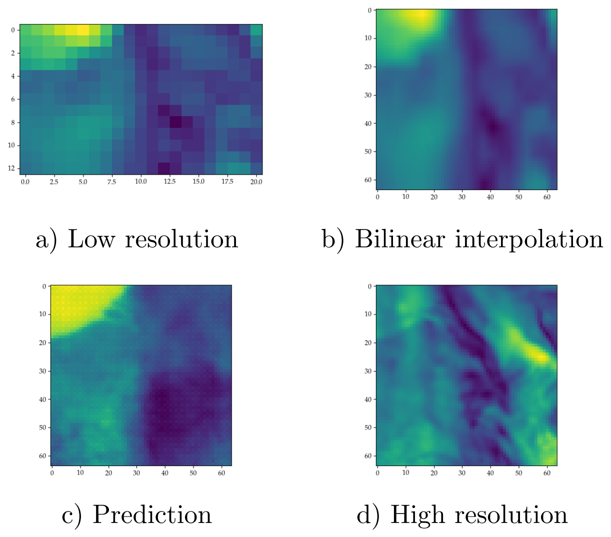


Figure 10: Geographic distribution of our best model's errors. The image shows that our model's predictions are not as good for the top-left corner of the image than they are for the rest of it. This is certainly due to the fact that this area of the image corresponds to the coastal area of Belgium.


a) Low resolution b) Bilinear interpolation

c) Prediction d) High resolution

Figure 11: Comparison of bilinear interpolation and our model's prediction for a poorly predicted example from the test set. The high resolution image seems not to follow the typical pattern with high wind levels close to the sea.

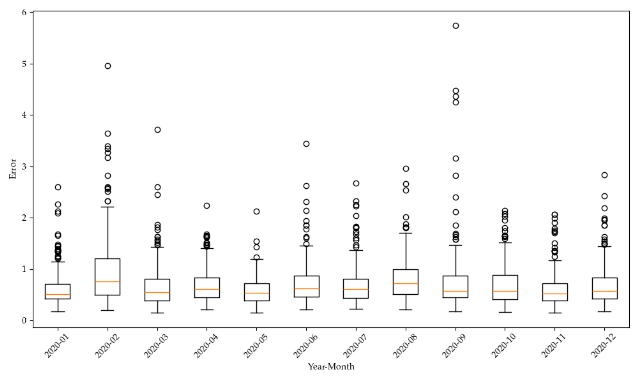


Figure 12: Monthly distribution of our best model's errors. The plot does not show any significant difference between the months. The only exception might be February and September, where our model exhibits high error values.

However, despite both datasets being derived from subsets of ERA5 and CERRA, the authors of [4] focused on predicting wind speeds in Italy, while our study is centered on Belgium. This makes direct comparisons challenging. Also, they cover a much larger region than we do, which implies their model was trained on a more comprehensive dataset. Furthermore, their images are significantly larger than ours, with high and low-resolution image sizes of 64x64 and 256x256 pixels, respectively, compared to our sizes of 13x21 and 64x64 pixels. Consequently, their low-resolution images contain substantially more information and structure, enhancing their model's predictions.

	MSE	**SSIM**
Our Model	0.709	0.538
Bilinear Interpolation	11.507	0.006

Table 7: MSE and SSIM on the test set for our final model. The table also shows the results obtained by applying bilinear interpolation to the low resolution images. Our model represents a major improvement with respect to interpolation.

Table 7 also presents the results obtained by applying bilinear interpolation to the low-resolution images, serving as a baseline for comparison. The table reveals that even though our results do not reach the level achieved in [4], they still represent a significant improvement compared to basic techniques such as bilinear interpolation. Figure 9 illustrates a comparison between our model's predictions, the true high-resolution image, and the results obtained through bilinear interpolation for an example from the test set. This further underscores that our model achieves significantly better outcomes than basic upscaling techniques such as interpolation.

Still, the example in Figure 9 corresponds to a case with typical wind speeds, characterized by generally low values across the country, except near the sea where they are significantly higher. However, wind patterns in Belgium do not always follow this distribution, occasionally causing our model to struggle with accurate predictions. This is illustrated in Figure 11, which depicts a scenario where the actual high-resolution image differs significantly from most images in the training set. The plot indicates that our model has difficulty to predict the structure of this image accurately. This demonstrates that while our model performs satisfactorily overall, there is room for improvement.

