Efficient Data-Driven Gap Filling of Satellite Image Time Series Using Deep Neural Networks with Partial Convolutions

MARIUS APPEL

a University of Münster, Institute for Geoinformatics, Münster, Germany

ABSTRACT: The abundance of gaps in satellite image time series often complicates the application of deep learning models such as convolutional neural networks for spatiotemporal modeling. Based on previous work in computer vision on image inpainting, this paper shows how three-dimensional spatiotemporal partial convolutions can be added as layers to other types of neural networks, making it relatively easy to integrate with existing deep learning models. However, the approach does not provide prediction uncertainties and further research is needed to understand and improve model transferability. The implementation of spatiotemporal partial convolutions and the U-Net-like model is available as open-source software.

SIGNIFICANCE STATEMENT: Gaps in satellite-based measurements of atmospheric variables can make the application of complex analysis methods such as deep learning approaches difficult. The purpose of this study is to present and evaluate a purely data-driven method to fill incomplete satellite image time series. The application on atmospheric carbon monoxide data suggests that the method can achieve prediction errors comparable to other approaches with much lower computation times. Results highlight that the method is promising for larger datasets but also that care must be taken to avoid extrapolation. Future studies may integrate the approach into more complex deep learning models for understanding spatiotemporal dynamics from incomplete data.

KEYWORDS: Remote sensing; Satellite observations; Data science; Deep learning; Neural networks

1. Introduction

Deep learning (DL) and particularly convolutional neural networks (CNNs) have been exceptionally successful for satellite image analysis tasks including object detection and segmentation. In recent years, DL models have been increasingly used also for modeling continuous spatiotemporal phenomena such as soil moisture (ElSaadani et al. 2021) or air temperature (Amato et al. 2020). Developments in De Bézenac et al. (2019) demonstrate that such models can reproduce fundamental physical properties of processes such as advection and diffusion in a purely data-driven way. Similarly, deep learning models have been successful in hydrologic modeling, e.g., to infer hydraulic conductivity (Moghaddam et al. 2021), surface water–groundwater exchanges (Moghaddam et al. 2022), and to forecast precipitation using convolutional and recurrent neural networks (Ehsani et al. 2022). Rasp and Thuerey (2021) even show that CNN-based models can perform medium-range weather prediction with performance close to an operational physical model. More generally, Camps-Valls et al. (2021) and Reichstein et al. (2019) describe challenges and approaches for hybrid data-driven and physical modeling. However, a fundamental challenge when applying CNN-based spatiotemporal models on satellite image time series is the existence of missing values e.g., due to atmospheric conditions (in many cases clouds).

Numerous approaches to fill gaps in spatiotemporal data have been proposed, including statistical methods based on discrete cosine transforms (Wang et al. 2012), singular spectrum analysis (Ghafarian Malamiri et al. 2018; von Buttlar et al. 2014), Markov random fields (Fischer et al. 2020), spatiotemporal interpolation (Cressie and Johannesson 2008; Appel and Pebesma 2020), and more algorithmic methods such as quantile regression in local neighborhoods (Gerber et al. 2018). In computer vision, a similar problem is referred to as image inpainting or video inpainting, where the aim is to restore corrupt parts of images and videos, respectively. Liu et al. (2018) present a promising approach for the former, using a U-Net-like (Ronneberger et al. 2015) encoder/decoder model with partial convolutional layers to fill gaps.

Since it has been shown that CNN models are capable of modeling complex tasks and at the same time predictions can be computationally efficient, this study aims at (i) making CNNs with partial convolutions applicable to spatiotemporal Earth observation data from satellite image time series,
(ii) evaluating prediction performance and computational aspects with regard to a quasi-global atmospheric dataset, and (iii) discussing limitations, advantages, and future work toward purely data-driven modeling of spatiotemporal dynamics from incomplete datasets.

Notice that recently Xing et al. (2022) similarly applied partial convolutions to a spatiotemporal snow cover dataset. In contrast to their work, the presented approach uses three-dimensional partial convolutions, compares predictions with other methods, uses atmospheric data on a (quasi) global scale, and discusses the approach as a computationally efficient gap filling method.

The remainder of this paper is organized as follows. Section 2 introduces the partial convolution operation and how it can be included in a model. Section 3 describes experimental details, applied models, and the dataset used, before results are presented in section 4. A discussion of limitations of the approach and potential for future research is given in section 5, and section 6 concludes the paper.

2. Methods

a. Partial convolutions

The following paragraph describes the partial convolution operation as introduced in Liu et al. (2018).