To further evaluate the performance of our diffusion model, we analyzed the geographic distribution of its prediction errors. Specifically, Figure 10 presents the model's mean squared error (MSE) per pixel across the test set. The image reveals that our model has significant difficulty accurately predicting the top-left region, which corresponds to the coastal area of Belgium. This is not surprising as wind speeds are typically higher and more variable near and over the sea. Consequently, the increased wind speed and its variability in this area pose greater challenges for accurate prediction, leading to higher prediction errors.

5.1 Further improvements

As several limitations of our approach were highlighted throughout this report, we make some suggestions for further improvements. First, enhancing the dataset used for our model would certainly be beneficial. The current low-resolution images (13 by 21 pixels) do not provide sufficient information to predict the high resolution images very accurately. We also suggest using larger high-resolution images, potentially including regions beyond Belgium, to provide more comprehensive data.

Additionally, our dataset could be improved by incorporating variables in addition to current and previous wind speeds. Conditioning on other weather variables, such as precipitation, could enhance the model's predictive performance. Increasing the number of samples would also provide the model with a broader dataset, which could further improve its accuracy.

Another critical improvement would be to implement DDIM inference. DDIM has been shown to significantly accelerate the inference process while maintaining high-quality results [7].

Despite our efforts to optimize the model parameters, time constraints limited our ability to fully explore the hyperparameter space. Hence, allocating more time to this process could yield better results.

Lastly, we found that adding blocks to our current U-Net architecture did not significantly improve its performance. Nonetheless, if larger images were used, increasing the U-Net depth would likely lead to more accurate predictions.

6 Conclusion

In conclusion, our project successfully demonstrated the use of diffusion models for downscaling wind speed predictions for Belgium. By increasing the resolution of forecasts from global climate models, our approach provides high-resolution outputs suitable for regional applications.

The model we constructed significantly outperformed bilinear interpolation, a basic image upscaling method. It achieved a mean squared error (MSE) of 0.709 and a structural similarity index (SSIM) of 0.538 on the test set. However, higher prediction errors were observed in Belgium's coastal area due to increased wind variability.

While our results are promising, there is ample room for improvement. This is evidenced by the quality discrepancy between our results and those presented in [4]. Future work should focus on increasing dataset resolution and coverage or incorporating additional weather variables. Additionally, employing advanced inference techniques such as DDIM would significantly speed up the sampling process. Fine-tuning hyperparameters and expanding the U-Net architecture could also improve accuracy.

Overall, this project exhibits promising results for the application of diffusion models in high-resolution weather prediction. Continued refinement of the methods presented is expected to enhance the accuracy and reliability of regional forecasts.

7 Code

The repository for our code is available at the following link: `https://github.com/Smorta/Deep`.

References

[1] Carlos Alberto Gomez Gonzalez. *DL4DS – Deep Learning for empirical DownScaling*. 2022. arXiv: `2205.08967 [cs.LG]`.

[2] Hans Hersbach et al. "ERA5 hourly data on single levels from 1940 to present". In: *Copernicus Climate Change Service (C3S) Climate Data Store (CDS)* (2023). DOI: `10.24381/cds.adbb2d47`.

[3] Ajay Jain Jonathan Ho and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *arXiv preprint arXiv:2006.11232* (2020).

[4] Fabio Merizzi, Andrea Asperti, and Stefano Colamonaco. *Wind speed super-resolution and validation: from ERA5 to CERRA via diffusion models*. Feb. 2024.

[5] Brian B. Moser et al. *Diffusion Models, Image Super-Resolution And Everything: A Survey*. 2024. arXiv: `2401.00736 [cs.CV]`.

[6] Neelesh Rampal et al. "High-resolution downscaling with interpretable deep learning: Rainfall extremes over New Zealand". In: *Weather and Climate Extremes* 38 (2022), p. 100525. ISSN: 2212-0947. DOI: `https://doi.org/10.1016/j.wace.2022.100525`. URL: `https://www.sciencedirect.com/science/article/pii/S2212094722001049`.

[7] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models". In: *arXiv preprint arXiv:2010.02502* (2020).

[8] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: `10.1109/TIP.2003.819861`.

[9] Sanghyun Woo et al. *CBAM: Convolutional Block Attention Module*. 2018. arXiv: `1807.06521 [cs.CV]`.