As compared with ordinary convolutions, partial convolutions not only receive a data subset of the same size as the filter kernel but also a corresponding binary mask as input. Let \( X, M, \) and \( K \) be the data input, the mask input, and kernel weights, respectively, all of identical shape. Vector \( M \) has 0s for missing values and 1s for valid observations, and we assume \( X \) is 0 for missing values, too. We can then write the partial convolution operation as an ordinary convolution of \( X \) and \( K \) followed by multiplication with the number of elements in \( K \) divided by the number of 1s in \( M \). The last step can be seen as applying a weight to adjust for missing values. Afterward, the mask value of the center pixel is set to 1 if there is at least one valid observation in \( X \). Formally, applying a partial convolution at one location \( x' \) can be written as (Liu et al. 2018)

\[
x' = \begin{cases} 
K \ast (X \odot M) \sum_{i} M^{i} / \sum_{i} M & \text{if } \sum_{i} M > 0 \\
0 & \text{otherwise}
\end{cases}
\]

where \( \odot \) is the element-wise multiplication, an asterisk indicates the ordinary convolution, and \( I \) is an array with 1s in the same shape as \( X, K, \) and \( M \). At the same location, the mask is updated by

\[
m' = \begin{cases} 
1 & \text{if } \sum_{i} M > 0 \\
0 & \text{else}
\end{cases}
\]

Similar to ordinary convolutional layers in neural networks, a bias can be added and, if the input has multiple channels, separate convolutions (with different weights) are applied before computing per-pixel sums of the convolved channels. Notice that partial convolutions implicitly provide a padding strategy by simply extending the mask and data subset at the boundaries with zeros.

b. Spatiotemporal models with partial convolutions

Similar to Liu et al. (2018) and Xing et al. (2022), we integrate partial convolutional layers in a U-Net-like (Ronneberger et al. 2015) model architecture consisting of encoder/decoder (or convolution/deconvolution) parts and skip connections. The encoder reduces resolution by strided partial convolutions and (typically) increases the channel depth while the decoder increases the resolution and combines lower resolution output with output from associated layers of the encoder.

In contrast to Liu et al. (2018) and Xing et al. (2022), our model applies three-dimensional partial convolutions. The input is a spatiotemporal block \( X \) of size \( n_t \times n_s \times n_x \) and a binary mask \( M_X \) of the same size, where 1 represents that the corresponding value in \( X \) is valid and 0 represents missing values.

At first, one or more partial convolutional blocks are applied to the input. A block applies one or more partial convolutions sequentially with a user-defined number of kernels, where the last convolution applies striding. Each partial convolutional layer is followed by a leaky ReLU (\( \alpha = 0.1 \)) activation function. Once the lowest spatiotemporal resolution is reached, the output is upsampled again to increase spatiotemporal resolution. The upsampled output is then concatenated with the output of the associated block from the convolutional phase, before another partial convolutional block (without striding) and a ReLU activation function are applied until the original size of the input blocks is reached.

Gaps are filled during the encoder part, while individual partial convolutions are applied. Using larger kernels and applying more and/or larger striding result in a faster filling of gaps. The depth of the model hence must be adapted to the size of the gaps to make sure that all gaps become filled.

Figure 1 illustrates the basic architecture of the model in an example, where spatiotemporal blocks have size \( 128 \times 128 \times 16 \), spatiotemporal resolution is reduced by a striding factor of \( 2 \) in all dimensions, there are three partial convolutional blocks in the encoder, each consisting of a single partial convolutional layer, and the channel depth is increased as the spatiotemporal resolution is decreased by using an increasing number of filters.

The proposed model is highly customizable. Table 1 in section 3 lists important hyperparameters with regard to the model architecture, training, and data preparation along with their values used in the experiments.

Our implementation (section 2e) allows us to define architectural parameters differently per dimensions. For example, it is possible to add purely temporal or spatial partial convolutional blocks, and to define different kernel sizes in space and time. This can be used to optimize the model for specific spatiotemporal phenomena, depending on spatial and temporal resolution and autocorrelations.

c. Addition of artificial gaps

To assess prediction errors during model training and validation, data must be predicted at locations with available
measurements. Let \( Y \) refer to a spatiotemporal block of the target data. Since \( Y \) may already contain missing values, let \( M_Y \) refer to the corresponding binary mask, where 1s represent available pixels and 0s represent missing values. Since the aim is to predict (larger) gaps, the following steps to create training samples with additional gaps are performed:

1) For all time slices in a spatiotemporal block \( Y \), simulate a two-dimensional Gaussian random field using a predefined covariance function \( \text{cov}(s, s') \) depending on the spatiotemporal distance between pairs of observations at locations \( s, s' \in [0, 1] \) within a block.

2) Create a binary mask \( M_{\text{sim}} \) by applying a threshold \( u \) on the simulated fields.

3) Mask corresponding pixels from \( Y \) to create \( X \), that is, calculate the combined mask \( M_X := M_Y \odot M_{\text{sim}} \) and set \( X := Y \odot M_X \), with \( \odot \) representing the element-wise product.

We use the exponential covariance function \( \text{cov}(s, s') = \sigma^2 \exp(-||s - s'||/d) \) with variance \( \sigma^2 = 0.95 \) and spatial range \( d = 0.4 \) for step 1 and a threshold \( \theta = 0.5 \) to select approximately 30% of the pixels for masking in \( M_{\text{sim}} \) on average in step 2. Figure 2 illustrates how the additional mask is applied on a single time slice of a spatiotemporal block.

The resulting \( X \) and \( M_X \) are inputs to our model, and \( Y \) is the target data used for training and validation. Since the amount of missing values in the original blocks varies strongly among the data, the amount of missing values in the synthetic mask is less important than the actual shape of added masks. To allow the model to learn long range predictions of a process, it is important to generate artificial gaps of different sizes instead of just leaving out random pixels, resulting in only very small gaps and overoptimistic validation scores.

d. Model training

Training the model to reconstruct \( Y \) from \( X \) and the corresponding mask \( M_X \) may consider different error metrics and different subsets of the data. As such, error metrics may consider all available pixels in \( Y \), only observations that have been masked additionally (artificially added gaps, available in \( Y \) but not in \( X \)), or only observations that are available in both, \( X \) and \( Y \). Prediction error metrics may use absolute or squared prediction errors to control the influence of outliers.
or extreme values. As shown in Liu et al. (2018) and Xing et al. (2022), it is possible to combine metrics, different data subsets, and include other losses such as the total variation to control the smoothness of transitions at gap boundaries.

Our implementation (section 2e) includes common prediction error metrics (mean absolute error, root mean squared error, and others) on either artificially added gaps only or on all pixels available in \( Y \), and additionally the total variation loss.

In the experiments (section 3c), we used the mean absolute error on artificially added gaps only as the loss function. Using RMSE and/or inclusion of pixels available in both \( Y \) and \( X \) did not lead to better results in terms of the validation scores. We also experimented with sums of several loss terms, including the total variation loss as suggested in Liu et al. (2018), Xing et al. (2022) with no improvement.

For optimizing the model weights, we used the Adam optimizer (Kingma and Ba 2014) and an adaptive learning rate schedule with exponential decay after an initial period with constant learning rate (see Table 1 for details).

e. Implementation

We implemented the presented approach in Python (Van Rossum and Drake 2009) using TensorFlow (Abadi et al. 2015) and Keras (Chollet et al. 2015). The implementation contains classes for three-dimensional partial convolutional layers and the U-Net-like model that can be customized by hyperparameters (see Table 1). Notice that we reuse the original implementation of ordinary convolutional layers and an available open-source implementation of two-dimensional partial convolutions in Keras (https://github.com/MathiasGruber/PConv-Keras; accessed 7 November 2022). The source code of our implementation, including a small example dataset and a pretrained model, is available on GitHub (https://github.com/appelmar/STpconv).

3. Application to Sentinel-5P data

To validate the proposed gap filling approach, we used imagery from the European Sentinel-5 Precursor (Sentinel-5P) mission for satellite-based monitoring of the atmosphere. The TROPOMI instrument onboard satellite measures atmospheric variables [total column observations of carbon monoxide, ozone, methane, nitrogen dioxide, and others (https://www.tropomi.eu/data-products/level-2-products; accessed 24 June 2022)] at high spatial resolution up to 3.5 km x 5.5 km and a revisit time of 1 day. Recently, Sentinel-5P NO2 observations have been integrated into operational Copernicus Atmosphere Monitoring Service forecasts by the European Centre for Medium-Range Weather Forecasts.

For the experiments, we downloaded 4518 images of total column carbon monoxide observations and corresponding per-pixel quality assessment images of the offline processing stream from the Sentinel-5P Level 2 open data catalog on Amazon Web Services (https://registry.opendata.aws/sentinel5p; accessed 7 November 2022). Images have been recorded between 1 January 2021 and 25 November 2021.

a. Data preprocessing

All 4518 images have been resampled to 0.1° spatial resolution, cropped to latitudes between −60° and 60°, and a pixel-wise filter to ignore pixels with quality values lower than or equal to 0.5, following the provider’s recommendation (Sentinel-5P Mission Performance Centre 2021), has been applied. Afterward, images have been aggregated daily, i.e., images from the same days covering different swaths have been combined, to yield daily composite images.

The resulting data cube has dimensions \((n_{\text{lat}}, n_{\text{lon}}, n_{\text{time}}) = (1200, 3600, 329)\) with coordinates \(\text{lat} \in [-60, 60], \text{lon} \in [-180, 180], \text{time} \in [1 \text{ January 2021}, 25 \text{ November 2021}],\) and a single variable (CO). All of the previous steps including the
resampling, aggregation, cropping, and filtering have been performed in the R software (R Core Team 2022) using the gdalcubes package (Appel and Pebesma 2019). Figure 3 shows the availability of valid observations of the resulting dataset. At most of the locations, less than 20% of the days are missing in the prepared dataset. Some exceptions include the northern coast of Australia, the Argentinian coast in the southern Atlantic Ocean, Southeast Asia, and farther coastal areas.

b. Training and validation sets

We divided the data cube into independent, nonoverlapping spatiotemporal blocks of size $(n_{\text{lat}}, n_{\text{lon}}, n_{\text{time}}) = (128, 128, 16)$. The block size was selected to take account of the fact that gaps are typically larger in space, and spatial autocorrelations between any two pixels are expected to show a longer range than temporal autocorrelations. This results in 5040 theoretically possible blocks, from which we randomly sampled two disjoint sets for model training ($n = 500$) and independent testing ($n = 250$).

For all considered spatiotemporal blocks, we created input data by adding artificial gaps as described in section 2c. For our final assessment of model performance, we apply an additional strategy to add gaps to the independent test set, where we completely leave out the last time slice from a block $Y$, but use all available data from previous time steps. As a result, our model assessment on the independent test set will not only calculate metrics for gap filling (as in section 2c), but also for one-step-ahead forecasting. In the following, we refer to this as validation strategies.

c. STpconv

For training our proposed U-Net-like STpconv model, 20% of the training samples were used for validation and the selection of hyperparameter values, where the number of convolutional blocks, striding, numbers of filters, as well as the loss functions and the learning rate have been varied. Considered loss functions during hyperparameter tuning include mean absolute error (MAE)/root mean squared error (RMSE) of additionally masked observations only, MAE/RMSE of all available observations, sums of the aforementioned functions, and the total variation loss added to the aforementioned functions. Using more complex and combined loss functions did not improve prediction errors on the 20% of the training blocks used for validation and, as a result, we simply used MAE on artificial gaps for training.

In our experiments, we trained two STpconv models with different hyperparameter values: STpconv$_L$ represents a moderately large model with 667,358 weights that uses standardized input blocks $x' = (x - \mu)/\sigma$, with global mean $\mu = 0.02679256$ mol m$^{-2}$ and standard deviation $\sigma = 0.02317798$ mol m$^{-2}$. In contrast, the smaller STpconv$_S$ model presents a good compromise of prediction performance and computation times, having only 21,678 weights and using raw input data without any transformation. Table 1 lists all hyperparameter values used for the models. The learning rate has been fixed for 30 (STpconv$_L$) and 50 (STpconv$_S$) epochs, respectively, before it was interactively reduced by an exponential decay $\text{LR}_{i+1} = \text{LR}_i \exp(-0.1)$ after every epoch.

d. Benchmark models

For comparison, predictions of the independent test dataset from the two selected STpconv models are compared with two naïve baseline models, two statistical models, and one neural network model using ordinary three-dimensional convolutional layers instead of partial convolution.

1) BLOCKWISE MEANS

First, we use the empirical mean from available observations of a spatiotemporal block $X$ to fill its missing values as a simple baseline.
2) Time series interpolation

As a second baseline, we independently apply linear interpolation on all time series of a block to fill gaps. We use the NumPy (Harris et al. 2020) implementation in numpy.interp with default parameters, meaning that if a time series starts or ends with missing values, the first/last available observation is carried forward/backward. As a result, the time series interpolation method is mostly repeating the last image in the case of one-step-ahead forecasting. If a time series has no valid observations, it is not predicted and omitted in the calculation of prediction scores. The result therefore might still contain gaps. To process all time series of a spatiotemporal block, numpy.apply_along_axis has been used.

3) Gapfill

The method described in Gerber et al. (2018), which we refer to as gapfill, constructs a spatiotemporal neighborhood for each missing value, whose size is increased until enough observations are available. Quantile regression is then applied on values in the neighborhood. This not only allows us to predict the median but also to quantify prediction uncertainties as confidence intervals. For details, the reader is referred to the original publication in Gerber et al. (2018). Notice that the method does not require any model training and hence was directly applied on the independent test dataset.

4) STMRA

We applied an efficient approximation of spatiotemporal Gaussian processes called multiresolution approximations (referred to as stmra) as proposed in Appel and Pebesma (2020). This approach recursively partitions the area of interest into smaller regions, assuming conditional independence between different regions at the same partitioning level, and uses a basis function representation of processes within regions to approximate residuals from previous partitioning levels. As compared with traditional Geostatistical modeling, this approach can be used for large datasets but still requires fitting a spatiotemporal covariance model. Let $\Delta_s$ and $\Delta_t$ be the spatial and temporal distances between any two locations, $\mathbf{I}$ represent a function that yields 1 if the argument is true and 0 otherwise, and $\mathbf{\theta}$ represent a parameter vector. We used the following separable spatiotemporal covariance function where both functions use the exponential covariance model:

$$
\text{cov}(\Delta_s, \Delta_t) = \mathbf{\theta}_1 \text{cov}_s(\Delta_s) \text{cov}_t(\Delta_t)
$$

$$
\text{cov}_s(\Delta_s) = [1 - \mathbf{\theta}_4 + \mathbf{\theta}_4 \mathbf{I}(\Delta_s = 0)] \exp(-\Delta_s / \mathbf{\theta}_2)
$$

$$
\text{cov}_t(\Delta_t) = [1 - \mathbf{\theta}_5 + \mathbf{\theta}_5 \mathbf{I}(\Delta_t = 0)] \exp(-\Delta_t / \mathbf{\theta}_3).
$$

The parameter vector $\mathbf{\theta}$ includes a joint overall variability, spatial and temporal ranges, and separate spatial and temporal nugget effects that include small-scale variability. For computational reasons, corresponding parameters have been fitted based on 10 randomly selected blocks of the training set.

5) Conv3D

To assess differences between partial convolutions and ordinary three-dimensional convolutions, we considered a model where all partial convolutional layers of our STpconv$_L$ model have been replaced by ordinary three-dimensional convolutional layers. We apply this model on the input data where missing values have been filled with zeros, representing the global mean after standardization. All other hyperparameter values were left unchanged, i.e., identical to the STpconv$_L$ model (see Table 1).

e. Model assessment

Prediction errors were compared in terms of mean absolute error (MAE), root-mean-square error (RMSE), Pearson’s correlation coefficient (CC), the coefficient of determination ($R^2$), and relative bias in percent (PBIAS) for both validation strategies (gap filling and one-step-ahead forecasting).

To assess skills of detecting large amounts of CO in the atmosphere, probability of detection (POD), false alarm ratio (FAR), and critical success index (CSI) have been calculated for whether pixels exceed the 0.9 quantile of the original data ($q_{0.9} = 0.0378971 \text{ mol m}^{-2}$). All error and detection metrics refer only to pixels that have been additionally left out but do not include predictions of pixels that were available in both X and Y. Definitions of all metrics are provided in the appendix.

We additionally calculated average computation times for predicting single blocks of the test dataset. For comparison, all predictions have been computed on an Intel Core i7-7700HQ CPU with 16 gigabytes of main memory, and a 512-gigabyte solid-state drive. However, the training of neural network models has been performed on a separate machine with an NVIDIA GeForce RTX 3090 GPU.

4. Results

a. Prediction performance

Table 2 presents error metrics for predictions of the independent test set using different models and both validation strategies (gap filling and one-step-ahead forecasting).

1) Gapfill

For gap filling, the statistical and STpconv models outperformed naïve approaches in terms of MAE, RMSE, CC, and $R^2$. STpconv$_L$ shows best RMSE, CC, and $R^2$ scores, whereas stmra predictions resulted in a slightly lower MAE. The higher RMSE value and relatively strong negative bias for stmra indicate difficulties in predicting outliers or extreme values, and generally dealing with the non-Gaussian distribution of the data. The performance of gapfill is similar to STpconv$_L$, presenting the best relative bias. STpconv$_S$ shows slightly worse prediction scores than gapfill and STpconv$_L$. Surprisingly, Conv3D scores are relatively close to simple time series interpolation and the strong relative bias suggests a tendency to underestimate.

The detection skills for predicting if values exceed the 0.9 quantile of the input data ($q_{0.9} = 0.0378971 \text{ mol m}^{-2}$)
reveal a similar characteristic for gap filling. Despite showing a tendency to underestimate, smr3a shows the highest probability of detection (POD), lowest false alarm ratio (FAR), and the best critical success index (CSI). gapfill, probability of detection (POD), lowest false alarm ratio reveal a similar characteristic for gap values similar to STpconv outperforms Conv3D and resulted in POD, FAR, and POD notice that in terms of detection skills, time series prediction that simply repeats the last available observation per pixel (last observation carried forward). However, series prediction that simply repeats the last available obser-

### 2) ONE-STEP-AHEAD FORECASTING

One-step-ahead forecasting led to slightly worse overall prediction scores. gapfill was not applicable because it could not find suitable neighborhoods in the completely missing time slice, whereas smr3a resulted in poor predictions even worse than using the global mean as prediction ($R^2 < 0$). To achieve better results, smr3a seems to require additional model fitting for one-step-ahead forecasting with a corresponding training dataset.

Both STpconv models outperformed other models, where the larger STpconvL model shows slightly better MAE, RMSE, CC, and $R^2$ values. Conv3D performs similar to time series prediction that simply repeats the last available observation per pixel (last observation carried forward). However, notice that in terms of detection skills, time series prediction outperforms Conv3D and resulted in POD, FAR, and POD values similar to STpconvL. Since the training dataset used only artificially added gaps as described in section 2c, none of the models have been explicitly trained to perform one-step-ahead forecasting.

### 3) COMPUTATION TIMES

Table 3 presents average computation times needed for predicting a single spatiotemporal block of size ($n_{lat} \times n_{lon} \times n_{time}$) = (128, 128, 16) and the time used for training/fitting a model before with regard to different models.

Interestingly, predictions with the smaller STpconvS model have been even slightly faster than simple time series interpolation, and approximately 300 and 3000 times as fast relative to smr3a and gapfill, respectively (all using on the same CPU and a single core only). The larger STpconvL took approximately 10 times as long for predicting a single block. In comparison with ordinary convolutions, the STpconv models require additional computations while applying and updating the binary masks. This was especially noticeable during model training, where 60 epochs for Conv3D and the comparable STpconvL were finished after approximately 31 and 49 min, respectively.

Computation times needed for model training/fitting are more difficult to compare and are only provided to give a rough understanding on the resources needed, because neural networks have been trained on a separate machine using an NVIDIA GeForce RTX 3090 GPU. In our experiments, less than one hour has been used for all models. While the neural network used 80% of the samples in the training dataset, we needed to randomly sample 10 blocks to achieve similar computation times to fit an smr3a model. The optimization time for smr3a is generally hard to estimate in advance and can vary strongly for different starting values, covariance functions, and other parameters (Appel and Pebesma 2020). Other methods did not require any model fitting at all.

Table 3. Computation times of model training/fitting ($T_{TRN}$) and predicting a single spatiotemporal block ($T_{PRED}$). Notice that the neural network models have been trained on a separate machine using an NVIDIA GeForce RTX 3090 GPU whereas model fitting for the smr3a approach used only 10 randomly selected blocks.

<table>
<thead>
<tr>
<th>Model</th>
<th>$T_{PRED}$ (s)</th>
<th>$T_{TRN}$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blockwise means</td>
<td>0.02</td>
<td>—</td>
</tr>
<tr>
<td>Time series interpolation</td>
<td>0.51</td>
<td>—</td>
</tr>
<tr>
<td>gapfill</td>
<td>1480</td>
<td>—</td>
</tr>
<tr>
<td>smr3a</td>
<td>160</td>
<td>1821</td>
</tr>
<tr>
<td>Conv3D</td>
<td>4.03</td>
<td>1857</td>
</tr>
<tr>
<td>STpconvL</td>
<td>4.17</td>
<td>2960</td>
</tr>
<tr>
<td>STpconvS</td>
<td>0.44</td>
<td>1977</td>
</tr>
</tbody>
</table>
produce quite different images, which is mostly visible in the southeastern area with large CO values. Conv3D and STpconv$_L$ show small “hotspots” in comparison with STpconv$_M$, where Conv3D values tend to drop faster toward the global mean. Visually, STpconv$_L$ predictions show larger spatial autocorrelations for higher CO values yet a similar smoothness relative to the other neural network models.

c. CO mapping

We used the small model (STpconv$_L$) to fill gaps in the original dataset and produce quasi-global (60°S–60°N) daily maps of total column CO for the time range of the data. Figure 6 shows a few days of the original incomplete and the corresponding filled images. Animated predictions can be found in the online supplemental material of this paper. To reduce artifacts at block boundaries, overlapping spatiotemporal blocks have been used as input: For each block, only inner observations are used as predictions, leaving out four pixels at the boundaries of the spatial axes and one pixel at the boundaries of the temporal axis. Adjacent spatiotemporal blocks hence overlap by two-times the number of ignored pixels per axis. Notice that Fig. 6 shows complete predictions, that is, the model output without combining with observations, where available. Hence, the results look relatively smooth and do not show any obvious artifacts at block boundaries. The general dynamics of the process seems well preserved. As expected, extremes tend to be underestimated. Interestingly, the gap-filled animation reveals some clearly visible pattern of extreme values above the southwestern Atlantic Ocean, mostly close to the South American coastline. These extreme values up to 0.7 mol m$^{-2}$ also occur in the original data but are only hardly visible because of the gaps.

5. Discussion

a. Prediction performance

Neural networks with spatiotemporal partial convolutional layers turned out to be able to efficiently fill gaps in atmospheric (Sentinel-5P) image time series. Overall, they outperformed simple blockwise mean prediction, time series interpolation, and a comparable neural network based on ordinary convolutional layers for both, gap filling and one-step-ahead forecasting, even though they have not been explicitly trained to perform the latter.

While achieved prediction errors for gap filling were similar to complex statistical models, computation times were faster by a factor of between 30 (STpconv$_L$ vs stmra) and 3000 (STpconv$_L$ vs gapfill). In terms of computation times for prediction, our smaller STpconv$_L$ model even outperformed native time series interpolation on the same CPU. For model training, a powerful GPU is required although training times have been less than 1 h in the experiments. Such GPUs are also available on cloud computing platforms for a few USD per hour (e.g., https://aws.amazon.com/de/ec2/instance-types/p3; accessed 20 May 2022), resulting in less than USD 3 total costs for training in our case.
All in all, our approach achieved good prediction scores for both gap filling and one-step-ahead forecasting at comparably low computation times. It is hence especially useful for processing large amounts of data.

As compared with the statistical approaches considered, predictions from the neural networks tend to look blurry. This is a well-known issue related to pixel-wise error losses such as MAE or RMSE, which this work does not address. However, it would be interesting to study how (i) the selection of hyperparameters, (ii) other model architectures, where e.g., input images are used at different resolutions to separate the process by frequencies, (iii) model ensembles trained with different losses, dropout, and other architectural hyperparameter values, and (iv) using different strategies to add artificial gaps to the input data can improve the reconstruction of small-scale variability. However, while there is certainly room for improvement, the original data are already noisy and, as an example, removing the typical vertical stripe pattern in Sentinel-5P imagery can be desired. One has to take into account whether the objective is to create realistically looking Sentinel-5P images, or if interest lies in the actual latent process of the variable (CO in our case), where smoothness to some extent might be a property of the phenomenon.

FIG. 5. Input, target, and predictions of six time slices from a single spatiotemporal block using (from top to bottom) gapfill, stmfa, Conv3D, STpconv, and STpconv. The time slices form a subset of a single spatiotemporal block from the test dataset; that is, none of the shown images and pixels have been used during training. Reported mean values refer to pixels of artificially added gaps, only. The images contain modified Copernicus Sentinel data (2021).
FIG. 6. (left) Quasi-global daily original dataset and (right) STpconv\textsubscript{3} model predictions at six consecutive days. Notice that original observations of available pixels have not been used to replace predicted values; that is, pure predictions are shown. Reported mean values $\mu$ refer to all visible pixels. The images contain modified Copernicus Sentinel data (2021).
**b. Model transferability and uncertainties**

As compared with the naive and statistical approaches, models that are based on partial convolutions come with the risk of excessive prediction errors under extrapolation conditions, that is, when the neural network is applied to data dissimilar to the input data in the training set (Meyer and Pebesma 2021), or if there are hardly any observations available in a block. More complex loss functions could improve the robustness of predictions. For example, terms that penalize if blockwise means of predictions deviate too much from actual means of available pixels could be added. The approach also does not provide uncertainty estimates such as prediction intervals as the statistical models do. Following Ehsani et al. (2022), it would be possible to add a Monte Carlo Dropout technique (Gal and Ghahramani 2016), where several predictions with random dropouts are generated and the standard deviation of predictions is used as a measure of prediction uncertainty. Alternatively, Zammit-Mangion and Wikle (2020) integrate a CNN for modeling spatiotemporal dynamics into a hierarchical statistical framework.

To assess prediction performance, the data were split into training, validation, and test sets by simple random sampling of spatiotemporal blocks, resulting in disjoint sets without any pixels being used in two or all three sets. However, blocks from the different sets may still come from the same sixteen day time period. In general, ensuring independence of the training and testing sets is not as trivial as it may seem by simply selecting different parts of the data. There might be more complex spatiotemporal dependencies like seasonality (e.g., when using the same month but different years in training and testing) that are ideally taken into account. We tried to reduce spatiotemporal dependencies by using relatively large spatiotemporal blocks and checking that only few pairs (approximately 0.1%) of training and testing samples are direct neighbors. Furthermore, we have applied the trained neural network models on 100 blocks from a completely different time period (December 2021) and found the prediction scores to be comparable or even slightly better (Table 4).

This study has not investigated how well models based on spatiotemporal partial convolutions work for other types of data such as optical imagery, or land surface temperatures. Xing et al. (2022) suggest that a similar method based on two-dimensional partial convolutions seems to work well for snow cover mapping, too. Similarly, it would be interesting to study how models trained on one variable can be used to predict other variables. Figure 7 exemplarily shows results when a model trained on CO observations is applied to NO2 data, after scaling to a similar value range. It should be noted that the result looks rather smooth because the underlying NO2 process has a smaller spatial scale than CO that was used to train the model. However, further experiments are needed to evaluate model transferability among other Sentinel-5P variables. To improve predictions, it might also be important to add other Sentinel-5P variables and/or further external variables such as elevation to the model.

**c. Model architecture and hyperparameter optimization**

Given the amount of hyperparameters, tuning can be time consuming and make the optimization of the model architecture, data preprocessing, and similar to specific datasets difficult in practical applications. In the presented experiments, we have found that data-related parameters such as the spatiotemporal block size had a stronger effect on prediction performance relative to architectural parameters such as the number of filters per layer. However, this might be different when using much larger sets of training data. Furthermore, it would be possible to use partial convolutional layers in other model architectures than the presented U-Net-like model. For example, it would be interesting to explore models following the idea of residual neural networks (He et al. 2016), where data at different spatial and/or temporal resolutions is provided as input to the model and residuals from filling gaps at lower resolution are recursively considered by partial convolutional blocks.

**d. Spatiotemporal dynamics from incomplete data**

DL-based models have become promising for modeling the dynamics of continuous spatiotemporal phenomena (De Bézénac et al. 2019; Zammit-Mangion and Wikle 2020; Rasp and Thuerey 2021; Keisler 2022). Since the abundance of gaps in satellite-derived observations makes their application often difficult, it would be very interesting to integrate partial convolutions into similar models applied on incomplete data. As a first experiment, one could replace convolutional layers in De Bézénac et al. (2019) with partial convolutional layers and study how the performance of forecasts changes with the amount of missing values.

**6. Conclusions**

This paper discussed the use of deep neural networks based on three-dimensional partial convolutional layers for efficiently filling gaps in satellite image time series.

We included spatiotemporal partial convolutions in a U-Net-like architecture and applied two models with different...
complexities to daily aggregated time series of quasi-global carbon monoxide observations from the satellite-based Sentinel-5P mission. To assess prediction errors and computational efficiency, two naive methods (blockwise means, time series interpolation), two statistical approaches (gapfill, stmra), and a U-Net-like neural network using ordinary convolutional layers with a reasonable background value (Conv3D) have been used for comparison. Model performance has been evaluated on an independent test dataset for normal gap filling as well as for one-step-ahead forecasting.

Our results indicate that our approach presents a computationally efficient method to fill gaps in incomplete satellite image time series with up to 85% missing values. Among all models, the suggested approach achieved the best overall prediction error scores (RMSE, CC, $R^2$) and was only slightly outperformed by the statistical approaches with regard to the detection of large values (POD, FAR, CSI).

For one-step-ahead forecasting, our approach provided the best prediction errors in terms of MAE, RMSE, CC, and $R^2$. In contrast to normal gap filling, the statistical models could not provide reasonable predictions but simply repeating the last available observation per time series did provide slightly better detection skills (POD, CSI).

Despite the additional effort to previously train models on a relatively powerful GPU, computing predictions of the presented approach turned out to be very efficient. Depending on the model size and architecture, we could even outperform naive time series interpolation on an identical CPU. Computing predictions for the statistical approaches was between 30 and 3000 times slower. As a result, the presented approach is well suited to process large amounts of data.

However, some challenges still need further consideration in future research and development. First, our implementation of spatiotemporal partial convolutions does not provide prediction uncertainties. Relatively simple techniques such as Monte Carlo dropout might render this possible but it is currently unclear how well they work for neural networks with partial convolutions as presented in this study.

Furthermore, models trained on pixel-wise error losses (MAE or MSE) tend to provide predictions that are smoother than input imagery. While this paper does not provide a solution to this problem, it would be interesting to study the effect of different model architectures and strategies to add artificial gaps on small-scale variability in the predictions. In our case, artificially added gaps mostly targeted at longer range spatiotemporal structures. Combining models trained on data with differently generated artificial gaps or using model ensembles may help to improve the prediction of small-scale variability. At the same time, the imagery used in this study is noisy and reproducing noise patterns like the vertical stripes may be undesirable to infer statements from the underlying (latent) process.

With simultaneous consideration of the achieved prediction scores, detection skills, and computational efficiency, the presented approach seems promising for the reconstruction of large satellite image time series as well as for future work on modeling spatiotemporal dynamics from incomplete data in a computationally efficient way.

Acknowledgments. This research has been funded by the Deutsche Forschungsgemeinschaft (DFG: German Research Foundation)—39661854. Thanks are given to Edzer Pebesma for comments and suggestions to improve the final paper.

Data availability statement. Data used for model training and validation have been made available on Zenodo (https://doi.org/10.5281/zenodo.6838651; Appel 2022). The dataset also
includes predictions from other models discussed in section 3d. To reproduce results, the method has been made available as open-source software at GitHub (https://github.com/appelmar/STpconv).

APPENDIX

Definitions of Validation Metrics

Let $\mathbf{Y}$ and $\mathbf{P}$ here be vectors of observations and predictions, respectively. We define the following prediction error metrics:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |Y_i - P_i|,$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (Y_i - P_i)^2},$$

$$\text{CC} = \frac{\sum_{i=1}^{n} (Y_i - \bar{Y})(P_i - \bar{P})}{\sqrt{\sum_{i=1}^{n} (Y_i - \bar{Y})^2 \sum_{i=1}^{n} (P_i - \bar{P})^2}},$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (Y_i - P_i)^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2}, \quad \text{and} \quad \sum_{i=1}^{n} (Y_i - \bar{Y})^2,$$

$$\text{PBIAS} = \frac{\sum_{i=1}^{n} P_i - Y_i}{\sum_{i=1}^{n} Y_i}.$$

Let TP, TN, FP, and FN be the number of true positives (hits), true negatives, false positives (false alarms), and false negatives (misses), respectively, from a binary classification. We use the following definitions of probability of detection (POD), false alarm rate (FAR), and critical success index (CSI) as detection skill scores:

$$\text{POD} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

$$\text{FAR} = \frac{\text{FP}}{\text{TP} + \text{FP}}, \quad \text{and} \quad \text{CSI} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}.$$

REFERENCES


He, K., X. Zhang, S. Ren, and J. Sun, 2016: Deep residual learning for image recognition. 2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, Institute of
Electrical and Electronics Engineers, 770–778, https://doi.org/10.1109/CVPR.2016.90.


